



Höhere Technische Bundeslehranstalt Kaindorf an der Sulm
Abteilung Robotik

Diplomarbeit
im Rahmen der Reife- und Diplomprüfung

Fiber Printer

Elias Gottsbacher
Nico Hütter
Jan Traußnigg

2024/2025

18.3.2025
Kaindorf an der Sulm

Betreuer: Dipl.-Ing. Thomas Jerman
Dipl.-Ing. Michael Lieschnegg, BSc
Ing. Dipl.-Ing. Werner Schöfmann
Projektpartner: XITEC Engineering GmbH

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung aller Autoren reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Die Autoren übernehmen keine Gewähr für die Funktionen einzelner Programme oder von Teilen derselben. Insbesondere übernehmen sie keinerlei Haftung für eventuelle aus dem Gebrauch resultierende Folgeschäden.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum

Elias Gottsbacher

Ort, Datum

Nico Hütter

Ort, Datum

Jan Traußnigg

Vorwort

Die vorliegende Diplomarbeit entstand im Rahmen der Reife- und Diplomprüfung an der Höheren Technischen Bundeslehranstalt Kaindorf an der Sulm und befasst sich mit der Entwicklung eines innovativen 3D-Drucksystems zur Integration von Fasern in den Druckprozess. Die Idee zu diesem Projekt entstand aus der Motivation, die mechanischen Eigenschaften additiv gefertigter Bauteile gezielt zu verbessern und gleichzeitig eine zugängliche Lösung für den faserverstärkten 3D-Druck zu schaffen. Die Umsetzung dieses Projekts erforderte eine interdisziplinäre Herangehensweise, die mechanische Konstruktion, Elektronikentwicklung und Softwareprogrammierung vereint. Die Zusammenarbeit innerhalb des Teams sowie mit externen Partnern spielte eine zentrale Rolle bei der erfolgreichen Realisierung des Prototyps. Insbesondere die Unterstützung durch die Firma XITEC Engineering GmbH war von großem Wert und ermöglichte eine praxisnahe Umsetzung der entwickelten Konzepte. Mit dieser Arbeit wird eine Grundlage für zukünftige Entwicklungen im Bereich des faserverstärkten 3D-Drucks geschaffen. Es bleibt zu hoffen, dass die erzielten Ergebnisse als Inspiration für weitere Forschungsarbeiten und Anwendungen in diesem Bereich dienen können.

Danksagung

Im Rahmen dieses Projekts wurde wertvolle Unterstützung durch verschiedene Personen und Institutionen geleistet, denen besonderer Dank gebührt. Unser aufrichtiger Dank gilt unseren Betreuern Thomas Jerman, Werner Schöfmann und Michael Lischegg, die mit ihrer fachlichen Expertise und engagierten Begleitung maßgeblich zur erfolgreichen Umsetzung beigetragen haben. Ein besonderer Dank richtet sich an die Firma XITEC Engineering GmbH, insbesondere an Michael Hartner-Stranimaier, der das Projekt von Beginn an förderte. Durch seine Unterstützung konnte die Konzeptentwicklung finanziert werden. Darüber hinaus bereicherte er das Vorhaben mit wertvollen Impulsen und ermöglichte ein Praktikum, das zur Vertiefung praktischer Kenntnisse beitrug. Zusätzlich danken wir Reinhard Unterweger für seine Unterstützung bei Simulationen sowie Kurt Zweityk für die fachkundige Begleitung in der Fertigung. Ebenso gilt unser Dank Udo Traubnigg für das sorgfältige Korrekturlesen. Darüber hinaus möchten wir uns bei zahlreichen weiteren Freund:innen und Unterstützer:innen bedanken, die mit ihrem Rat und ihrer Hilfe zur erfolgreichen Umsetzung des Projekts beigetragen haben. Zudem lieferte das Open-Source-Projekt GladiusSlicer (Ince, 2021) wertvolle Anregungen für die Entwicklung der Slicing-Software und ermöglichte die Integration bewährter Softwarekomponenten.

Zusammenfassung

Die Diplomarbeit FiberPrinter beschäftigt sich mit der Entwicklung eines innovativen 3D-Drucksystems, das es ermöglicht, Fasern direkt innerhalb des Hotends mit dem Filament zu vereinen und zu drucken. Im Gegensatz zu herkömmlichen Methoden, bei denen Fasern nachträglich in das gedruckte Bauteil eingebracht werden, integriert dieses System die Faserverstärkung bereits während des Druckvorgangs. Das Hauptziel des Projekts ist die Entwicklung eines funktionsfähigen Prototyps, der die Herstellung mechanisch optimierter Bauteile ermöglicht und zugleich einfach reproduzierbar ist. Zur Steuerung der Faserplatzierung wurde die Software FiberSlice entwickelt, die Anwender:innen eine präzise Kontrolle über den Druckprozess bietet. Neben der mechanischen Konstruktion des Druckkopfs umfasst die Arbeit auch die Entwicklung einer speziellen Elektronik zur Steuerung der Faserintegration. Diese basiert auf einem eigenentwickelten Breakoff-Board mit einem ATmega328P und einer USB-zu-Seriell-Schnittstelle. Die Firmware wurde so gestaltet, dass sie eine effiziente Kommunikation zwischen den einzelnen Komponenten gewährleistet. Das Projekt wurde in Zusammenarbeit mit der Firma XITEC Engineering GmbH realisiert und verfolgt das Ziel, eine Open-Source-Lösung für faserverstärkten 3D-Druck bereitzustellen.

Abstract

The FiberPrinter diploma thesis focuses on the development of an innovative 3D printing system that enables the fusion of fibers with filament directly within the hotend during the printing process. In contrast to conventional methods, where fibers are added to the printed component afterward, this system integrates fiber reinforcement during the printing process itself. The main goal of the project is to develop a functional prototype that allows the production of mechanically optimized components while being easily reproducible. To control fiber placement, the FiberSlice software was developed, providing users with precise control over the printing process. In addition to the mechanical design of the print head, the project also includes the development of specialized electronics to manage fiber integration. This system is based on a self-developed breakoff board featuring an ATmega328P and a USB-to-serial interface. The firmware is designed to ensure efficient communication between the individual components. The project was carried out in collaboration with XITEC Engineering GmbH and aims to provide an open-source solution for fiber-reinforced 3D printing.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Kurzbeschreibung des Themas und der Problemstellung	1
1.2 Aufgabenstellung	1
1.3 Aufbau der Arbeit	1
1.4 Methodik	1
1.5 Projektteam	2
1.5.1 Elias Gottsbacher	2
1.5.2 Nico Hütter	2
1.5.3 Jan Traußenigg	2
1.6 Entwicklungszeit	3
1.7 Übersicht der Meilensteine	3
1.8 Verwendete Software	4
2 Recherche	6
2.1 Definition der additiven Fertigung	6
2.2 Verbreitete 3D-Druckverfahren	6
2.3 Vorteile der Faserintegration im FDM-Druck	8
2.3.1 Mechanische Verbesserung durch Fasern	8
2.3.2 Vergleich von Kurzfasern und Endlosfasern	8
2.3.3 Unterschiede der Faserarten	8
2.4 Methoden der Faserintegration im 3D-Druck	9
2.5 Marktforschung	10
2.6 Schlussfolgerung	11
3 Konstruktion	12
3.1 Einleitung	12
3.2 Konstruktionsziele	12
3.3 Bauteilerklärung	13
3.3.1 Hotend	13
3.3.2 Heatbreak und Coldend	13
3.3.3 Schneidmechanismus	13
3.3.4 Faservorschub	13
3.3.5 Materialauswahl	13
3.4 Simulationen in Fusion 360 und SimScale	13
3.4.1 Einleitung	13
3.4.2 Zielsetzung	14
3.4.3 Rahmenbedingungen	14
3.4.4 Nutzen von Simulationen	15
3.5 Versionsverlauf	15
3.6 Version 1	16
3.6.1 Konstruktion der ersten Version	16
3.6.2 Thermische Analyse	17
3.6.3 Unterverisionen	18
3.6.4 Fazit	18
3.7 Version 2	19
3.7.1 Konstruktion der zweiten Version	19
3.7.2 Thermische Analyse	20
3.7.3 Unterverisionen	21
3.7.4 Fazit	22
3.8 Version 3	23
3.8.1 Konstruktion der dritten Version	23
3.8.2 Thermische Analyse	24
3.8.3 Unterverisionen	25
3.8.4 Fazit	25
3.9 Version 4	26
3.9.1 Konstruktion der vierten Version	26

3.9.2	Unterversionen	29
3.9.3	Fazit	29
3.10	Version 5	30
3.10.1	Konstruktion der fünften Version	30
3.10.2	Simulationen	33
3.10.3	Unterversionen	39
3.10.4	Fazit	40
3.11	Version 6	41
3.11.1	Konstruktion der sechsten Version	41
3.11.2	Thermische Simulation	47
3.11.3	Unterversionen	48
3.11.4	Fazit	49
3.12	Version 7 (aktuelle Version)	50
3.12.1	Konstruktion der siebten Version	51
3.12.2	Thermische Simulation	54
3.12.3	Unterversionen	55
3.12.4	Fazit	55
4	Elektronik	56
4.1	Elektronische Komponenten des Druckkopfes	56
4.2	Systemarchitektur der Elektronik	57
4.3	Anforderungen an die Elektronik	58
4.4	Lösungsansätze und Konzepte zur Datenübertragung	59
4.4.1	Konzept 1 zur Datenübertragung: I ² C-Verbindung	59
4.4.2	Konzept 2 zur Datenübertragung: USB-C-Verbindung	60
4.4.3	Entscheidungsmatrix für die Platinenwahl	60
4.5	Lösungsansätze und Konzepte zur Platinenbauform	62
4.5.1	Konzept 1 zur Platinenbauform: Gestapelte Platine	62
4.5.2	Konzept 2 zur Platinenbauform: Einfache Platine	62
4.5.3	Entscheidungsmatrix für die Platinenbauform	63
4.6	Modulare Aufteilung der Elektronik	64
4.6.1	Mechanische Stabilität	64
4.6.2	Leistungs-Modul	64
4.6.3	Lüfter-Modul	64
4.6.4	Motoren-Modul	65
4.7	Auswahl der Komponenten	65
4.7.1	Spannungsversorgung und Strombedarf	65
4.7.2	Übersicht der Teilbereiche	67
4.7.3	Einplatinencomputer – Raspberry Pi 3 Model B+	68
4.7.4	Mikrocontroller – ATMEGA328P-AU	69
4.7.5	Schrittmotor – 14HM08-0504S	70
4.7.6	Servo – MG90S	72
4.7.7	Spannungswandler 24 V auf 12 V – TPS56837HRPAR	73
4.7.8	Spannungswandler 12 V auf 5 V – TPS565208DDCR	74
4.8	Schaltungsdesign	75
4.8.1	Leistungseingang und Schutzschaltung	75
4.8.2	USB-Datenverbindung und Schnittstellenwandler	76
4.8.3	24 V auf 12 V Spannungswandler	77
4.8.4	12 V auf 5 V Spannungswandler	78
4.8.5	Lüftersteuerung	79
4.9	Simulationen und Berechnungen	81
4.9.1	Simulation Verpolungsschutz	81
4.9.2	Simulation der 12 V-Spannungsregelung	83
4.9.3	Simulation der 5 V-Spannungsregelung	85
4.10	Leiterplatten-Design (PCB-Layout)	87
4.10.1	Allgemein	87
4.10.2	Layout des 12 V-Spannungsreglers	88
4.10.3	Layout des 5V-Spannungsreglers	89

4.10.4	Zusammenföhrung der Module	90
4.10.5	Beschaffung	91
4.11	Lötprozess	92
5	Firmware	93
5.1	Einleitung	93
5.2	Verfügbare Firmware-Optionen	93
5.3	Auswahl der Firmware	93
5.4	Klipper: Definition und Funktionsweise	93
5.5	Installation von Klipper	94
5.5.1	Verwaltung und Aktualisierung mit KIAUH	94
5.6	Klippy: Die zentrale Steuerlogik von Klipper	94
5.6.1	Interaktion zwischen Klippy und dem Mainboard	94
5.7	Mainsail: Webinterface zur Steuerung von Klipper	94
5.8	Moonraker: API-Server für Mainsail	95
5.9	Klipper Display: Lokale Steuerung von Klipper	95
5.10	Konfiguration von Klipper	95
5.10.1	Aufbau der Konfigurationsdatei	95
5.10.2	Anpassung der Konfiguration	95
5.10.3	Makros und erweiterte Funktionen	95
5.10.4	Erweiterung der Funktionalität	96
5.11	Klipper-Plugin zur Steuerung des Schneidervatos	96
5.11.1	Konfiguration des Plugins	96
5.11.2	PWM-Steuerung des Servos	96
5.11.3	Implementierung des Plugins	97
5.12	Klipper-Plugin zur Steuerung des Fasereextruders	97
5.12.1	Konfiguration des Fasereextruders	97
5.12.2	Integration in den Toolhead von Klippy	97
5.12.3	Bewegungssteuerung	98
6	Slicing-Software	99
6.1	Einleitung	99
6.2	Lösungsansätze	99
6.2.1	Weiterentwicklung eines bestehenden Open-Source-Slicers	99
6.2.2	Entwicklung eines Slicers von Grund auf ("Ground-up")	99
6.2.3	Vergleich der Ansätze	100
6.3	Gründe für die komplette Neuentwicklung eines Slicers	100
6.4	Verwendung von Rust	100
6.4.1	Sicherheit und Fehlervermeidung	100
6.4.2	Leistung und Effizienz	100
6.4.3	Lernen und moderne Programmierung	100
6.5	Entwicklung mit Rust	100
6.5.1	Entwicklungsumgebungen	100
6.5.2	Rust-Bibliotheken und der Paketmanager Cargo	101
6.5.3	Rust-Book und Dokumentation	102
6.6	Bibliotheken	102
6.6.1	Benutzeroberfläche	102
6.6.2	Rendering	103
6.6.3	Mathematik und Vektorrechnungen	104
6.6.4	Fenster- und Eingabeverwaltung	104
6.6.5	Asynchrone Verarbeitung	104
6.6.6	Weitere verwendete Crates	105
6.7	Softwarearchitektur	106
6.7.1	Start der Applikation	106
6.7.2	Event-Loop und Steuerung der Applikation	107
6.7.3	Initialisierung der Module	107
6.7.4	Erstellung der Module mit dem Adapter-Trait	108
6.8	Benutzeroberflächen	108
6.8.1	Funktionalität des UiAdapter	108

6.8.2	Struktur des UiAdapter	108
6.8.3	Die Implementierung des UiAdapter	108
6.8.4	Event-Verarbeitung im UiAdapter	109
6.8.5	Rendering der UI	109
6.9	Benutzeroberflächen-Komponenten	110
6.9.1	Anzeige der Benutzeroberfläche	110
6.9.2	Verarbeitung von Toast-Nachrichten	111
6.9.3	Viewport-Berechnung für das Rendering	111
6.9.4	Interaktion mit Werkzeugen und Addons	112
6.9.5	Tools (Steuerbare UI-Fenster)	112
6.9.6	Komponenten (UI-Strukturelemente)	112
6.9.7	Addons (3D-Interaktionselemente)	113
6.10	Transformationen und Manipulationen für 3D-Objekte	113
6.10.1	Darstellung der Transformationswerkzeuge	114
6.10.2	Transformation von 3D-Objekten	115
6.11	Effiziente Anzeige von G-Code	116
6.11.1	Implementierung des EfficientReader	117
6.11.2	Anzeige von G-Code mit dem EfficientReader	117
6.12	Kamera	118
6.12.1	Funktionalität des CameraAdapter	118
6.12.2	Die Implementierung des CameraAdapter	118
6.12.3	Verarbeitung der Kamera-Ereignisse	118
6.13	Orbit-Kamera	119
6.13.1	Berechnung der Kameraposition	119
6.13.2	Berechnung der View- und Projection-Matrizen	120
6.13.3	Steuerung der Kamera	121
6.13.4	Zoom-Funktion	121
6.13.5	Einschränkungen der Kamerabewegung	121
6.13.6	Automatische Anpassung der Entfernung	121
6.14	Der Kamera-Kontroller	122
6.14.1	Implementierung des CameraController	122
6.14.2	Verarbeitung von Window-Events	122
6.14.3	Verarbeitung von Geräte-Events	123
6.15	Visualisierung	124
6.15.1	Funktionalität des RenderAdapter	124
6.15.2	Implementierung	124
6.15.3	Rendering-Prozess	125
6.15.4	Fensterevent-Handling	125
6.15.5	Vertex- und Texturverarbeitung	126
6.15.6	Texturen und Framebuffer	127
6.15.7	Erstellung von Tiefentexturen	128
6.15.8	Laden und Verarbeiten von Bildtexturen	128
6.15.9	Pipeline Erstellung und Management	129
6.15.10	Render Descriptor	130
6.16	Modell und Transformation	131
6.16.1	Transformationen	132
6.16.2	Rendering der Modell-Struktur	133
6.17	Werkzeugpfad-Visualisierung	134
6.17.1	Struktur und Hierarchie	134
6.17.2	Erstellung eines TraceTree	135
6.17.3	Kollisionen und Auswahl	135
6.17.4	Interaktive Funktionen	136
6.17.5	Rendering	136
6.17.6	Erzeugung und Verwaltung der Werkzeugpfad-Meshes	136
6.17.7	Berechnung des Querschnitts (TraceCrossSection)	137
6.17.8	Berechnung der Querschnittsverschiebung	137
6.17.9	Berechnung der Mesh-Vernetzung	137
6.17.10	Verbindungsnetz zwischen Abschnitten	138

6.17.11 Generierung der Werkzeugpfade	138
6.17.12 Verwaltung der Pfade	139
6.17.13 Erstellung eines <i>SlicedObject</i>	139
6.17.14 Integration mit <i>TraceMesher</i>	140
6.17.15 Kollisionserkennung mit <i>HitboxNode</i>	140
6.18 Shader für Werkzeugpfade	141
6.18.1 Vertex Shader	141
6.18.2 Sichtbarkeitsprüfung	142
6.18.3 Datenstrukturen	142
6.18.4 Fragment Shader	142
6.19 Verwaltung und Rendering geslicerter Objekte	143
6.19.1 Rendering der Werkzeugpfade	143
6.19.2 Laden und Verarbeitung geslicerter Daten	143
6.19.3 Export der Werkzeugpfade als G-Code	144
6.19.4 Steuerung der Sichtbarkeit von Pfaden	144
6.20 Verwaltung und Rendering von CAD-Modellen	145
6.20.1 Rendering der CAD-Modelle	145
6.20.2 Laden von CAD-Dateien	145
6.20.3 Verarbeitung des CAD-Modells	146
6.21 Auswahl von Objekten	147
6.21.1 Umwandlung der Mauskoordinaten in Normalized Device Coordinates (NDC)	147
6.21.2 Umwandlung in Clip-Space-Koordinaten	147
6.21.3 Rücktransformation in Eye-Space	148
6.21.4 Transformation in Weltkoordinaten	148
6.21.5 Berechnung der Schnittpunkte mit Ebenen	148
6.21.6 Hierarchische Kollisionserkennung mit Hitboxen	149
6.21.7 Verwendung einer Prioritätswarteschlange für Kollisionen	150
6.22 Selektierer für Werkzeugpfade und Modelle	151
6.22.1 Berechnung der Achsenausrichtenden Begrenzungsbox (AABB)	151
6.22.2 Transformation von Objekten	151
6.22.3 Erstellung der Auswahlbegrenzung	152
6.22.4 Löschen und Aktualisieren der Auswahl	152
6.23 Slicer-Algorithmus	153
6.23.1 Slicing: Zerlegung in horizontale Schichten	154
6.23.2 Berechnung der Schnittpunkte einer Schicht mit Dreiecken	154
6.23.3 Erstellung der Schnittpolygone	155
6.23.4 <i>Slice</i> -Struktur und Slicing-Repräsentation	156
6.23.5 Erzeugung eines <i>Slice</i>	157
6.23.6 Berechnung der Druckzeit	158
6.24 Slicer-Einstellungen	158
6.24.1 Grundlegende Einstellungen	158
6.24.2 Erweiterte Einstellungen und Anpassung	159
6.24.3 Validierung von Einstellungen	159
6.25 Plotter und Bewegungserzeugung	160
6.25.1 Bewegungsplanung und Druckpfade	160
6.25.2 Erzeugung von Perimetern	161
6.25.3 Bestimmung des Bewegungstyps in der Perimetergenerierung	161
6.25.4 Optimale Positionierung des Nahtpunkts (Seam)	162
6.25.5 Partielle vs. vollständige Flächenfüllung	162
6.25.6 Ermittlung der Spurfläche	164
6.26 Zusammenfassen von Faserbewegungen	165
6.26.1 Einzelne Faserbewegungen verarbeiten	165
6.26.2 Faserbewegungen in normale Bewegungen umwandeln	166
6.26.3 <i>FiberChain</i> -Struktur und deren Methoden	166
6.26.4 Rückwärtssuche des Schnittpunkts	167
6.27 Maskierungsfunktionen für das Slicing	168
6.27.1 Mask-Erstellung und Transformation	168
6.27.2 Beschneiden des maskierten Objekts	169

6.27.3 Zufällige Variation der Maskenbereiche	169
6.28 Druckpfadtyp und Farbzuordnung	170
6.28.1 Farbauswahl (<i>into_color_vec4</i>)	170
6.28.2 MoveType-Integration	170
6.29 Command-Aufbau und Aufgaben	171
6.29.1 Move-Varianten mit Extrusion	171
6.29.2 Weitere Bewegungsbefehle	171
6.29.3 Systembefehle	172
6.29.4 Filament- und Faser-Abfrage	172
6.30 Generierung und Speicherung von G-Code	173
6.30.1 Datenstruktur für G-Code	173
6.30.2 Schreiben von G-Code	174
6.30.3 Generierung von G-Code	175
6.30.4 Bewegungen ohne Extrusion	175
6.30.5 Bewegungen mit Extrusion	175
6.30.6 Faserverstärkte Bewegungen mit Schnitt	176
7 Prototypfertigung und Validierung	178
7.1 Einleitung zur Prototypfertigung	178
7.1.1 Zeitlicher Ablauf der Prototypfertigung	178
7.2 Einleitung zur Validierung	179
7.2.1 Zeitlicher Ablauf der Validierung	179
7.3 SLA-Druck Hotendprototyp	180
7.4 Schneidmechanismus	182
7.4.1 Fertigung	182
7.4.2 Validierung	182
7.5 Prototyp 1	185
7.5.1 Fertigung	185
7.5.2 Validierung	185
7.6 Prototyp 2	188
7.6.1 Fertigung	188
7.6.2 Validierung	189
8 Lastenheft	192
8.1 Funktionale Anforderungen	192
8.1.1 Anforderungen Druckqualität	192
8.1.2 Hotend	194
8.1.3 Coldend	196
8.1.4 Plattformunabhängige Integration	198
8.1.5 Software	200
8.1.6 Firmware	203
9 Anhang	204
9.1 Simulationslasten	204
9.2 Materialliste	205
9.3 Stückliste	206
9.4 Vollständiger Schaltplan	207
9.5 Vollständiges Layout	212
9.6 Datenblatt des DC-Motor-Controllers TPM16050	215
9.7 Layout Empfehlungen aus dem Datenblatt des TPS56387	216
9.8 Layout Empfehlungen aus dem Datenblatt des TPS565208	217
9.9 12 V-Schaltregler Simulation	218
9.10 5 V-Schaltregler Simulation	223
9.11 Bildschirmaufnahmen (Slicing-Software)	228
10 Literaturverzeichnis	229
11 Abbildungsverzeichnis	232

12 Tabellenverzeichnis	235
13 Programmausdrücke	236
14 Abkürzungsverzeichnis	239

INHALTSVERZEICHNIS

1 Einleitung

1.1 Kurzbeschreibung des Themas und der Problemstellung

Das Thema dieser Arbeit ist die Entwicklung eines 3D-Druckkopfes mit der Fähigkeit, Endlosfasern in das Filament zu integrieren. Ziel ist es, die mechanischen Eigenschaften der gedruckten Bauteile durch die Verstärkung mit Endlosfasern erheblich zu verbessern.

Die Motivation für dieses Projekt liegt in der Herausforderung, einen kostengünstigen Druckkopf zu entwickeln, der in der Lage ist, Fasern direkt im heißen Ende des Druckkopfes mit dem Filament zu vereinen. Derzeit existieren am Markt bereits Unternehmen, die den Druck mit Endlosfasern als Dienstleistung anbieten. Allerdings gibt es keine weit verbreitete und kosteneffiziente Technologie, die es ermöglicht, diese Technik in handelsüblichen 3D-Druckern zu nutzen.

Unterschiedliche Methoden zur Integration von Endlosfasern in Filament sind bereits bekannt. Ebenso existieren Slicer-Softwarelösungen, die eine Steuerung der Faserverlegung ermöglichen. Die zentrale Fragestellung dieser Arbeit ist jedoch, ob ein solches System kosteneffizient realisiert werden kann, indem Filament und Fasern bereits im heißen Ende des Druckkopfes zusammengeführt werden.

1.2 Aufgabenstellung

Das Hauptziel dieser Arbeit ist die Entwicklung eines Druckkopfes, der die Endlosfasern während des Druckvorgangs in das Filament integriert. Dazu gehört die Entwicklung einer Platine zur Steuerung der notwendigen Aktoren und Sensoren sowie die Programmierung eines Slicers mit einer Benutzeroberfläche, die eine gezielte Faserverlegung erlaubt. Die Endlösung soll mit handelsüblichen 3D-Druckern kompatibel sein, um eine möglichst breite Anwendung zu ermöglichen. Genauere technische Anforderungen sind im Lastenheft definiert.

Diese Arbeit ist von besonderer Relevanz für die Maker-Community und technikaffine Anwender:innen, die ihre 3D-Drucker durch einen kostengünstigen, faserverlegenden Druckkopf erweitern möchten. Das Projekt ist als Open-Source-Lösung konzipiert, um eine möglichst breite Verfügbarkeit zu gewährleisten und interessierten Nutzer:innen eine eigenständige Nachbau- und Weiterentwicklungsmöglichkeit zu bieten. Als Endprodukt wird ein funktionaler Druckkopf samt zugehöriger Platine entwickelt, der einfach auf bestehende 3D-Drucker montiert werden kann. Ergänzend dazu wird eine Softwarelösung entwickelt, die eine präzise Steuerung der Faserverlegung ermöglicht.

Die Arbeit basiert auf bestehenden Erkenntnissen zur Funktionsweise von 3D-Druckköpfen, der typischen Benutzeroberfläche gängiger Slicer-Software sowie den elektrischen und mechanischen Anforderungen eines Druckkopfes. Zur Validierung der Entwicklung werden Testdrucke durchgeführt, bei denen ein mit dem neuen System erstelltes Bauteil mit einem herkömmlichen, nicht faserverstärkten Druck verglichen wird.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in mehrere zentrale Abschnitte: Zunächst erfolgt eine ausführliche Recherche zu bestehenden Technologien und wissenschaftlichen Erkenntnissen im Bereich des Endlosfaserdrucks. Anschließend wird die Konstruktion des Druckkopfes beschrieben, wobei ein besonderer Fokus auf eine effiziente Wärmeentkopplung zwischen Hotend und Coldend sowie eine optimierte Kühlung gelegt wird. Der nächste Abschnitt behandelt den Prototypenbau und die Validierung durch Testdrucke. Darauf folgen die Entwicklung der zugehörigen Elektronik sowie die Erstellung der Firmware und der Softwarelösung für den Slicer. Jeder dieser Abschnitte enthält eine detaillierte Beschreibung der entwickelten Lösungen und deren Implementierung.

1.4 Methodik

Zur Erarbeitung der theoretischen Grundlagen wurden wissenschaftliche Artikel analysiert, die sich mit dem Thema des Endlosfaserdrucks befassen. Zudem wurde der Kontakt zu den Autor:innen dieser Arbeiten gesucht, um bestehende Lösungsansätze besser zu verstehen.

Die Konstruktion des Druckkopfes erfolgte iterativ, wobei verschiedene Designvarianten getestet und optimiert wurden. Besondere Herausforderungen bestanden in der Wärmeentkopplung des Hotends vom Coldend sowie der effizienten Kühlung des Coldends. Die Elektronik wurde in enger Abstimmung mit der Konstruktion und der Firmware entwickelt, um eine reibungslose Integration zu gewährleisten.

Für die Softwareentwicklung wurden verschiedene Bibliotheken evaluiert, um eine geeignete Grundlage für den Slicer zu identifizieren. Die finale Implementierung der Software basiert auf der vielversprechendsten Lösung, die eine flexible und präzise Steuerung der Faserverlegung ermöglicht.

1.5 Projektteam

Das Projektteam setzt sich aus folgenden Personen zusammen:

1.5.1 Elias Gottsbacher



E-Mail: goteln20@htl-kaindorf.at

Aufgabenbereiche: Software und Firmware

Betreuer: Dipl.-Ing. Michael Lieschnegg, BSc

1.5.2 Nico Hütter



E-Mail: huenin20@htl-kaindorf.at

Aufgabenbereiche: Konstruktion, Fertigung und Firmware

Betreuer: Ing. Dipl.-Ing. Werner Schöfmann

1.5.3 Jan Traußnigg



E-Mail: trajan20@htl-kaindorf.at

Aufgabenbereiche: Elektronik, Projektplanung und Recherche

Betreuer: Dipl.-Ing. Thomas Jerman

1.6 Entwicklungszeit

Die Gesamtarbeitszeit innerhalb des Projekts verteilte sich auf die einzelnen Mitglieder wie folgt:

- Elias Gottsbacher: 587 Stunden
- Nico Hütter: 360 Stunden
- Jan Traußnigg: 220 Stunden

Eine schematische Darstellung der Aufgabenverteilung ist in Abbildung 1 ersichtlich.

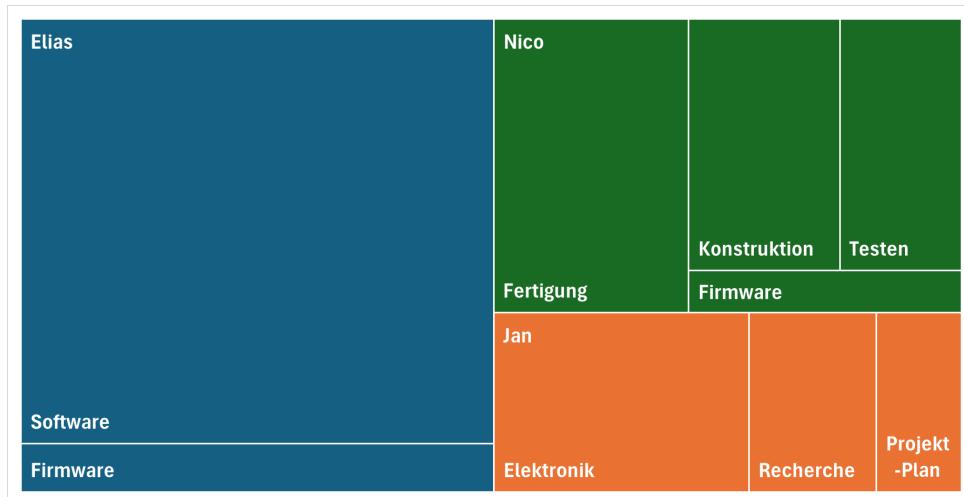


Abb. 1: Grafische Darstellung der Aufgabenverteilung innerhalb des Projekts.

1.7 Übersicht der Meilensteine

Tabelle 1 zeigt die wichtigsten Meilensteine des Projekts mit den geplanten Fertigstellungsterminen.

Tabelle 1: Meilensteine des Projekts

Mechanische Entwicklung	
Konzept-Konstruktion des Druckkopfes	1.9.2023
Simulation des Druckkopfes	1.10.2023
Entwicklung des Schneidmechanismus	1.5.2024
Fertigung des ersten Prototyps des Druckkopfes	1.8.2024
Optimierung des Druckkopfes	1.11.2024
Fertigung des zweiten Prototyps des Druckkopfes	1.12.2024
Elektronische Entwicklung	
Konzept für Elektronik	1.1.2024
Auswahl der Komponenten und Entwicklung des Schaltplans	1.6.2024
Entwicklung des Platinenlayouts (PCB) und Abstimmung mit Konstruktion	1.10.2024
Implementierung der Firmware	1.12.2024
Softwareentwicklung	
Entwicklung der Slicing-Software FiberSlice	1.5.2024
Implementierung der Steuerung für den Schneidmechanismus	1.8.2024
Optimierung der Faserverlegungsalgorithmen	1.12.2024
Validierung und Optimierung	
Erstes gedrucktes Bauteil mit Faserverstärkung	1.1.2025
Dokumentation der Diplomarbeit und Optimierung	1.3.2025

1.8 Verwendete Software

Autodesk Inventor

Autodesk Inventor ist eine professionelle CAD-Software für 3D-Konstruktion mit Fokus auf parametrische Modellierung und Baugruppendesign. Die Software wird zur mechanischen Konstruktion des Druckkopfes genutzt.



Abb. 2: Autodesk Inventor

Fusion 360

Fusion 360 ist eine CAD/CAM-Software für 3D-Modellierung und Fertigung. Sie wird zur Erstellung von thermischen Simulationen und CAM-Programmen für die mechanischen Komponenten des Druckkopfes verwendet.



Abb. 3: Fusion 360

SimScale

SimScale ist eine Cloud-basierte Simulationssoftware für Strömungs-, Struktur- und Wärmeübertragungssimulationen. Sie wird zur Durchführung von Strömungs-simulationen für die Kühlung des Druckkopfes verwendet.



Abb. 4: SimScale

KiCad

KiCad ist eine Open-Source-Software-Suite für die Erstellung elektronischer Schaltpläne und Platinenlayouts. KiCad wird zur Entwicklung der Schaltpläne und des PCB-Layouts der benötigten Steuerungselektronik für den Druckkopf verwendet.



Abb. 5: KiCad

Webench Power Designer

Webench Power Designer ist eine Webanwendung von Texas Instruments, die zur Auswahl und Simulation von Spannungswandlern verwendet wird. Sie wird zur Auswahl und Simulation der Spannungswandler für die Steuerungselektronik des Druckkopfes verwendet.



Abb. 6: Webench Power Designer

LTspice

LTspice ist ein leistungsfähiges Schaltungssimulations-Programm von Linear Technology, das auf dem SPICE-Simulator basiert. Es bietet die Möglichkeit, elektronische Schaltungen zu simulieren und unterstützt den Import von Bauteilmodellen verschiedener Hersteller. Es wird zur Simulation Steuerungselektronik des Druckkopfes verwendet.



Abb. 7: LTspice

WireViz

WireViz ist ein Tool zur Visualisierung von Verdrahtungsdiagrammen. Es wird zur Erstellung von Verdrahtungsdiagrammen für die Steuerungselektronik des Druckkopfes verwendet.

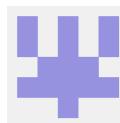


Abb. 8: WireViz

Visual Studio Code

Visual Studio Code ist ein leistungsstarker, erweiterungsfähiger Quellcode-Editor mit Debugging- und Git-Funktionalität. Er wird als Entwicklungsumgebung für den Slicer verwendet.



Abb. 9: Visual Studio Code

GeoGebra

GeoGebra ist eine dynamische Mathematiksoftware für Geometrie, Algebra, Statistik und Analysis. Sie wird zur Erstellung mathematischer Modelle und zur Visualisierung geometrischer Zusammenhänge für die Entwicklung des verwendet.

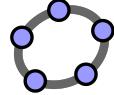


Abb. 10: GeoGebra

GitHub

GitHub ist eine Plattform zur Versionskontrolle und Zusammenarbeit, die auf Git basiert. Sie wird verwendet, um den Quellcode der Software zu speichern und zu verwalten.



Abb. 11: GitHub

MiniProtocol

MiniProtocol ist ein selbst entwickeltes Programm zur Dokumentation und Zeitverwaltung. Es wird zur Dokumentation der Arbeitsschritte, zur Zeiterfassung und zur automatischen Sicherung von Arbeitsordnern auf OneDrive verwendet.



Abb. 12: MiniProtocol

YouTrack

YouTrack ist ein webbasiertes Tool zur Verwaltung von Projekten und Aufgaben. Es wird zur Verwaltung der Aufgaben und des Fortschritts der Diplomarbeit verwendet. Test test test test test



Abb. 13: YouTrack

L^AT_EX

L^AT_EX ist ein Satzsystem, das für die Erstellung von wissenschaftlichen Dokumenten, Büchern und Artikeln verwendet wird. Es wird für die Dokumentation der Diplomarbeit verwendet.



Abb. 14: L^AT_EX

yEd

yEd ist ein Editor für Diagramme, der zur Erstellung von Flussdiagrammen und Schaltplänen verwendet wird. Er wird zur Erstellung von Diagrammen für die Dokumentation der Diplomarbeit verwendet.



Abb. 15: yEd

2 Recherche

2.1 Definition der additiven Fertigung

Der 3D-Druck ist ein Fertigungsverfahren, bei dem Objekte durch schichtweises Auftragen von Material erstellt werden. Im Gegensatz zu subtraktiven Fertigungsverfahren, wie dem Fräsen oder Drehen, wird das Material gezielt hinzugefügt, um die gewünschte Geometrie zu erreichen. (Wikipedia, 2025)

Die additive Fertigung bietet eine hohe Gestaltungsfreiheit, da komplexe Strukturen, innenliegende Kanäle und personalisierte Designs ohne großen Mehraufwand hergestellt werden können. Dies macht sie insbesondere für die Luft- und Raumfahrt, Medizintechnik und den Prototypenbau interessant. (Gibson et al., 2015)

2.2 Verbreitete 3D-Druckverfahren

Innerhalb der additiven Fertigung existieren verschiedene Verfahren, die sich durch das verwendete Material und die Art der Verarbeitung unterscheiden. Zu den wichtigsten zählen:

Fused Deposition Modeling (FDM)

Das **Fused Deposition Modeling (FDM)** ist das am weitesten verbreitete 3D-Druckverfahren aufgrund seiner Kosteneffizienz und einfachen Handhabung. Dabei wird ein thermoplastisches Filament durch eine beheizte Düse extrudiert und Schicht für Schicht auf eine Bauplattform aufgetragen. (L. C. et al., 2020) Die wichtigsten Eigenschaften sind:

- **Vorteile:** Geringe Kosten, einfache Bedienung, große Materialvielfalt
- **Nachteile:** Sichtbare Schichtlinien, begrenzte mechanische Festigkeit
- **Materialien:** PLA, ABS, PETG, Nylon, TPU

Die folgende Abbildung veranschaulicht das Verfahren: Ein thermoplastisches Filament wird durch eine beheizte Düse (Extruder) geschmolzen und schichtweise auf die Bauplattform aufgetragen, wodurch ein dreidimensionales Objekt entsteht.

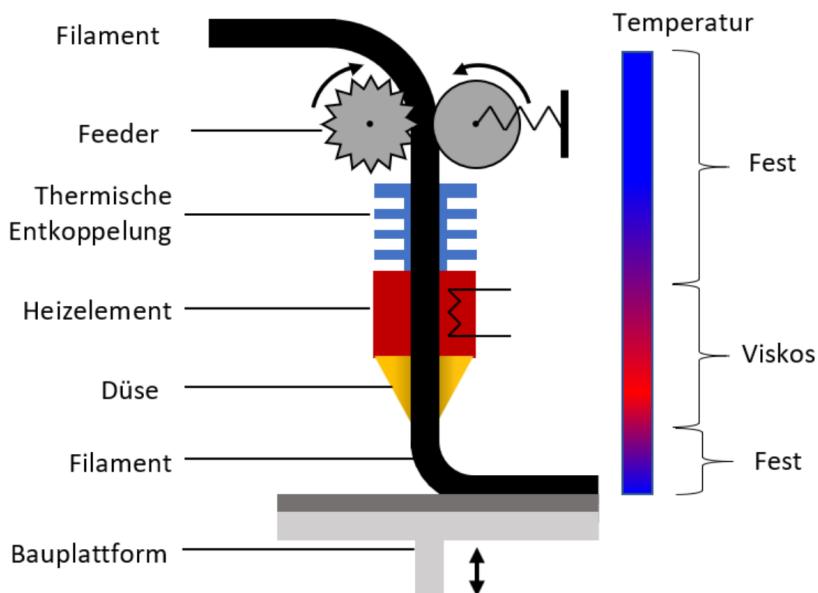


Abb. 16: Schematische Darstellung des FDM-Druckverfahrens (nach Institut für Fertigungstechnik, TU Graz, 2025)

Stereolithografie (SLA)

Bei der **Stereolithografie (SLA)** handelt es sich um ein harzbasiertes Verfahren, bei dem flüssiges Photopolymer durch UV-Licht ausgehärtet wird. Dadurch entstehen hochpräzise Drucke mit einer sehr feinen Oberflächenstruktur. SLA eignet sich besonders für Anwendungen, die hohe Detailgenauigkeit erfordern, wie z. B. Zahntechnik und Schmuckdesign. (Jariwala & Ding, 2019)

- **Vorteile:** Hohe Auflösung, glatte Oberflächen, feine Details
- **Nachteile:** Teure Materialien, Nachbearbeitung erforderlich, empfindlich gegenüber UV-Licht
- **Materialien:** Photopolymere (Harze)

Selektives Lasersintern (SLS)

Das **Selektive Lasersintern (SLS)** nutzt einen Hochleistungslaser, um feines Pulver (meist Nylon oder Metall) gezielt zu verschmelzen. Es ermöglicht die Fertigung komplexer Geometrien ohne Stützstrukturen, da das umgebende Pulver als Träger dient. SLS wird häufig in der Luft- und Raumfahrt sowie für funktionale Prototypen eingesetzt. (Kruth, 2005)

- **Vorteile:** Hohe Stabilität, keine Stützstrukturen nötig, komplexe Geometrien möglich
- **Nachteile:** Hohe Maschinenkosten, aufwendige Nachbearbeitung
- **Materialien:** PA12, Alumide, TPU, Metalle

2.3 Vorteile der Faserintegration im FDM-Druck

Ein vielversprechender Ansatz zur Verbesserung der mechanischen Eigenschaften von FDM-Drucken ist die Integration von Verstärkungsfasern. Hierbei werden Fasern aus verschiedenen Materialien in die Matrix eingebettet, wodurch sich die Festigkeit und Steifigkeit erheblich verbessern. (Tian, 2016)

2.3.1 Mechanische Verbesserung durch Fasern

Die Hauptvorteile der Faserintegration im FDM-Druck umfassen:

- **Erhöhte Festigkeit und Steifigkeit:** Fasern erhöhen die Belastbarkeit und sorgen für eine bessere mechanische Performance. (S. L. et al., 2012)
- **Geringere Dichte bei hoher Stabilität:** Im Vergleich zu Vollmaterialien sind faserverstärkte Kunststoffe leichter, aber dennoch hoch belastbar.

2.3.2 Vergleich von Kurzfasern und Endlosfasern

In der additiven Fertigung werden zur Verstärkung von Kunststoffen sowohl Kurzfasern als auch Endlosfasern eingesetzt. Beide Varianten beeinflussen die mechanischen Eigenschaften der hergestellten Bauteile unterschiedlich.

Kurzfasern sind kurze Stücke faserigen Materials, deren Länge von wenigen Millimetern bis zu Zentimetern variieren kann. Sie werden in die Kunststoffmatrix eingemischt und dienen als Verstärkung im gesamten Material. Aufgrund ihrer zufälligen Ausrichtung innerhalb der Matrix bieten sie eine isotrope Verstärkung, die die Steifigkeit und Festigkeit des Materials erhöht. Die Verarbeitung von Kurzfasern ist weniger komplex und kostengünstiger, was sie für Anwendungen mit geringeren mechanischen Anforderungen attraktiv macht. (3Dnatives, 2024)

Endlosfasern hingegen sind lange Stränge, die sich über die gesamte Länge des Bauteils erstrecken. Sie werden während des 3D-Druckprozesses kontinuierlich in die Kunststoffmatrix integriert. Diese gezielte Platzierung ermöglicht eine signifikante Erhöhung der mechanischen Eigenschaften, da die Fasern entlang der Hauptbelastungsrichtungen ausgerichtet werden können. Bauteile mit Endlosfaserverstärkung erreichen Festigkeiten, die mit denen von Metallen vergleichbar oder Metalle sogar überlegen sind. Allerdings erfordert die Verarbeitung von Endlosfasern spezialisierte 3D-Drucktechnologien, was zu höheren Kosten und einem komplexeren Herstellungsprozess führt. (3Dnatives, 2024)

2.3.3 Unterschiede der Faserarten

Die mechanischen Eigenschaften von faserverstärkten 3D-Druck-Bauteilen variieren je nach verwendeter Faserart. Die wichtigsten Fasern sind Kohlefaser, Kevlarfaser, Glasfaser und hochtemperaturbeständige Glasfaser (HSHT). Ihre spezifischen Eigenschaften sind in Tabelle 2 zusammengefasst.

Tabelle 2: Mechanische Eigenschaften verschiedener Faserarten (Prototec GmbH, 2025).

Prüfkriterium	Kohlefaser	Kevlarfaser	Glasfaser	HSHT Glasfaser
Zugmodul in GPa	60	27	21	21
Zugfestigkeit in MPa	800	610	590	600
Zug-Bruchdehnung in %	1,5	2,7	3,8	3,9
Biegemodul in GPa	51	26	22	21
Biegefestigkeit in MPa	540	240	200	420
Warmformbeständigkeit HDT in °C	105	105	105	150
Kerbschlagzähigkeit in J/m	960	2000	2600	3100
Dichte in g/cm³	1,4	1,2	1,5	1,5

2.4 Methoden der Faserintegration im 3D-Druck

Die Integration von Verstärkungsfasern in den 3D-Druck ist eine vielversprechende Methode, um die mechanischen Eigenschaften von Bauteilen zu verbessern. Verschiedene Verfahren ermöglichen es, Kurz- oder Endlosfasern gezielt in die Matrix eines thermoplastischen Polymers einzubetten.

Überblick über die Faserverlegungsmethoden

Die Abbildung 17 zeigt eine schematische Darstellung verschiedener Methoden zur Integration von Fasern in den 3D-Druckprozess. Diese lassen sich grob in sechs Kategorien unterteilen:

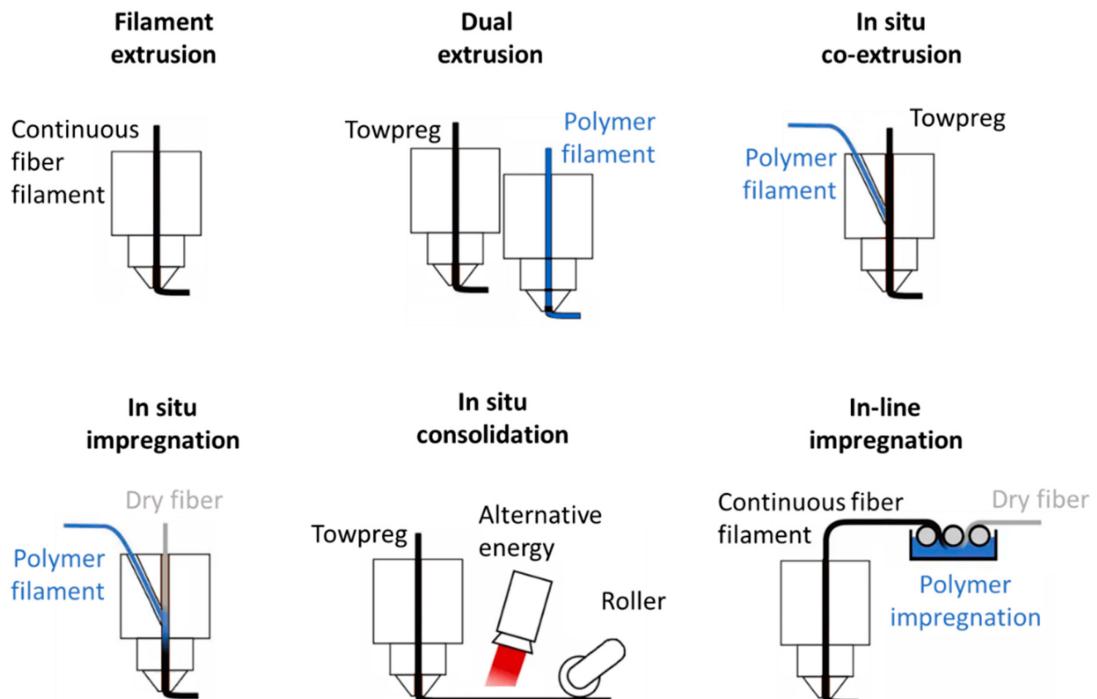


Abb. 17: Verschiedene Verfahren zur Faserintegration im 3D-Druck (nach Mashayekhi et al., 2021).

- **Filament Extrusion:** Ein kontinuierliches Faserfilament wird zusammen mit dem Polymer extrudiert. Dies ist eine einfache Methode, die jedoch eine bereits imprägnierte Faser erfordert.
- **Dual Extrusion:** Hierbei wird eine Polymer- und eine Faserextrusion parallel genutzt. Ein Towpreg (mit Harz imprägnierte Faser) wird gleichzeitig mit einem Polymerfilament extrudiert.
- **In-situ Co-Extrusion:** Ein Polymerfilament wird während des Drucks mit einer Towpreg kombiniert und direkt in der Düse miteinander verbunden.
- **In-situ Imprägnierung:** Eine trockene Faser wird während des Extrusionsprozesses mit flüssigem Polymer imprägniert, um eine direkte Durchmischung zu erzielen.
- **In-situ Konsolidierung:** Eine bereits imprägnierte Faser (Towpreg) wird über alternative Energiequellen wie Laser oder Heizrollen zusätzlich konsolidiert, um eine höhere Festigkeit zu erzielen.
- **In-line Imprägnierung:** Hierbei wird eine trockene Faser durch ein Polymerbad geführt, um sie während des Drucks zu imprägnieren und in den Druckprozess einzubinden.

Jedes dieser Verfahren hat spezifische Vorteile und Herausforderungen. Die Wahl der Methode hängt von den gewünschten mechanischen Eigenschaften, der Prozesskompatibilität und den verfügbaren Materialien ab.

2.5 Marktforschung

Mehrere Unternehmen bieten 3D-Drucksysteme mit faserverstärkten Materialien an, wobei unterschiedliche Methoden zur Integration der Fasern in den Druckprozess genutzt werden.

- **Markforged:** Setzt auf das *Continuous Fiber Reinforcement* (CFR)-Verfahren, bei dem Endlosfasern wie Carbon, Glasfaser oder Kevlar in eine thermoplastische Matrix eingebettet werden. Die Fasern werden präzise entlang der Belastungslinien platziert, wie in Abbildung 18 dargestellt. Dies führt zu hoher mechanischer Festigkeit. Produkte wie der **Mark Two** und der **FX20** ermöglichen diese Technik für Desktop- und industrielle Anwendungen. (Markforged, 2025)

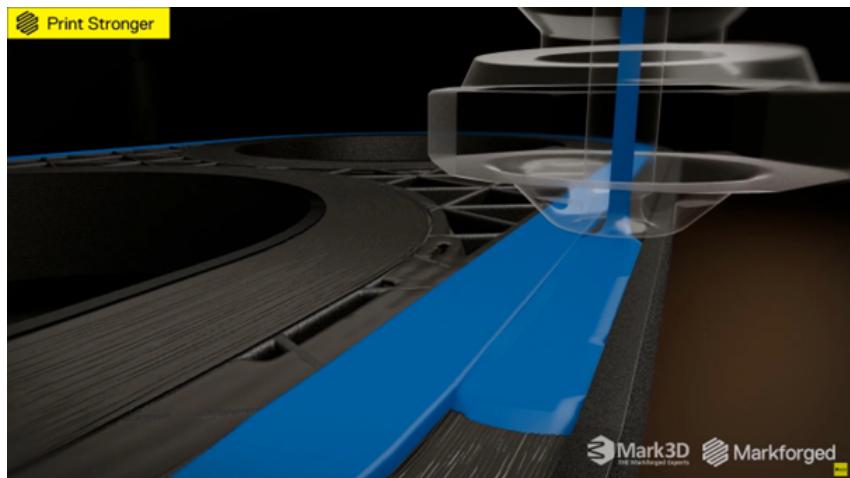


Abb. 18: Darstellung der Fasereinbettung beim kontinuierlichen Faserverstärkungsverfahren CFR (nach Mark3D GmbH, 2022)

- **Anisoprint:** Nutzt die *Composite Fiber Co-extrusion* (CFC)-Methode, bei der trockene Fasern mit einem polymeren Bindemittel kombiniert werden, bevor sie mit der thermoplastischen Matrix zusammengeführt werden. Dadurch kann die Faserorientierung gezielt gesteuert werden. (Anisoprint, 2025)
- **Prototec:** Bietet Dienstleistungen zur Fertigung faserverstärkter 3D-Druck-Bauteile an. Dabei werden sowohl Endlos- als auch Kurzfaser verstärkungen eingesetzt, um anwendungsspezifische mechanische Eigenschaften zu erzielen. (Prototec GmbH, 2025)
- **EB-TEC:** Entwickelt industrielle 3D-Drucksysteme mit Endlosfaserverstärkung und richtet sich insbesondere an Anwendungen im Maschinenbau, der Luftfahrt und der Automobilindustrie. (EB-TEC, 2025)
- **Vinox Systems:** Der **V-Rex** kombiniert eine präzise Steuerung der Faserverlegung mit Hochtemperaturdrucktechnologien und ist auf großformatige, hochbelastbare Bauteile spezialisiert. (Vinox Systems, 2025)

Diese Anbieter setzen unterschiedliche Verfahren zur Faserintegration ein. Während Markforged auf eine zweite Extrusionsdüse setzt, um Fasern direkt in eine thermoplastische Matrix einzubringen (CFR), wird bei Anisoprint eine trockene Faser mit einem Polymermantel umhüllt und erst beim Druck mit dem Hauptmaterial verbunden (CFC). Großformatige Systeme wie jene von Vinox Systems oder EB-TEC ermöglichen zudem den Druck mit Hochleistungspolymeren wie PEEK oder PEKK.

Die Kosten für diese Technologien variieren erheblich. Während Desktop-Systeme wie der **Mark Two** für kleinere Anwendungen konzipiert sind, erfordern großformatige Drucksysteme eine signifikante Investition. Neben den Maschinenkosten fallen zusätzliche Ausgaben für spezielle Hochleistungsfasern und thermoplastische Materialien an.

2.6 Schlussfolgerung

In der vorliegenden Arbeit wurden verschiedene Faserverstärkungstechnologien im 3D-Druck analysiert, wobei insbesondere der Vergleich zwischen Kurz- und Endlosfasern sowie die marktführenden industriellen Lösungen betrachtet wurden. Industrielle Systeme wie die von Markforged, Prototec und Anisoprint setzen fortschrittliche Technologien zur präzisen Faserverlegung ein, um hochfeste und gleichzeitig leichte Bauteile herzustellen. Diese Lösungen bieten herausragende mechanische Eigenschaften, sind jedoch aufgrund der hohen Kosten und speziellen technischen Anforderungen für Privatnutzer schwer zugänglich. Das Ziel dieser Arbeit ist die Entwicklung einer alternativen Lösung zur Herstellung faserverstärkter 3D-Drucke mit einer kosteneffizienten und einfach adaptierbaren Methode. Dabei soll eine neuartige Technik implementiert werden, bei der Filament und Fasern direkt im Hotend miteinander vereint werden, um eine optimale Integration der Verstärkungsfasern in das Bauteil zu erreichen. Dieses Verfahren zielt darauf ab, eine erschwingliche Alternative für Privatkunden darzustellen und mit gängigen 3D-Druckern kompatibel zu sein.

Wahl der Fasern

Für dieses Projekt wurden Kevlar-Fasern ausgewählt. Diese Entscheidung basiert auf mehreren Faktoren:

- **Ausgewogene mechanische Eigenschaften:** Kevlarfasern bieten ein optimales Verhältnis zwischen Festigkeit und Flexibilität. Mit einem Zugmodul von 27 GPa und einer Zugfestigkeit von 610 MPa weisen sie eine höhere Zug-Bruchdehnung von 2,7% auf, was zu einer besseren Schlagzähigkeit führt.
- **Kostengünstige Verfügbarkeit:** Im Vergleich zu anderen Hochleistungsfasern sind Kevlarfasern für Privatkunden am Markt am günstigsten zu erhalten. Die Fasern werden von einem Online-Anbieter (AliExpress, Artikelnr. 1005004530366533) bezogen, was die Skalierbarkeit des Projekts unterstützt.
- **Verarbeitbarkeit:** Die bezogenen Kevlar-Fasern müssen nach Lieferung entgarnt werden, da sie in der Standardform aus drei verdrillten Garnen bestehen. Für die vorgesehene Anwendung werden die einzelnen Garne verwendet, was eine feinere Integration in den Druckprozess ermöglicht.



Brand:	LJY
Name:	Aramid sewing thread
Material:	Aramid fiber

Abb. 19: Die ausgewählten Kevlar-Fasern (nach AliExpress, 2025)

Diese Eigenschaften machen Kevlar-Fasern besonders geeignet für das geplante Vorhaben, da sie sowohl die notwendige Verstärkung bieten als auch praktisch einsetzbar sind. Durch die Kombination dieser Fasern mit einer speziell entwickelten Faser-Filament-Integration im Hotend des 3D-Druckers kann eine effiziente und leistbare Lösung geschaffen werden. Der entwickelte Ansatz ermöglicht es Privatkund:innen, faserverstärkte Bauteile herzustellen, ohne auf teure Industrie anlagen angewiesen zu sein, und trägt somit zur Demokratisierung dieser innovativen Fertigungstechnologie bei.

3 Konstruktion

3.1 Einleitung

Diese Dokumentation beschreibt die Erarbeitung des CAD-Modells für das Projekt "FiberPrinter". Das Hauptziel dieses Projekts ist die Entwicklung eines innovativen 3D-Druckkopfs, der in der Lage ist, Filament mit Fasern innerhalb des Hotends zu vereinen und zu drucken. Dieser Abschnitt gibt eine detaillierte Übersicht über die Konstruktionsziele, die technischen Details sowie die Herausforderungen, die während der Entwicklung des Modells bewältigt wurden.

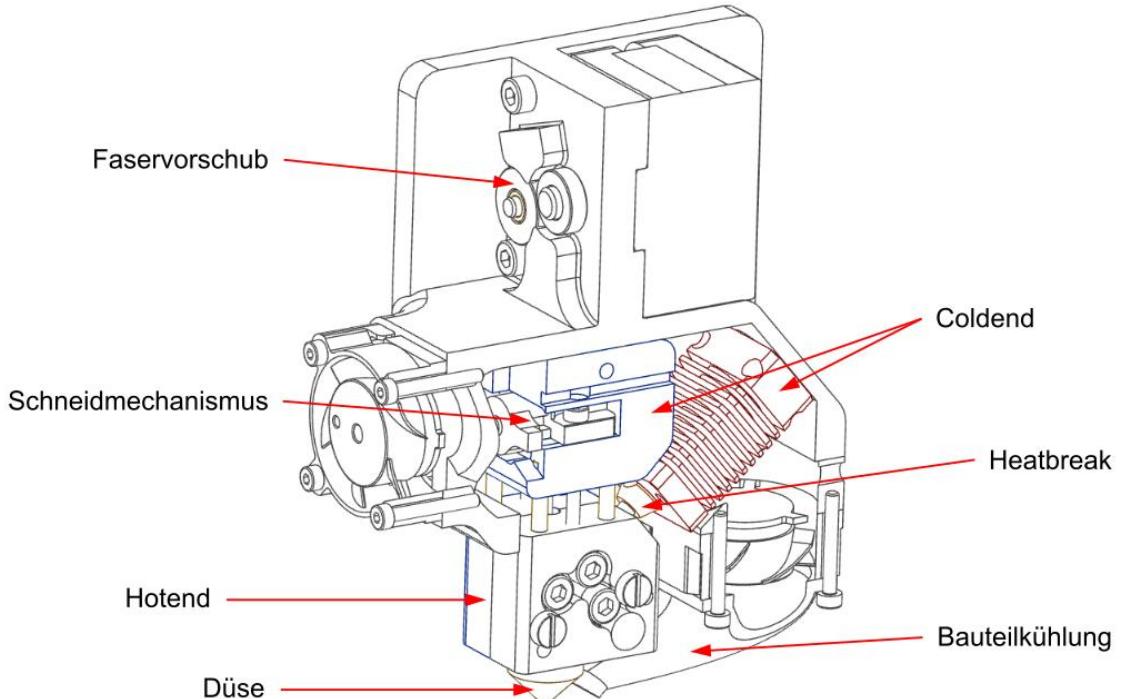


Abb. 20: Ein Überblick des vollständigen CAD-Modells

3.2 Konstruktionsziele

Die Konstruktion des CAD-Modells basiert auf den folgenden Hauptzielen:

- **Effiziente Fusion von Filament und Fasern:** Das Hotend muss in der Lage sein, Filament und Fasern präzise und effizient zu vereinen.
- **Wartungsfreundlichkeit:** Alle Komponenten des Druckkopfs sollen leicht zugänglich und wartbar sein.
- **Plattformunabhängigkeit:** Der Druckkopf soll auf verschiedenen handelsüblichen 3D-Druckern montierbar sein.
- **Präziser Schneidmechanismus:** Ein zuverlässiger Mechanismus zum Schneiden der Fasern muss integriert sein.
- **Zuverlässiger Faservorschub:** Der Vorschub der Fasern soll gleichmäßig und präzise erfolgen.

3.3 Bauteilerklärung

In Abbildung 20 sind die zentralen Bauteile des Modells dargestellt. Im Folgenden werden diese Komponenten näher erläutert:

3.3.1 Hotend

Das Hotend ist das funktionale Zentrum des Druckkopfes. Es ist für das Schmelzen des Filaments und die Fusion mit den Fasern verantwortlich.

3.3.2 Heatbreak und Coldend

Das Heatbreak dient dazu einen kurzen Übergang zwischen dem Hotend und dem Coldend zu gewährleisten sodass das Filament solange wie möglich vor dem eindringen in das Hotend fest bleibt. Das ist notwendig damit das Filament nicht zu früh die Glasübergangstemperatur erreicht, weich zu werden beginnt und in dem Coldend kleben bleibt. Heatcreep beschreibt genau diesen Vorgang dass das Coldend zu warm ist und das Filament im Heatbreak und Coldend weich zu werden beginnt.

- **Filament- und Faserführung:** PTFE-Schläuche leiten das Filament und die Fasern von der Filament- und Faserspule zum Coldend.
- **Wärmeisolierung:** Kühllüfter und Materialien mit geringer Wärmeleitfähigkeit verhindern das zu warm werden des Coldends.

3.3.3 Schneidmechanismus

Der Schneidmechanismus dient dazu die Faser zuverlässig zu schneiden und diese nach dem Schneidvorgang wieder in die Faserführung zum Hotend zu bringen.

3.3.4 Faservorschub

Der Faservorschub dient dazu die Faser nach dem Schneiden wieder bis zum Hotend zu schieben.

- **Vorschubmechanismus:** Zwei Räder aus Gummi, schieben die Fasern nach vorne. Ein Schrittmotor treibt eines der Gummiräder an.

3.3.5 Materialauswahl

Die Materialauswahl für die Komponenten des Druckkopfes ist entscheidend für die Leistungsfähigkeit und Langlebigkeit des Systems.

- **Hotend und Coldend:** Die Materialien für das Hotend und Coldend müssen eine hohe Wärmeleitfähigkeit aufweisen um die Wärme in den Bauteilen gut verteilen zu können. Hierzu wird in der Testphase Aluminium verwendet.
- **Düse:** Die Düse muss so wie das Hotend und Colend auch aus einem Material bestehen das eine hohe Wärmeleitfähigkeit aufweist. Rücksicht zu nehmen ist jedoch auf die abresive Wirkung der Faser welche die Aluminiumdüse sehr schnell verschleissen lasse kann
- **Schneidmechanismus:** Stifte und Buchsen aus gehärtetem Stahl zur Minimierung des Verschleißes.

3.4 Simulationen in Fusion 360 und SimScale

3.4.1 Einleitung

Zur Analyse und Optimierung des 3D-Druckkopfes wurden thermische sowie strömungstechnische Simulationen durchgeführt. Diese wurden mit den Softwaretools Fusion 360 und SimScale realisiert. Während die thermischen Simulationen dazu dienten, die Wärmeverteilung und das thermische Verhalten der Bauteile zu analysieren, wurden die Strömungssimulationen genutzt, um die Luftströmung um den Druckkopf sowie den Fluss des flüssigen Filaments zu visualisieren. Ziel war es, mögliche Schwachstellen frühzeitig zu erkennen und Konstruktionsanpassungen vorzunehmen.

3.4.2 Zielsetzung

Da im 3D-Druckprozess hohe Temperaturen auftreten, ist eine passende Wärmeverteilung essenziell, um eine stabile Druckqualität sicherzustellen. Gleichzeitig beeinflussen Luftströmungen und der Materialfluss die Druckergebnisse erheblich. Um diesen Einfluss zu bewerten, wurden folgende Analysen durchgeführt:

- **Thermische Simulationen:** Untersuchung der Temperaturverteilung, Identifikation von Hotspots und Optimierung von Kühlkonzepten.
- **Strömungssimulationen:** Visualisierung der Luftströmung um den Druckkopf und des Filamentflusses zur Erkennung potenzieller Fehlerquellen.

3.4.3 Rahmenbedingungen

Thermische Simulationen

Für die thermischen Analysen werden folgende Rahmenbedingungen festgelegt:

Lasten

- **Umgebungstemperatur:** Als Umgebungstemperatur werden 20 °C angenommen.
- **Wärmequelle:** Das Heizelement wird mit einer Temperatur von 250 °C belastet.
- **Konvektion1:** Diese Konvektion wird angesetzt um den Luftstrom der Kühllüfter zu simulieren. Die Konvektion1 wird mit $30 \frac{W}{m^2K}$ angesetzt.
- **Konvektion2:** Diese Konvektion wird angesetzt die Bewegung des Druckkopfs zu simulieren. Die Konvektion2 wird mit $6 \frac{W}{m^2K}$ angesetzt.

Die Veranschaulichung der Lasten ist unter Abschnitt 9.1 *Simulationslasten*, S.204 zu finden. Die Konvektionswerte werden der Abbildung 21 entnommen.

Die Wärmeabstrahlung wird in der Simulation vernachlässigt, da die Konvektion bereits eine hohe Variabilität aufweist und die resultierenden Schwankungen der Wärmeübergangskoeffizienten den Einfluss der Abstrahlung übersteigen. Da die Abstrahlung innerhalb dieses Varianzbereichs liegt, ist ihr Einfluss auf das Gesamtergebnis vernachlässigbar.

Luft an der Wand		Wärmeübergangskoeffizient α (W/(m ² *K))
Luft senkrecht zur Metallwand [6]	ruhend	3,5...35
Luft senkrecht zur Metallwand [6]	mäßig bewegt	23...70
Luft senkrecht zur Metallwand [6]	kräftig bewegt	58...290
Luft längs zu ebener Wand [6]	polierte Oberfläche $v < 5 \text{ m/s}$	$5,6 + 4 * v$
Luft längs zu ebener Wand [6]	polierte Oberfläche $v > 5 \text{ m/s}$	$7,12 * v^{0,78}$

Abb. 21: Wärmeübergangskoeffizienttabelle (Schweizer-fn-Luft, 2025)

Materialien

Für die Simulation werden den Bauteilen folgende Materialien zugewiesen:

- **Hotend:** Kupfer
- **Coldend:** Aluminium
- **Heizelement:** Aluminium
- **Heatbreak:** Titan und Kupfer
- **Isolierung:** Wird je nach Design zugewiesen

Bei einer Versionsänderung eines Bauteils wird abhängig von den modifizierten Eigenschaften ein spezifisches Material für die Simulation zugewiesen, um eine realitätsnahe Abbildung der physikalischen Parameter sicherzustellen.

Strömungssimulationen

Die Strömungssimulationen dienten ausschließlich dem relativen Vergleich und nicht der exakten physikalischen Berechnung, da keine genauen Materialkennwerte für den flüssigen Kunststoff vorliegen. Ziel ist es, potenzielle Probleme in der Strömungsführung zu identifizieren:

- **Luftstromanalyse:** Untersuchung der Luftströmung um das Coldend, um die Effizienz der aktiven Kühlung zu bewerten und unerwünschte Verwirbelungen zu erkennen.
- **Filamentfluss:** Simulation des Flusses des geschmolzenen Kunststoffs durch die Düse, um Engstellen, ungleichmäßige Strömungen oder unerwartete Ablenkungen zu visualisieren.

Materialien

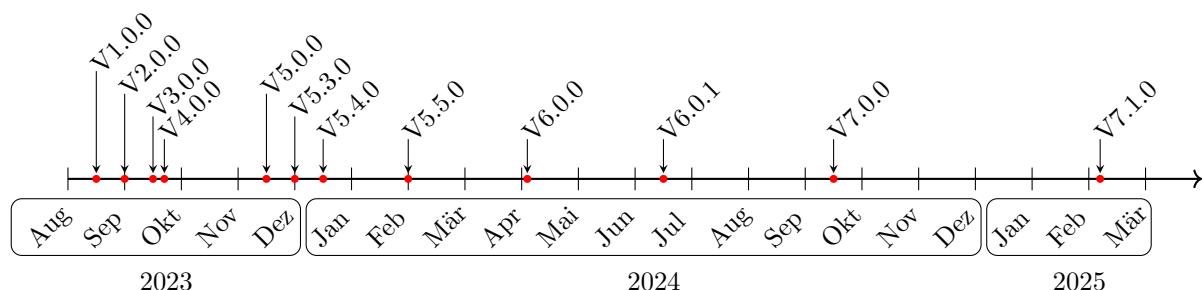
Da keine präzisen Kennwerte für die Viskosität und Fließfähigkeit des geschmolzenen Kunststoffs verwendet wurden, dienen diese Simulationen ausschließlich zur qualitativen Analyse und zur Erkennung offensichtlicher Konstruktionsfehler. Für die Simulation wurden Materialmodelle eingesetzt, deren thermophysikalische Eigenschaften den realen Materialien möglichst nahekommen.

3.4.4 Nutzen von Simulationen

Simulationen ermöglichen eine effiziente und kostengünstige Analyse komplexer physikalischer Prozesse, bevor experimentelle Tests erfolgen. Sie bieten die Möglichkeit, verschiedene Designvarianten unter kontrollierten Bedingungen zu vergleichen und kritische Bereiche frühzeitig zu identifizieren. Dadurch verkürzen sie die Entwicklungszeit, reduzieren Materialkosten und ermöglichen gezielte Optimierungen. Während Experimente oft zeitaufwendig und ressourcenintensiv sind, erlauben Simulationen eine schnelle Anpassung von Parametern und eine detaillierte Visualisierung physikalischer Phänomene, die in realen Tests schwer messbar sind.

3.5 Versionsverlauf

Hier wird der vollständige Versionsverlauf der wichtigsten Versionen übersichtlich dargestellt.



3.6 Version 1

3.6.1 Konstruktion der ersten Version

Druckkopf

Der entwickelte Druckkopf orientiert sich an bewährten Konzepten etablierter Hersteller, insbesondere an den Lösungen von Slice Engineering. Die grundlegende Bauweise wird weitgehend beibehalten, jedoch um die Zuführung einer Faser als zusätzliche Funktion erweitert. Dies erfordert eine Anpassung der internen Geometrie sowie der Materialzuführung, um eine zuverlässige Integration der Faser in den Druckprozess zu gewährleisten.

In der aktuellen Entwicklungsstufe liegt der Fokus primär auf der Funktionsweise und der mechanischen Integration der Faserzuführung. Fertigungstechnische Aspekte, insbesondere in Bezug auf konventionelle Herstellungsverfahren, treten zunächst in den Hintergrund. Die Produktion des Druckkopfs erfolgt mittels Selektivem Laserschmelzen (SLM), wodurch konstruktive Einschränkungen durch klassische Fertigungsverfahren entfallen. Dies ermöglicht eine freie Gestaltung komplexer Geometrien, die gezielt auf die Anforderungen der Faserführung, die thermische Stabilität sowie die mechanische Belastbarkeit des Druckkopfs optimiert werden können.

Durch die additive Fertigung lassen sich zudem Funktionselemente, wie optimierte Kühlstrukturen oder integrierte Kanäle zur Materialzufuhr, direkt in die Bauteilgeometrie integrieren. Dies trägt zur Kompaktheit und Effizienz des Gesamtsystems bei, ohne die strukturelle Integrität oder die thermische Performance des Druckkopfs zu beeinträchtigen.

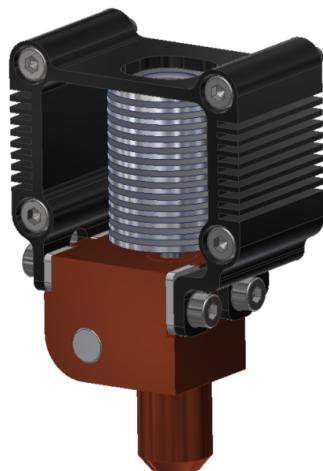


Abb. 22: Version 1.1.0



Abb. 23: Mosquito Hotend by SliceEngineering
(Sliceengineering, 2025)

Isolationspads

Zur thermischen Entkopplung zwischen dem Befestigungsrahmen und dem Hotend wurden speziell entwickelte Isolationspads aus Keramik konstruiert. Diese gewährleisten eine effiziente Isolation des Befestigungsrahmens, um eine ungewollte Wärmeübertragung auf die Druckerachsen zu verhindern. Eine Erhöhung der Temperatur in diesen Bereichen könnte die mechanische Präzision des Systems negativ beeinflussen und zu thermisch bedingten Ausdehnungen führen, welche die Druckqualität beeinträchtigen könnten.

Die Wahl von Keramik als Material für die Isolationspads basiert auf deren hervorragenden thermischen Isolationseigenschaften sowie der hohen Temperaturbeständigkeit.

3.6.2 Thermische Analyse

Bauteilspezifische Materialien

- **Isolationspads:** Da Keramik in Fusion 360 standardmäßig nicht als Simulationsmaterial zur Verfügung steht wird den Isolationspads Glas als Material zugewiesen.
- **Schrauben:** Den Schrauben wurde Stahl als Simulationsmaterial zugewiesen

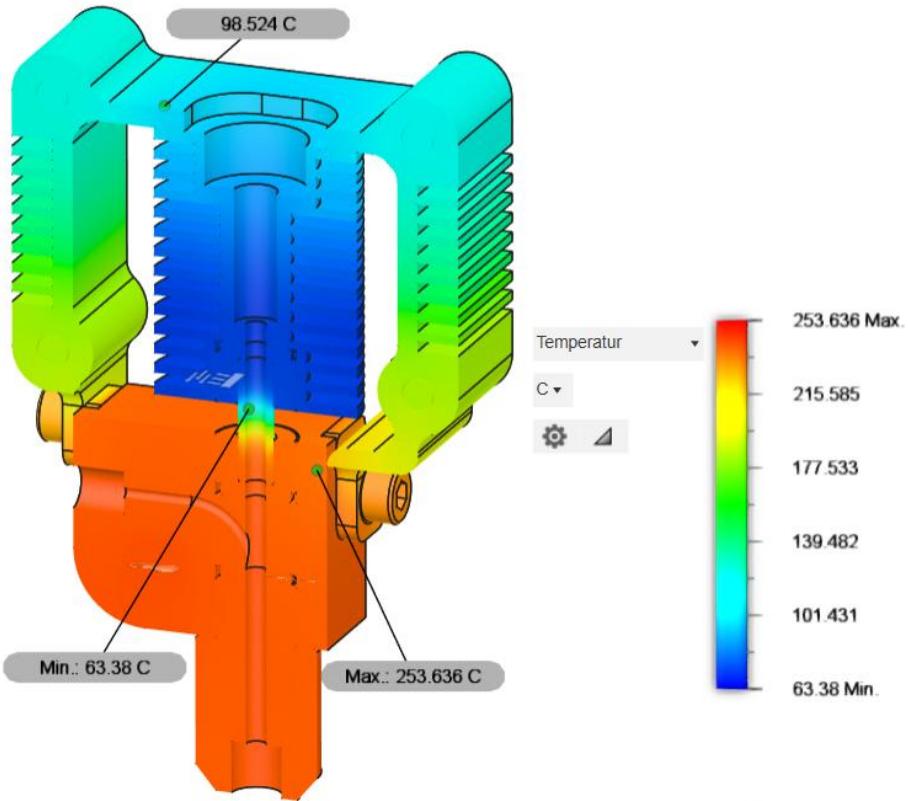


Abb. 24: Thermische Simulation

Thermische Analyse des Druckkopfs

Die durchgeführte thermische Simulation zeigt eine effektive Wärmeableitung im Bereich des Kühlkörpers, jedoch erreicht der Befestigungsrahmen Temperaturen von bis zu 98 °C und das Coldend Temperaturen von weit über 60 °C.

Aufgrund der niedrigen Erweichungstemperatur von PLA bei etwa 60 °C besteht die Gefahr, dass das Filament bereits im Coldend zu weich wird und dadurch den Materialfluss blockiert. Diese frühzeitige Verformung führt häufig zu Verstopfungen, da das teilverflüssigte PLA den vorgesehenen Transportweg nicht mehr ungehindert passieren kann. Hohe Temperaturen am Befestigungsrahmen können Probleme für die Kinematik darstellen und diese beeinträchtigen.

Um eine sichere Verarbeitung von PLA zu gewährleisten, ist eine optimierte thermische Isolation zwischen dem Hotend und der Befestigungsstruktur erforderlich. Dies kann durch den Einsatz von hochisolierenden Materialien oder zusätzlichen Wärmebarrieren realisiert werden, um die Wärmeübertragung weiter zu minimieren und eine thermische Entkopplung sicherzustellen.

Analyse der maximalen Temperaturüberschreitung

In der thermischen Simulation wurde eine Temperatur von 250 °C für das Hotend angesetzt, jedoch zeigt die Analyse eine maximale Temperatur von 253,636 °C. Diese Überschreitung kann auf verschiedene Faktoren zurückgeführt werden:

- **Numerische Diskretisierung:** Durch die Gitterdiskretisierung und Interpolation innerhalb der Simulationssoftware kann es zu geringfügigen Abweichungen kommen.
- **Materialeigenschaften:** Die verwendeten Wärmeleitfähigkeiten und spezifischen Wärmekapazitäten der Materialien beeinflussen die Wärmeverteilung und können in der Simulation zu leichten Temperatursteigerungen führen.
- **Numerische Konvergenz:** Die iterative Lösung der Wärmeleitungsgleichungen kann in Abhängigkeit von der gewählten Toleranz geringe numerische Fehler verursachen.

Die Abweichung von 3,6 °C liegt im tolerierbaren Bereich und beeinflusst die Praxis voraussichtlich nicht erheblich, kann jedoch durch präzisere Simulationsparameter weiter minimiert werden.

3.6.3 Unterversionen

Der folgende Zeitstrahl zeigt die Entwicklungsschritte von Version 1 des Druckkopfes.

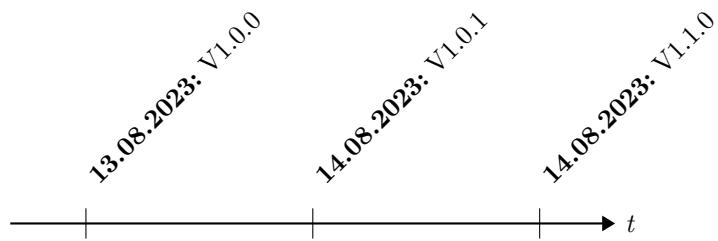


Tabelle 3: Arbeitszeitübersicht der Entwicklungsphasen

Version	Beschreibung	Arbeitszeit
V1.0.0	Hotend überarbeitet, Coldend konstruiert, thermische Analyse, Befestigungsrahmen konstruiert, Materialauswahl für Isolationspads ausstehend.	05:46:54
V1.0.1	Thermische Analyse durchgeführt.	00:14:50
V1.1.0	Genauere thermische Analyse	00:39:21
Gesamt		06:41:05

3.6.4 Fazit

Die erste Version des Druckkopfes bildet eine stabile Grundlage für die weitere Entwicklung. Erste Konstruktionsanpassungen sowie thermische Analysen ermöglichen gezielte Optimierungen hinsichtlich der Wärmeverteilung und mechanischen Stabilität.

Die Ergebnisse der thermischen Analyse zeigen jedoch, dass das Coldend weiterhin eine zu hohe Temperatur aufweist. Dies deutet auf eine unzureichende Wärmeableitung oder eine unzureichende thermische Entkopplung hin. Weitere Anpassungen an der Kühlstruktur, den Materialien oder der Geometrie sind erforderlich, um eine effektivere Temperaturniedrigung zwischen dem Hotend und dem Coldend zu erreichen.

3.7 Version 2

3.7.1 Konstruktion der zweiten Version

Druckkopf

Die zweite Version des Druckkopfes baut auf den Erkenntnissen der vorherigen Entwicklungsstufe auf und integriert gezielte Verbesserungen hinsichtlich der thermischen Isolation. Die grundlegende Bauweise wurde beibehalten, jedoch wurden Anpassungen vorgenommen, um die Wärmeableitung effizienter zu gestalten und die Temperaturübertragung auf andere Bauteile weiter zu reduzieren.

Ein Schwerpunkt dieser Version liegt auf der Optimierung der Kühlkörper im Bereich des Befestigungsrahmens, um die Wärmeübertragung gezielt zu steuern.

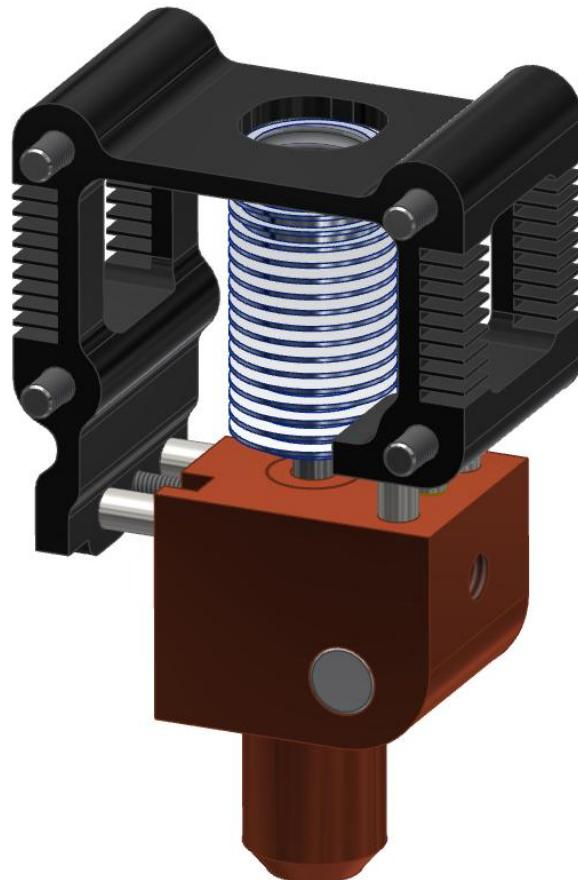


Abb. 25: Version 2.1.1

Isolationsstifte

Zur weiteren Verbesserung der thermischen Isolation werden die Isolationspads durch Titanstifte ersetzt, da Keramik zu bearbeiten im Rahmen des Projekts nicht möglich ist.

3.7.2 Thermische Analyse

Bauteilspezifische Materialien

- **Isolationsstifte:** Den Stiften wurde Titan als Simulationsmaterial zugewiesen.
- **Schrauben:** Den Schrauben wurde Stahl als Simulationsmaterial zugewiesen.

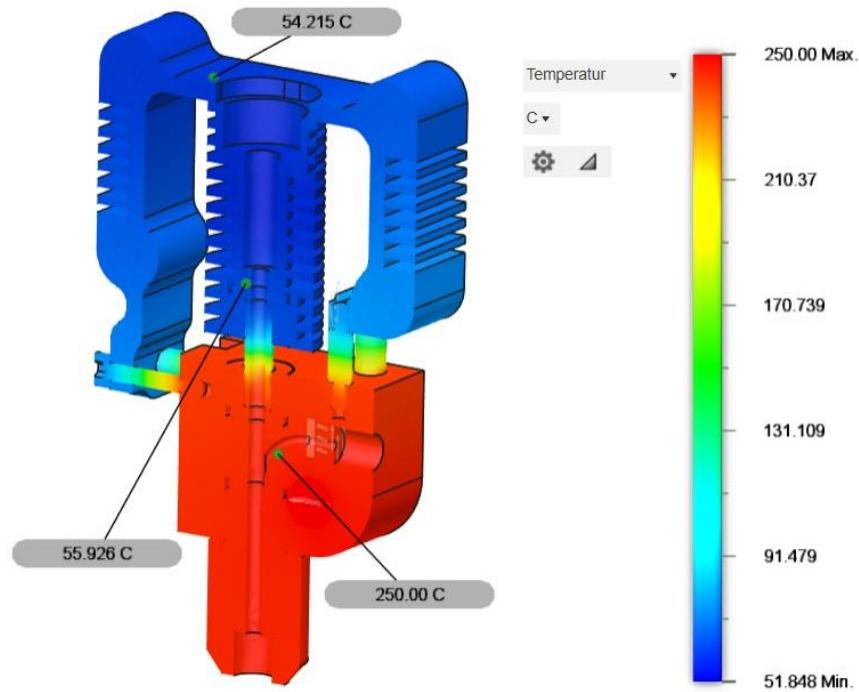


Abb. 26: Thermische Simulation V2.0.0

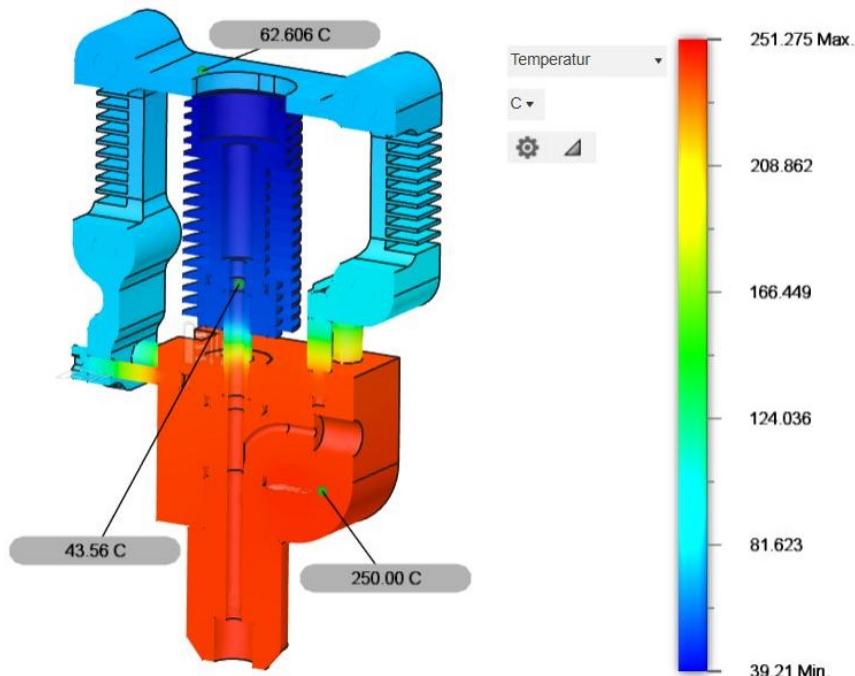


Abb. 27: Thermische Simulation V2.1.1

Thermische Analyse des Druckkopfs

Die aktualisierte thermische Simulation zeigt eine deutliche Verbesserung der Wärmeableitung im Bereich des Coldends. Während in der vorherigen Version Temperaturen von bis zu 98 °C erreicht wurden, konnte dieser Wert nun auf 54,215 °C gesenkt werden. Dies bestätigt eine effizientere Kühlleistung durch die Optimierung der Kühllamellen sowie die verbesserte thermische Isolation zwischen Hotend und Befestigungsstruktur. Die niedrigste erfasste Temperatur der Version 2.0.0 liegt bei 51,848 °C, was auf eine gleichmäßige Wärmeverteilung im Kühlbereich hindeutet. Trotz der erzielten Verbesserungen besteht weiteres Potenzial zur Optimierung der Wärmeableitung. Die Simulation zeigt, dass die niedrigste erfasste Temperatur in Version 2.1.1 bei 39,21 °C liegt und im Coldend auftritt. Im Befestigungsrahmen wurde eine höhere Temperatur als in Version 2.0.0 festgestellt, was vermutlich auf die Reduzierung der Kühllamellen zurückzuführen ist. Durch den Spalt zwischen Coldend und Befestigungsrahmen ist das Coldend jedoch besser thermisch abgekoppelt, wodurch eine geringere Temperatur erreicht wird.

Analyse der Wärmeabfußoptimierung

Die Ergebnisse zeigen eine verbesserte thermische Entkopplung zwischen Hotend und Coldend, jedoch besteht weiterhin eine geringfügige Wärmeübertragung an den Schraubverbindungen. Dies könnte durch eine gezielte Anpassung der Materialwahl oder zusätzliche Isolationsmaßnahmen weiter minimiert werden.

Optimierungspotenzial:

- Erweiterung der Kühllamellen:** Eine größere Oberfläche zur Wärmeabfuhr könnte die Temperatur weiter reduzieren.
- Materialoptimierung:** Die Verwendung eines Materials mit höherer Wärmeleitfähigkeit für den Kühlkörper könnte die Wärmeabfuhr verbessern.
- Zusätzliche Isolationsmaßnahmen:** Eine bessere thermische Entkopplung der Schraubverbindungen könnte eine weitere Temperaturabsenkung ermöglichen.

3.7.3 Unterversionen

Der folgende Zeitstrahl zeigt die Entwicklungsschritte von Version 2 des Druckkopfes.

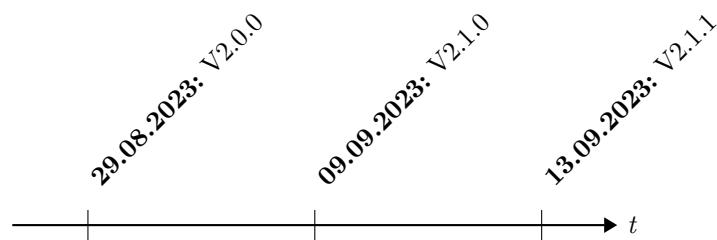


Tabelle 4: Arbeitszeitübersicht der Entwicklungsphasen von Version 2

Version	Beschreibung	Arbeitszeit
V2.0.0	Verbesserte thermische Isolation	02:10:46
V2.1.0	Optimierung der Kühllamellen, thermische Analyse, Recherche Mosquito Hotend	01:53:59
V2.1.1	Materialreduktion beim Hotendmount zur Minimierung der Temperaturausweitung, Belastungsanalyse	04:46:13
Gesamt		08:51:58

3.7.4 Fazit

Die zweite Version des Druckkopfes zeigt signifikante Fortschritte hinsichtlich der thermischen Isolation und Kühlleistung. Die bisherigen Optimierungen haben die Temperatur des Coldends erheblich reduziert, dennoch sind weitere Anpassungen an der Kühlstruktur und den Verbindungselementen erforderlich, um die Wärmeableitung weiter zu verbessern.

3.8 Version 3

3.8.1 Konstruktion der dritten Version

Druckkopf

Die dritte Version des Druckkopfes markiert einen grundlegenden Fortschritt in der Kühltechnologie, indem das Coldend erstmals mit Wasser anstelle von Luft gekühlt wird. Das Coldend besteht aus Aluminium 6061 und wurde so konstruiert, dass der Kühlkanal spiralförmig um den Filamentkanal verläuft. Diese Geometrie ermöglicht eine gleichmäßige Wärmeabfuhr und eine effizientere Temperaturkontrolle im Vergleich zu den bisherigen Versionen. Zusätzlich werden die Montagelöcher direkt in das Coldend integriert. Die Verbindung zwischen Cold- und Hotend wird mithilfe von Titanstiften und Schrauben verstärkt, um ein Brechen des Heatbreaks bei einem Crash zu verhindern und den gesamten Druckkopf steifer zu gestalten. Aufgrund des komplexen internen Kühlkanals ist dieses Bauteil ausschließlich mit dem Selektiven Laserschmelzen (SLM) herstellbar, da konventionelle Fertigungsmethoden diese Geometrie nicht realisieren können. Es bereitet jedoch keine signifikanten Mehrkosten, da auch das Hotend mit dem SLM Prozess hergestellt wird.

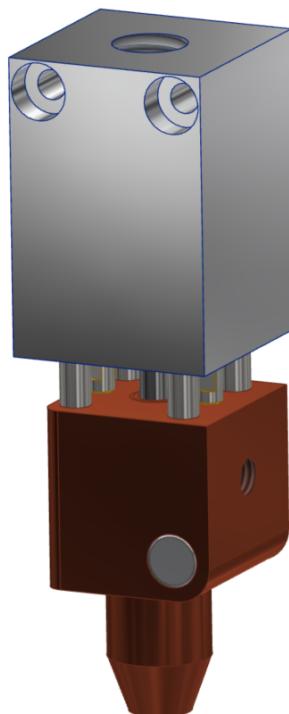


Abb. 28: Version 3.1.1

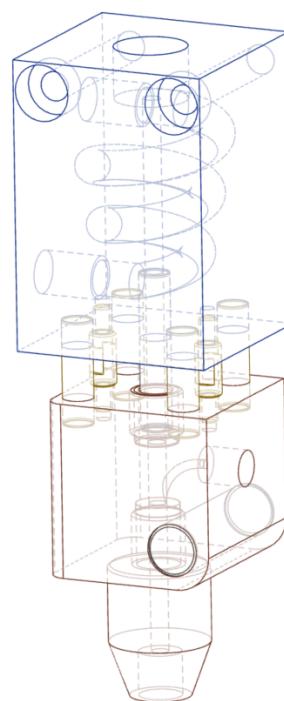


Abb. 29: Version 3.1.1, interne Struktur

3.8.2 Thermische Analyse

Bauteilspezifische Materialien

- Schrauben und Isolationsstifte** Den Bauteilen wird für die Simulation Titan zugewiesen.

Für die Simulation wird eine neue Konvektionslast angesetzt um den Druckkopf zu simulieren. Der Wärmetübergangskoeffizient wird der Abbildung 30 entnommen, dabei wird eine Strömungsgeschwindigkeit von $1 \frac{\text{m}}{\text{s}}$ angenommen.

Wasser	$v = 1 \text{ m/s}$	4000	
Wasser	$v = 5 \text{ m/s}$	16000	

Abb. 30: Wärmeübergangskoeffizient Wasser (Schweizer-fn-Wasser, 2025)

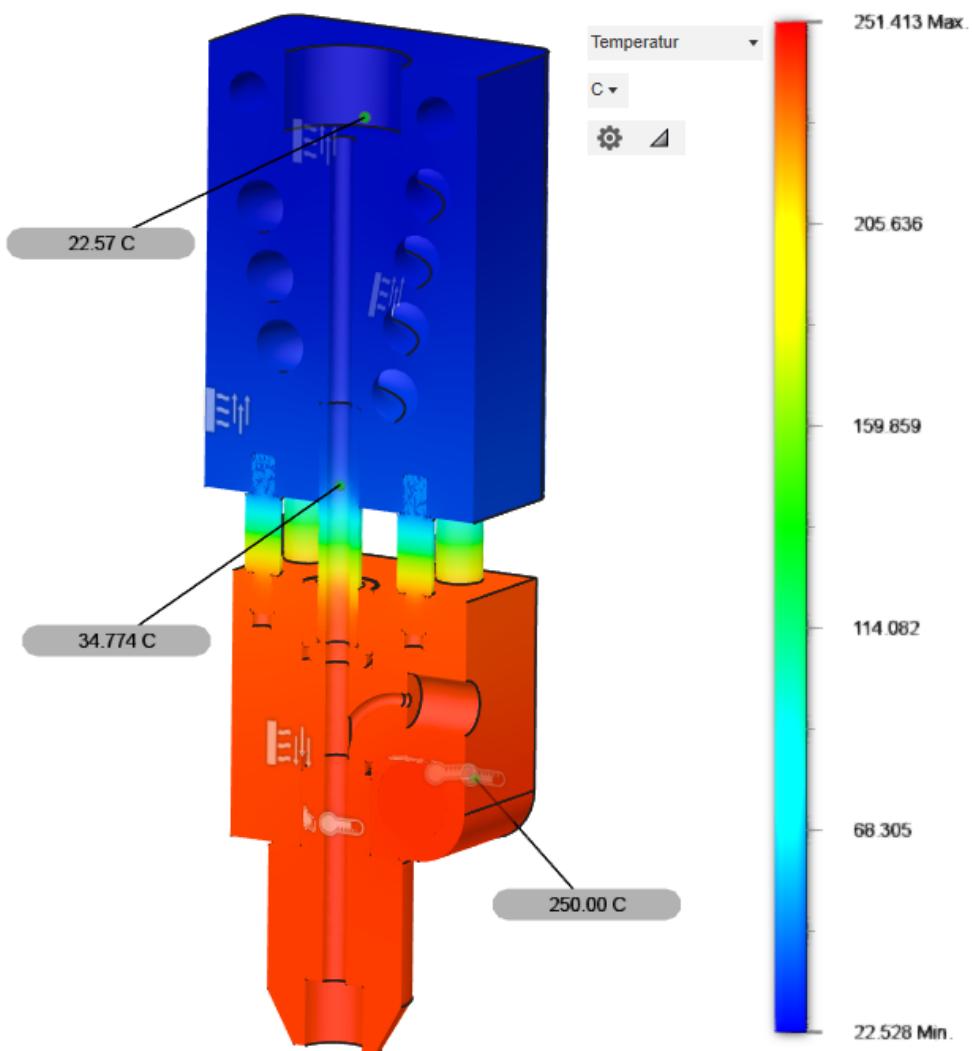


Abb. 31: Thermische Simulation V3.1.1

Thermische Analyse des Druckkopfs

Die durchgeführte thermische Simulation bestätigt die signifikanten Verbesserungen der Wärmeableitung durch die Einführung der Wasserkühlung. Die wesentlichen Ergebnisse sind:

- Das **Hotend** erreicht eine maximale Temperatur von 250,00 °C, entsprechend der Simulationsrandbedingungen.
- Der Temperaturbereich des Coldends zeigt eine erhebliche Reduktion:
 - Übergangsbereich zum Hotend: 34,774 °C
 - Oberer Bereich des Coldends: 22,57 °C
 - Niedrigste erfassste Temperatur: 22,528 °C
- Die Temperaturverteilung ist homogen, was auf eine effiziente Wärmeabfuhr durch die spiralförmig angeordneten Kühlkanäle hindeutet.

Im Vergleich zu den vorherigen Versionen, in denen der Befestigungsrahmen und das Coldend noch Temperaturen von über 53 °C erreichten, zeigt sich eine signifikante Verbesserung. Die Implementierung der Wasserkühlung minimiert die Wärmeübertragung auf das Coldend erheblich und gewährleistet eine effiziente thermische Entkopplung zwischen Coldend und Hotend.

Zudem führt die direkte Integration der Montagelöcher in das Coldend zu einer weiteren Reduktion der Wärmeleitung, wodurch eine ungewollte Temperaturübertragung auf mechanische Komponenten effektiv verhindert wird.

3.8.3 Unterversionen

Der folgende Zeitstrahl zeigt die Entwicklungsschritte von Version 3 des Druckkopfes.

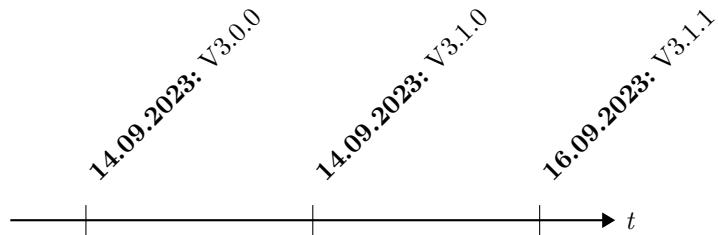


Tabelle 5: Arbeitszeitübersicht der Entwicklungsphasen von Version 3

Version	Beschreibung	Arbeitszeit
V3.0.0	Einführung der Wasserkühlung, Coldend aus Aluminium 6061 mit spiralförmigem Kühlkanal.	02:36:13
V3.1.0	Thermische Analyse bestätigt die Effizienz der Wasserkühlung. FEM-Simulation zur Optimierung der Befestigungsschrauben geplant. Integration eines Temperatursensors erforderlich.	00:39:58
V3.1.1	Die Integration eines Motors für die Faserzuführung wurde aufgrund von Platzmangel und einer ineffizienten Lösungsstrategie verworfen, ohne dass eine Konstruktion dazu erstellt wurde.	02:49:15
Gesamt		06:05:26

3.8.4 Fazit

Die dritte Version des Druckkopfes stellt einen erheblichen Fortschritt in der thermischen Optimierung dar. Durch die Implementierung der Wasserkühlung konnte die Wärmeableitung signifikant verbessert und das Coldend auf unter 35 °C gekühlt werden. Dies gewährleistet eine effektive thermische Entkopplung zwischen Hotend und Coldend, wodurch weitere Anpassungen an der Kühlstruktur nicht erforderlich sind.

3.9 Version 4

3.9.1 Konstruktion der vierten Version

Die vierte Version des Druckkopfes zeichnet sich durch eine grundlegende Umkonstruktion aus. Anstatt eines additiv gefertigten Bauteils kommt nun ein Splithotend-Design zum Einsatz, das sich durch zwei exakt ausgerichtete Hälften (mittels Passstiften) auszeichnet. Diese Bauform ermöglicht eine subtraktive Fertigung (z.B. CNC-Fräsen), was dank glatter Oberflächen und besserer Zugänglichkeit zu einer vereinfachten Montage und Wartung führt.

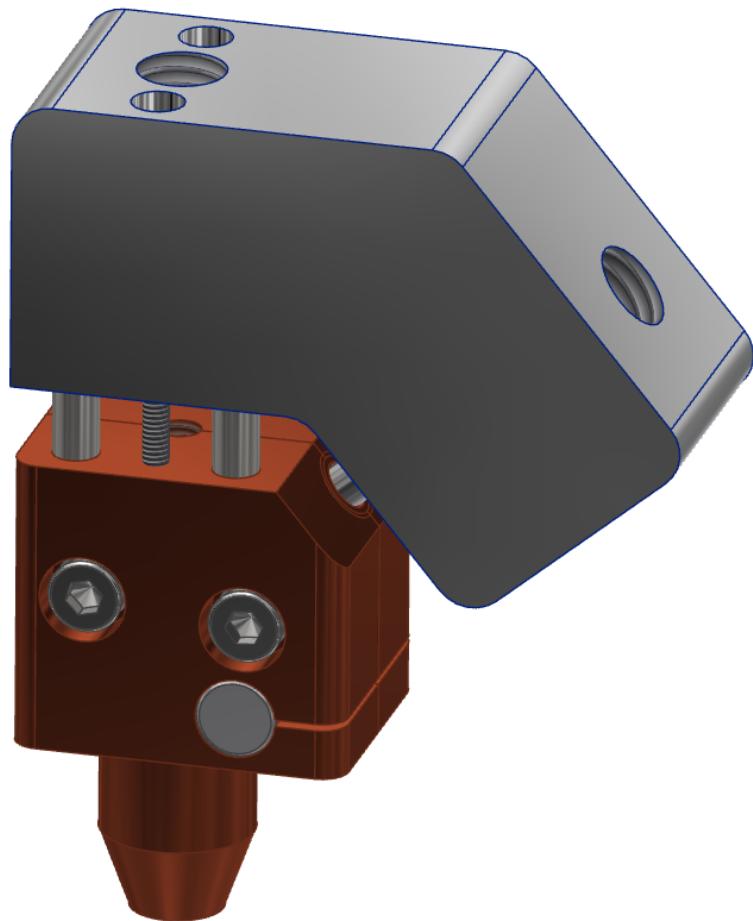


Abb. 32: Version 4.1.1

Wichtige Merkmale:

- **Wasserkühlung beibehalten:** Wie in Version 3 bewährt, kommt weiterhin eine Spiralkühlung zum Einsatz, weshalb auf eine erneute thermische Simulation verzichtet wird.
- **Geradlinige Faserzuführung:** Die Faser wird von oben eingebracht, um einen möglichst störungsfreien Faserfluss zu gewährleisten. Ein kleines, röhrenförmiges Insert aus gehärtetem Stahl führt die Faser bis in den Schmelzbereich und zentriert sie.
- **Splithotend-Aufbau:** Zwei Hälften werden durch Passstifte exakt ausgerichtet, damit Filament- und Faserkanäle präzise aufeinandertreffen.

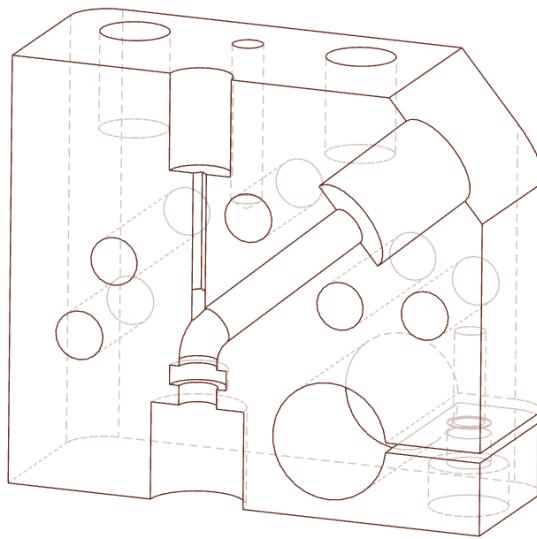


Abb. 33: Splithotend

- **Fertigungsoptimierungen:**
 - **Standardisierte Schrauben:** Anstelle eigens gefertigter Schrauben werden nun gängige M2x20 Zylinderschrauben (Innensechskant) von oben verbaut.
 - **Rautenform für die Faserbohrung:** Anstelle einer reinen Bohrung wird eine rautenförmige Kontur eingefräst, um sie mit einem 90°-Fräser effizient fertigen zu können.
 - **Insert-Fertigung:** Das sehr kleine Insert aus gehärtetem Stahl stellt weiterhin eine Herausforderung dar, da es höchste Präzision in der Herstellung erfordert.
- **Faserschnitt-Konzept:** Ein Mechanismus zum automatischen Kürzen der Faser (basiert auf dem Prinzip einer Magnetspule) befindet sich in einer ersten Konzeptphase; eine finale Konstruktion ist jedoch noch nicht abgeschlossen.
- **Thermische Isolierung:** Bewährte Titanstifte minimieren weiterhin die Wärmeübertragung zwischen Hotend und den restlichen Bauteilen.

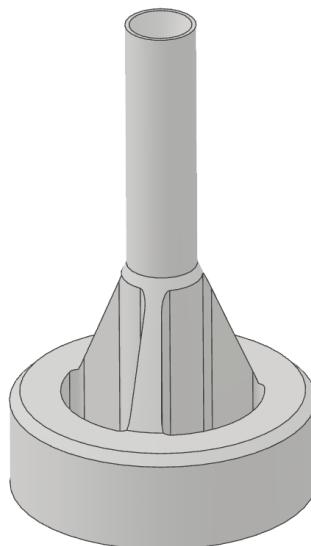


Abb. 34: Insert

Vorteile der Neukonzeption:

- **Verbesserte Fertigbarkeit Hotend:** Subtraktive Prozesse lassen sich deutlich einfacher umsetzen als SLM-Druck mit anspruchsvollen Oberflächenanforderungen.
- **Geringere Montagekomplexität:** Durch das Splithotend-Design können alle Komponenten besser inspiziert und ausgerichtet werden.
- **Konstante Kühlung:** Die erprobte Wasserkühlung garantiert weiterhin eine zuverlässige Temperaturregulierung ohne zusätzliche Modifikationen.
- **Zukunftssicherheit:** Die klare Trennung von Schmelz- und Faserzuführbereich vereinfacht spätere Anpassungen, wie etwa die Integration der Faserschnitt-Einheit.

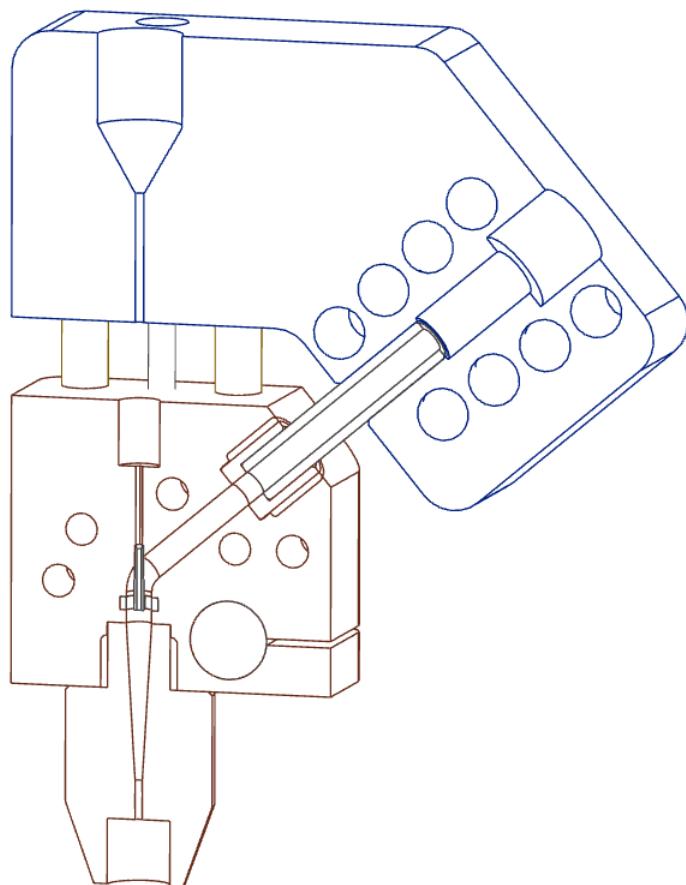


Abb. 35: Halbschnitt, Version 4.1.1

3.9.2 Unterversionen

Der folgende Zeitstrahl zeigt die Entwicklungsschritte von Version 4 des Druckkopfes.

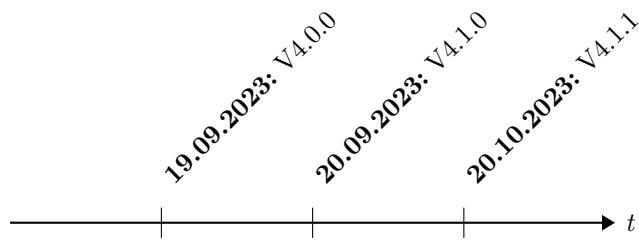


Tabelle 6: Arbeitszeitübersicht der Entwicklungsphasen von Version 4

Version	Kurzbeschreibung	Arbeitszeit
V4.0.0	Erste SplitHotend-Skizzen; Faser wird von oben zugeführt	02:12:02
V4.1.0	Komplett überarbeitetes Hotend; Insert aus Stahl zur Faserzentrierung	04:10:02
V4.1.1	Fertigungs- und Montageoptimierungen; Standard-Schrauben, Raute-Fräzung	01:07:24
Gesamt		07:29:28

3.9.3 Fazit

Das neue SplitHotend-Konzept ermöglicht eine deutlich vereinfachte CNC-Fertigung und Montage des Hotends. Die geradlinige Faserzuführung verbessert das Einführen der Verstärkungsfasern, während ein Insert aus gehärtetem Stahl für die präzise Zentrierung sorgt. Dank der bewährten Wasserkühlung wird auf eine erneute thermische Simulation verzichtet. Künftige Entwicklungen konzentrieren sich auf die Umsetzung des Faserschnitt-Mechanismus und die Feinabstimmung des Inserts, um einen stabilen und zuverlässigen Druckprozess für faserverstärkte Materialien sicherzustellen.

Trotz der erzielten Verbesserungen wird diese Version nicht weiterverfolgt, da die Wasserkühlung aufgrund ihres erhöhten konstruktiven und betrieblichen Aufwands nicht den Anforderungen an eine modulare Integration in verschiedene Druckersysteme entspricht. Zudem konnte die Integration eines funktionalen Faservorschubs in dieser Version nicht realisiert werden.

3.10 Version 5

Die fünfte Version des Druckkopfes stellt eine umfangreiche Weiterentwicklung dar. Neben der Implementierung eines luftgekühlten Coldends und eines überarbeiteten Gehäuses liegt der Fokus insbesondere auf der Konstruktion und Analyse des Schneidmechanismus (SMN). Zusätzlich werden thermische Simulationen und Strömungssimulationen durchgeführt, um die Effizienz und Funktionalität der einzelnen Komponenten zu evaluieren und gezielt zu verbessern.

3.10.1 Konstruktion der fünften Version

Druckkopf

Die grundsätzliche Konstruktion des Druckkopfes wird überarbeitet, um eine effizientere Fertigung mittels CNC-Fräsen zu ermöglichen. In früheren Versionen wird ein selektives Laserschmelzen (SLM) in Betracht gezogen, jedoch aufgrund der zu rauen Oberflächen verworfen. Passstifte zwischen den beiden Hälften des Hotends gewährleisten eine präzise Ausrichtung der Kanäle für das Filament und die Faser. Zusätzlich werden die Titanstifte in Version 5.5 hohl ausgeführt, um die Wärmeleitung zu reduzieren.



Abb. 36: Gesamtansicht des Druckkopfes, Version 5.5.0

Coldend

Das Coldend wird mit einer Luftkühlung ausgestattet, wobei 1-mm-dicke Lamellen für eine verbesserte Wärmeableitung sorgen. Zwei Kühllufter mit einer Durchflussmenge zwischen 3 und 5 m³/h sind direkt am Toolhead montiert. Eine Strömungssimulation zeigt Verbesserungspotential bei der Luftführung.

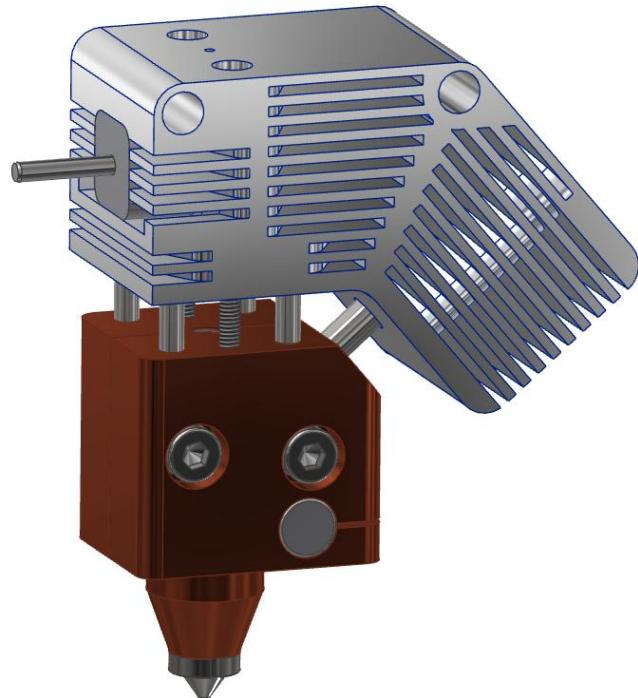


Abb. 37: Druckkopf, Version 5.5.0

Schneidmechanismus

Der Schneidmechanismus (SMN) wird neu entwickelt und besteht aus folgenden Hauptkomponenten:

- **Bohrbuchse:** Dient als feste Führung und Schnittkante, gefertigt aus gehärtetem Stahl mittels EDM-Verfahren.
- **Passstift:** Schneidet die Faser linear ab, ebenfalls gehärtet und hochpräzise gefertigt.
- **Stahlgehäuse:** Schützt die internen Mechanismen und integriert sie in das Gesamtsystem.
- **Rückstellfeder:** Bringt den Passstift nach jedem Schneidvorgang in die Ausgangsposition zurück.

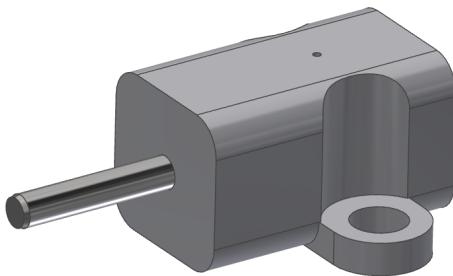


Abb. 38: Schneidmechanismus mit Bohrbuchse, Passstift und Gehäuse

SMN-Aktuator

Dieser für den Schneidmechanismus vorgesehene Aktuator erweist sich als Konstruktionsfehler. Ursprünglich ist ein Elektromagnet vorgesehen, der einen Hebel betätigen soll, welcher wiederum auf den Passstift des Schneidmechanismus drückt. Aufgrund mangelnder Recherche hinsichtlich der Funktionsweise dieses Elektromagneten kann er jedoch nicht in dieser Weise verwendet werden. Zukünftige Versionen des Schneidmechanismus könnten daher auf alternative Antriebe wie Servomotoren oder Solenoid-Magnete setzen, um eine präzisere und leistungsstärkere Schneidbewegung zu gewährleisten.

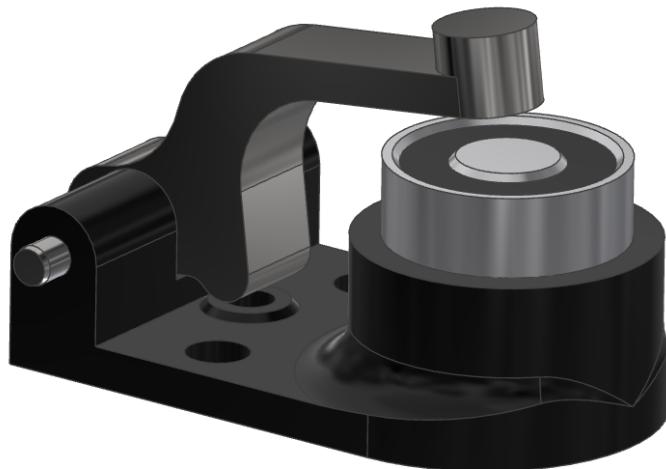


Abb. 39: SMN-Aktuator mit Elektromagnet

Insert

Das Insert dient der Faserzentrierung und befindet sich innerhalb des Schmelzgangs. Ursprünglich besitzt es drei Verstrebungen, jedoch zeigt eine Strömungssimulation, dass dies zu einer ungleichmäßigen Strömung und möglichen Faserabweichungen führt. Durch Entfernen einer Strebe kann eine homogener Strömung erreicht werden.

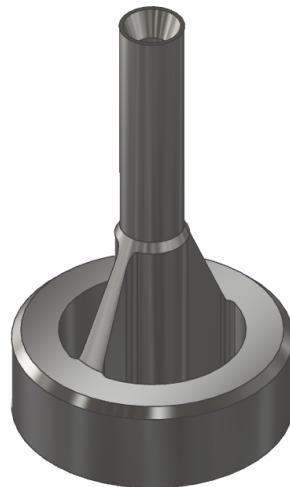


Abb. 40: Insert mit ursprünglicher Verstrebung

3.10.2 Simulationen

Bauteilspezifische Materialien

- **Isolationsstifte:** Den Stiften wurde Titan als Simulationsmaterial zugewiesen.
- **Schrauben:** Den Schrauben wurde Stahl als Simulationsmaterial zugewiesen.

Thermische Simulation V5.0.0

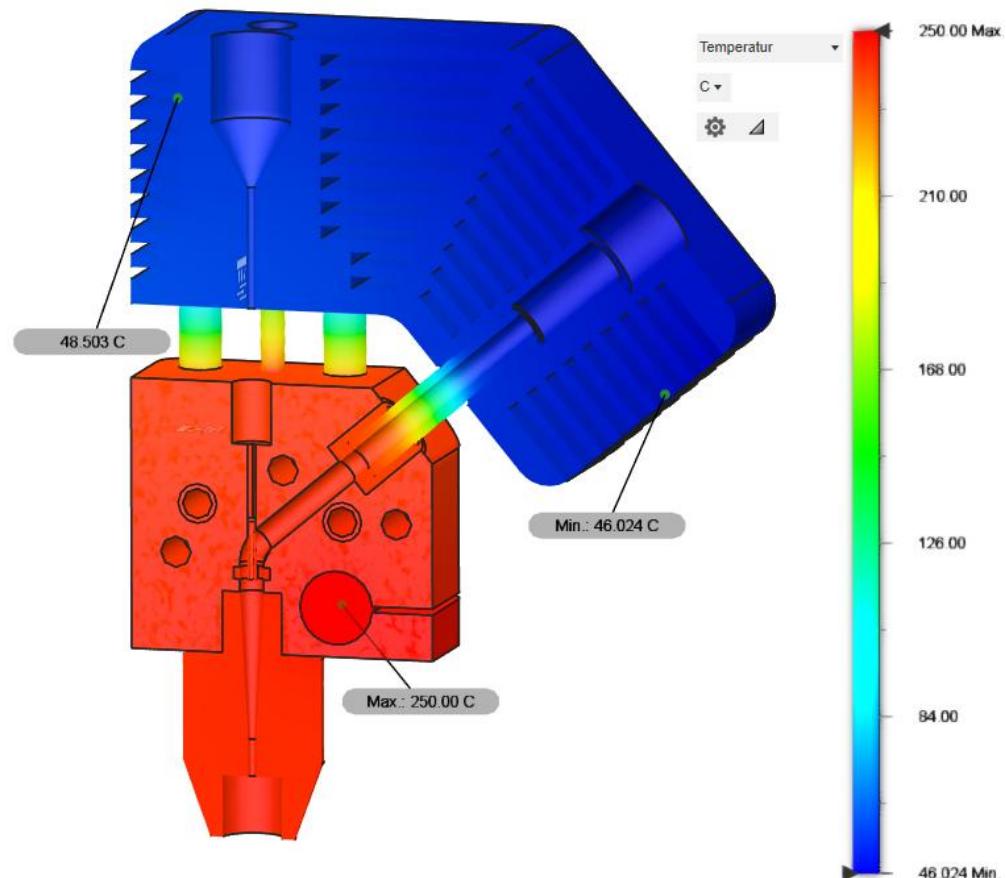


Abb. 41: Thermische Simulation V5.0.0 - Temperatur

Die thermische Simulation der Version 5.0.0 zeigt, dass das Coldend eine maximale Temperatur von 48,503 °C erreicht. Dieses Ergebnis erweist sich als unzureichend. Zur detaillierten Ursachenanalyse wird die Darstellung von der absoluten Temperaturverteilung auf den Temperaturgradienten umgestellt und die Skalierung entsprechend angepasst.

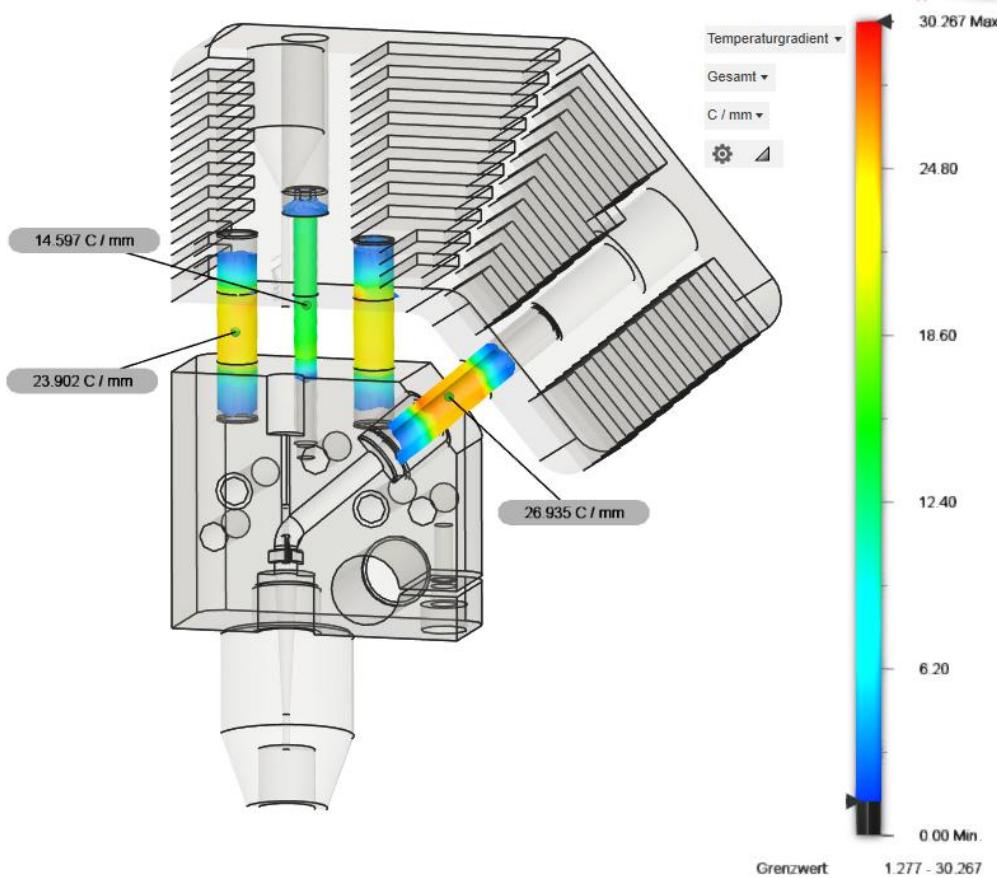


Abb. 42: Thermische Simulation V5.0.0 - Temperaturgradient

Die Analyse ergibt, dass die Titanstifte eine Wärmeleitfähigkeit von $23,902 \frac{\text{°C}}{\text{mm}}$ aufweisen, die Schraube $14,597 \frac{\text{°C}}{\text{mm}}$ und das Heatbreak $26,935 \frac{\text{°C}}{\text{mm}}$. Eine mögliche Optimierungsmaßnahme besteht in der Verlängerung der Titanstifte, um die Wärmeübertragung zu reduzieren, oder in einer gezielten Erhöhung des Temperaturgradienten. Ein Wärmeeintrag mit vergleichbarer Charakteristik wie beim Heatbreak stellt ein optimales Ziel dar.

Thermische Simulation V5.3.0

Zur Verbesserung des Temperaturgradienten werden die Titanstifte hohl ausgeführt, um die effektive Wärmeübertragungsfläche sowie das Volumen zu reduzieren. Dadurch verringert sich die Wärmeleitung innerhalb der Stifte, wodurch eine geringere thermische Übertragung auf angrenzende Komponenten erreicht wird. Die Gesamtlänge der Stifte bleibt dabei unverändert.

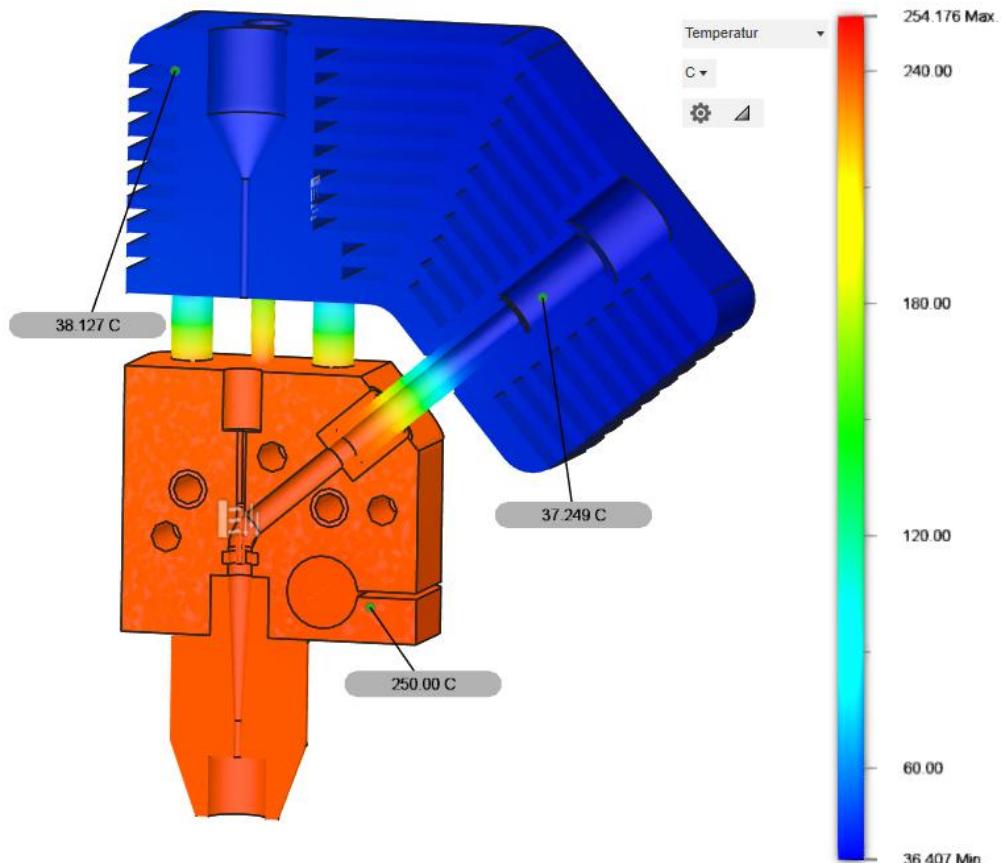


Abb. 43: Thermische Simulation V5.3.0 - Temperatur

Die thermische Simulation der Version 5.3.0 zeigt, dass das Coldend eine maximale Temperatur von 38,127 °C erreicht. Im Vergleich zur vorherigen Version wird eine signifikante Verbesserung der Temperaturverteilung festgestellt. Die Reduzierung der Wärmeübertragung resultiert aus der Modifikation der Titanstifte, um die thermische Leitfähigkeit zu verringern.

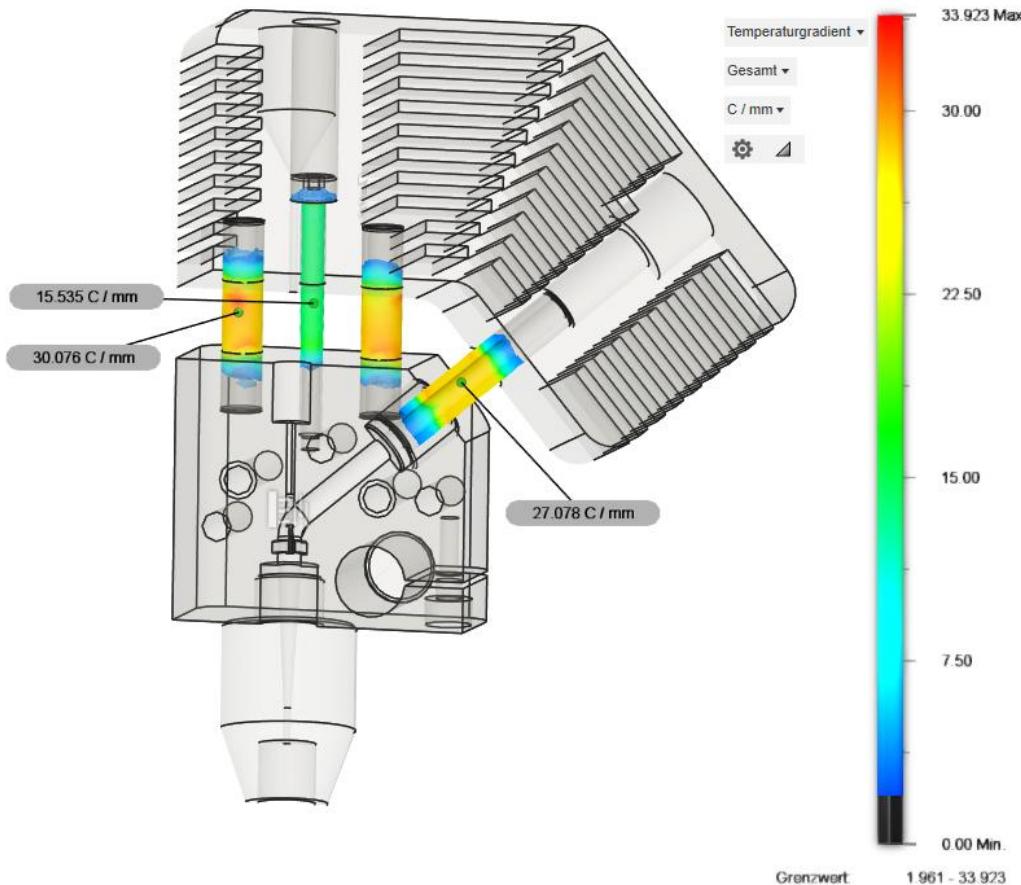


Abb. 44: Thermische Simulation V5.3.0 - Temperaturgradient

Zur genaueren Analyse wird die Darstellung auf den Temperaturgradienten umgestellt. Die Titanstifte weisen nun einen Temperaturgradienten von $30,076 \frac{^{\circ}\text{C}}{\text{mm}}$ auf, während das Heatbreak $27,078 \frac{^{\circ}\text{C}}{\text{mm}}$ überträgt. Die Temperaturgradientenabweichung des Heatbreaks zur vorherigen Simulation resultiert aus Simulationsgenauigkeiten. Diese Werte zeigen, dass die Wärmeeinleitung durch die strukturelle Anpassung der Stifte gezielt reduziert wurde. Besonders hervorzuheben ist die Angleichung des Temperaturgradienten an das Heatbreak, was eine gleichmäßige Temperaturverteilung ermöglicht.

Durch die Optimierung der Geometrie konnte somit eine effizientere thermische Entkopplung zwischen dem Hotend und dem Coldend erreicht werden, was eine stabilere Prozessführung bei der Verarbeitung temperaturkritischer Materialien ermöglicht.

Flusssimulation

Insert

Die Strömungssimulation des Inserts zeigt anfänglich eine ungleichmäßige Verteilung durch die drei Verstrebungen. Diese führen zu unkontrollierten Strömungsschwankungen, welche die Faserzentrierung beeinträchtigen können. Nach Anpassung auf zwei Verstrebungen kann ein gleichmäßigerer Fluss erreicht werden.

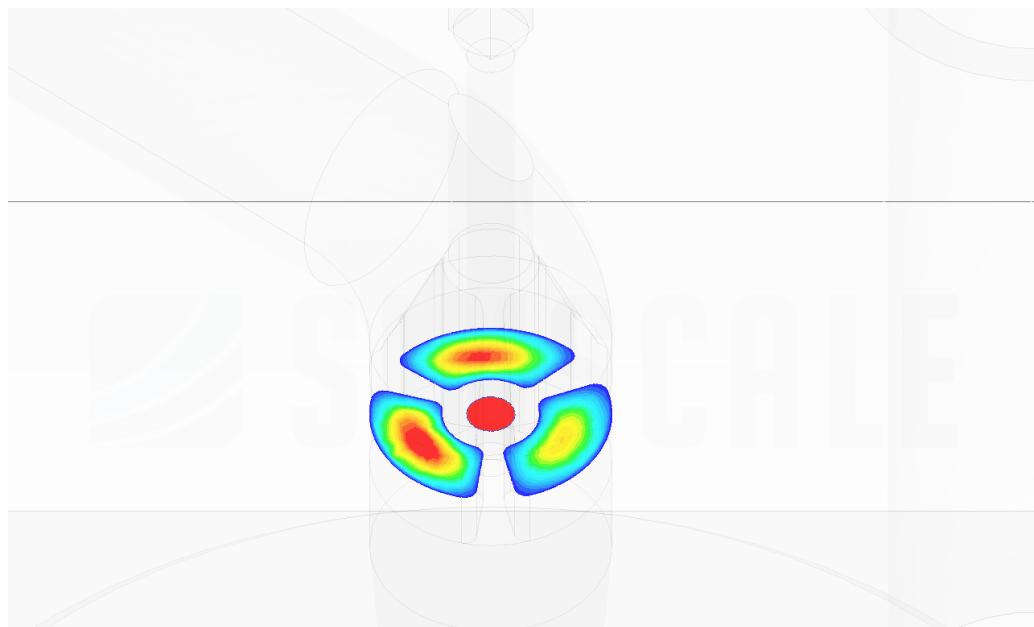


Abb. 45: Flusssimulation mit ungleichmäßiger Verteilung, 3D-Ansicht

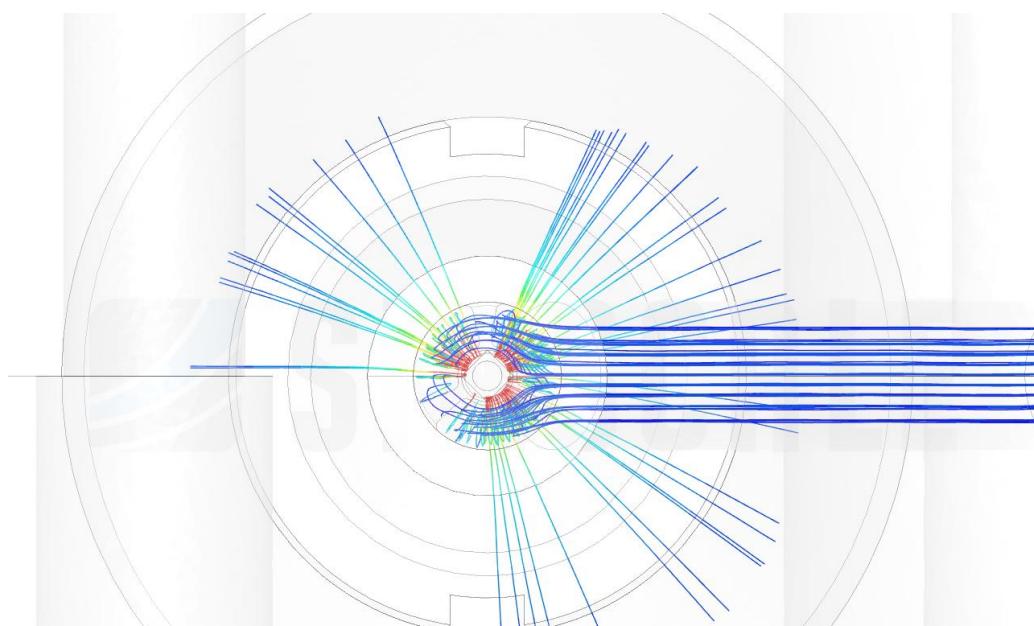


Abb. 46: Optimierte Flusssimulation mit gleichmäßiger Strömung, Draufsicht

Gehäuse/Druckkopf

Diese Strömungssimulation dient der Bewertung der Strömungsführung innerhalb des Gehäuses. Ziel ist die Analyse der Luftführung zu untersuchen. Dabei wird geprüft, ob die Luftströmung gezielt durch das Gehäuse geleitet wird oder unerwünschte Verwirbelungen und Stauzonen auftreten.

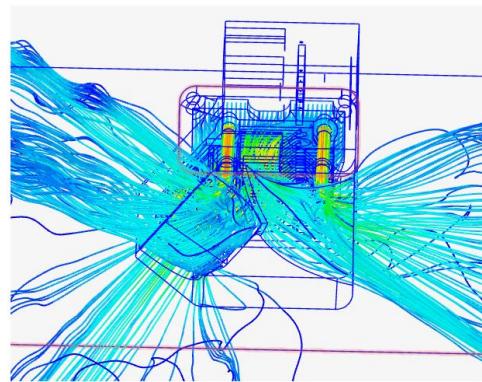


Abb. 47: Strömungssimulation Ansicht 1, V5.5.0

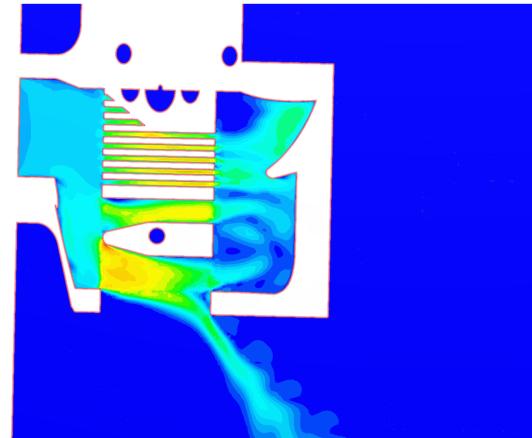


Abb. 48: Luftentweichung

Die Strömungssimulation zeigt die Luftführung innerhalb des Gehäuses anhand von Stromlinien. Es wird ersichtlich, dass die Kühlluft nicht ausschließlich durch die vorgesehenen Kanäle strömt, sondern teilweise unkontrolliert entweicht. Dies kann zu einer ineffizienten Kühlung und lokalen erhöhten Temperaturen führen.

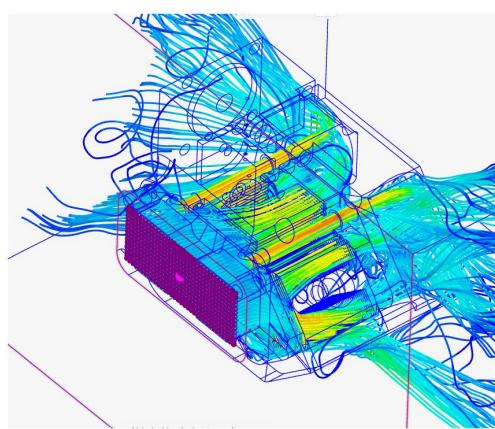


Abb. 49: Strömungssimulation Ansicht 2, V5.5.0

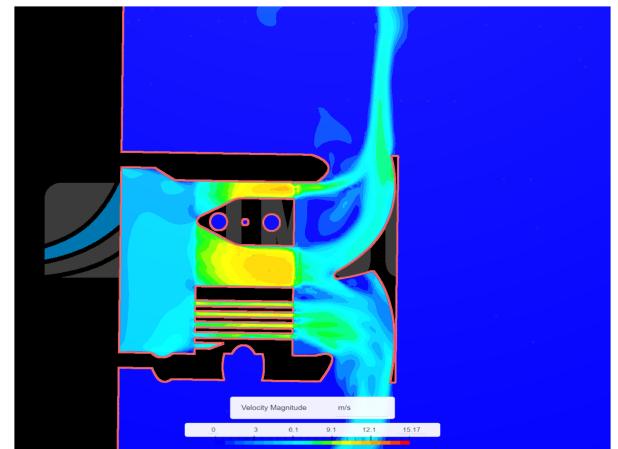


Abb. 50: Luftenabführung

Die erste Darstellung zeigt die Strömungslinien innerhalb des Systems. Es wird ersichtlich, dass die Luftströmung gezielt durch die Gehäusekonstruktion geleitet wird und eine gleichmäßige Verteilung erreicht.

Die zweite Darstellung stellt die Geschwindigkeitsverteilung der Luft im Querschnitt dar. Hier zeigt sich,

dass die Luftströmung effektiv auf beide Luftauslässe aufgeteilt wird, was eine funktionale Strömungsführung bestätigt.

3.10.3 Unterversionen

Der folgende Zeitstrahl zeigt die Entwicklungsschritte von Version 5 des Druckkopfes.

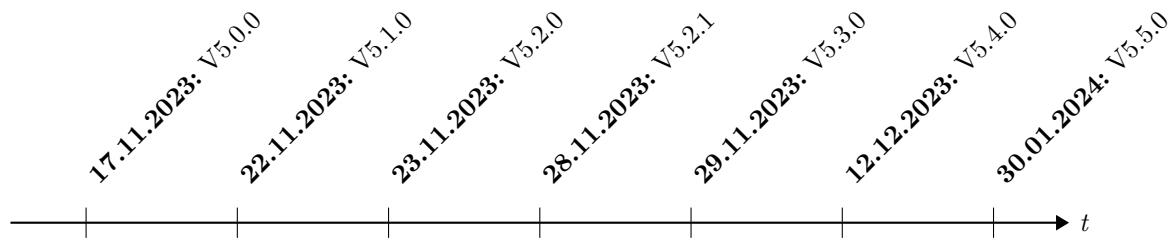


Tabelle 7: Arbeitszeitübersicht der Entwicklungsphasen von Version 5

Version	Kurzbeschreibung	Arbeitszeit
V5.0.0	Konstruktion des luftgekühlten Coldends mit 1-mm-Lamellen, thermische Analyse	03:22:42
V5.1.0	Beginn der Gehäuse-Konstruktion, Entwicklung eines Direct-Drive-Faservorschubs mit einem Nema-11-Motor	02:41:10
V5.2.0	Konstruktion der Luftkühlung mit einem 25x25x10 mm Lüfter, erste Planung der Luftabführung	02:05:02
V5.2.1	Konstruktion einer gehärteten Stahl-Druckdüse, Anpassung der Lüfterhalterung, Gewichtsbestimmung des Druckkopfs (365 g)	01:23:45
V5.3.0	Simulation der Luftführung, erste Konstruktion der Luftpustauslässe, Flusssimulation mit aktualisiertem CAD-Modell in SimScale, Vorbereitung für Conjugate Heat Transfer Analyse, gescheiterte Simulation, Korrektur notwendig, Fortschritte beim Faserextruder, Isolationsstifte hohl ausgeführt zur Reduktion der Wärmeleitung,	10:19:09
V5.4.0	Anpassung der Faservorschubrollen, Materialreduzierung zur Gewichtsoptimierung, Beginn der Konstruktion des Schneidmechanismus (SMN), Auswahl von Bohrbuchsen nach DIN 172, Berechnung der Schneidkraft, Recherche zur EDM-Fertigung von 0.5-mm-Durchmessern, Passungsauswahl (3F7 und 3m6), Konzeptumsetzung des SMN, Wahl eines Elektromagneten als Aktuator, Konstruktion der SMN-Aktuator-Einheit	10:02:54
V5.5.0	Leichte Modifikationen am Gehäuse und SMN-Aktuator, Simulation des neuen Druckkopfs in Fusion 360, Flusssimulation des Heizblocks mit Insert zur Überprüfung der Faserzentrierung, Anpassung des Inserts von drei auf zwei Verstrebungen	04:52:22
Gesamt		34:47:04

3.10.4 Fazit

Die fünfte Version des Druckkopfes stellt eine signifikante Weiterentwicklung dar, insbesondere durch die Wiedereinführung des luftgekühlten Coldends und die Optimierung des Schneidmechanismus. Das überarbeitete Coldend mit feineren Kühlrippen und verbesserter Luftführung gewährleistet eine effizientere Wärmeableitung. Zudem ermöglicht die hohlgebohrte Ausführung der Titanstifte eine gezielte Reduktion der Wärmeübertragung, wodurch der Temperaturgradient optimiert wird.

Durch die Simulationsanalysen wurde festgestellt, dass die Luftströmung innerhalb des Gehäuses gezielt gelenkt wird und die Kühlleistung den Anforderungen entspricht. Die Ergebnisse zeigen eine verbesserte Strömungsverteilung, insbesondere durch die Anpassung der Luftabführungen und die Positionierung der Lüfter.

Die Konstruktion des Schneidmechanismus erweist sich als zentrale Herausforderung dieser Version. Während erste Konzepte mit Servomotoren geprüft wurden, fiel die Entscheidung auf einen Elektromagneten als Antrieb, um eine kompakte und leichte Bauweise zu ermöglichen. Nach weiteren Recherchen stellt sich heraus, dass der Elektromagnet nicht die angenommene Funktionsweise aufweist.

Die Simulationsergebnisse des Inserts ergaben eine leichte Ablenkung der Faser aufgrund der ursprünglichen Geometrie mit drei Verstrebungen. Durch die Reduktion auf zwei Verstrebungen konnte eine gleichmäßige Führung erreicht werden.

Insgesamt zeigt Version 5 eine hohe funktionale Reife, insbesondere durch die aerodynamischen Verbesserungen und thermischen Optimierungen. Künftige Entwicklungen sollten sich auf eine weitere Verfeinerung des Schneidmechanismus und eine gezielte Steuerung der Luftströmung konzentrieren, um die Zuverlässigkeit und Effizienz des Druckkopfes weiter zu steigern.

3.11 Version 6

Die sechste Version des Druckkopfes stellt eine umfassende Weiterentwicklung der bestehenden Konstruktion dar. Der Fokus liegt auf der Optimierung des Schneidmechanismus (SMN), der Verbesserung des thermischen Managements durch Komponenten von Slice Engineering sowie der thermischen Analyse zur Validierung der Wärmeableitung.

3.11.1 Konstruktion der sechsten Version

Druckkopf

Die Konstruktion des Druckkopfes erfährt in dieser Version eine grundlegende Modifikation. Das gesamte Gehäuse sowie die Anordnung der Kühlkomponenten werden neu konzipiert, um die thermische Effizienz zu maximieren und die Montage zu vereinfachen. Auch wird das Gewicht, durch die Neukonstruktion des Gehäuses und der Abstand der Düse zur Kinematik des Druckers verringert, was zu weniger Vibrationen während des Drucks führt.

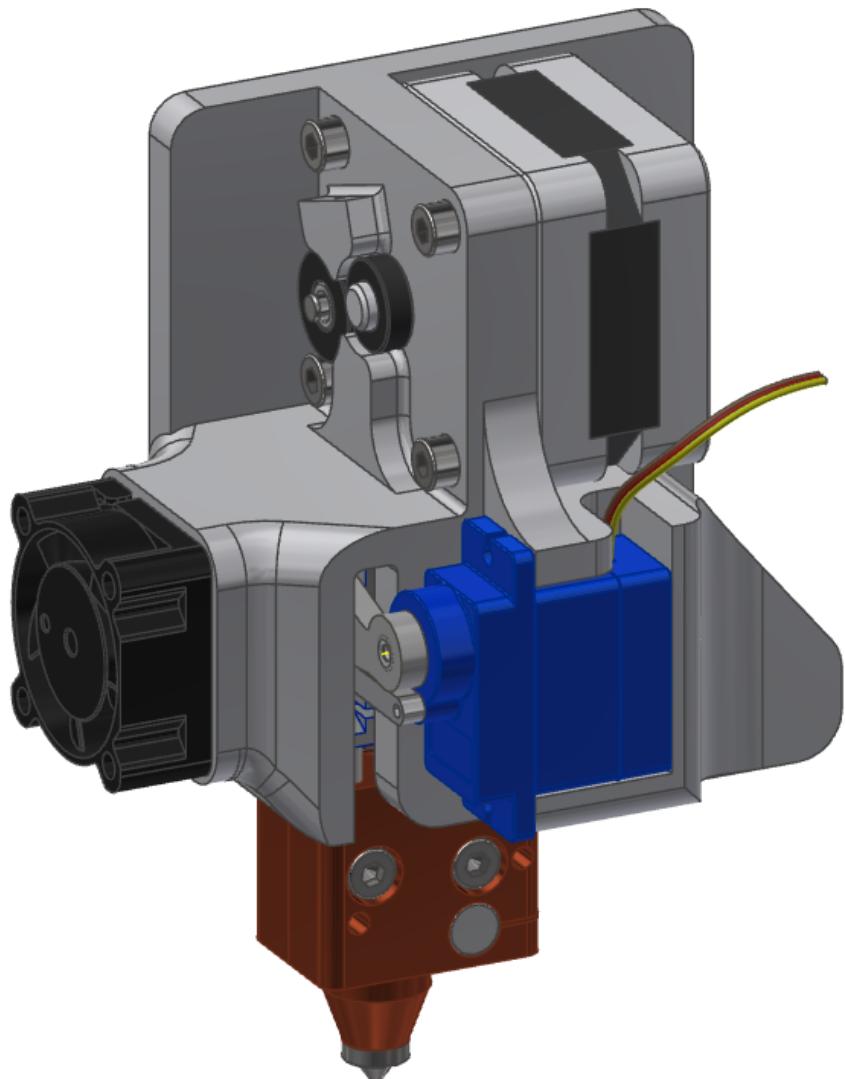


Abb. 51: Gesamtansicht des Druckkopfes, Version 6.0.1

Coldend

Zur Optimierung der thermischen Leistung wird das Coldend nicht mehr als Eigenkonstruktion umgesetzt, sondern ein Coldend von Slice Engineering integriert. Die Notwendigkeit zusätzlicher thermischer Simulationen entfällt, da die Effizienz dieses Coldends bereits durch den Hersteller validiert ist.

Dennoch ist die Fertigung eines spezifischen Abschnitts des Coldends erforderlich, da dieser für die Umhausung des Schneidmechanismus verantwortlich ist und eine stabile Befestigung des gesamten Hotends am Gehäuse gewährleistet.

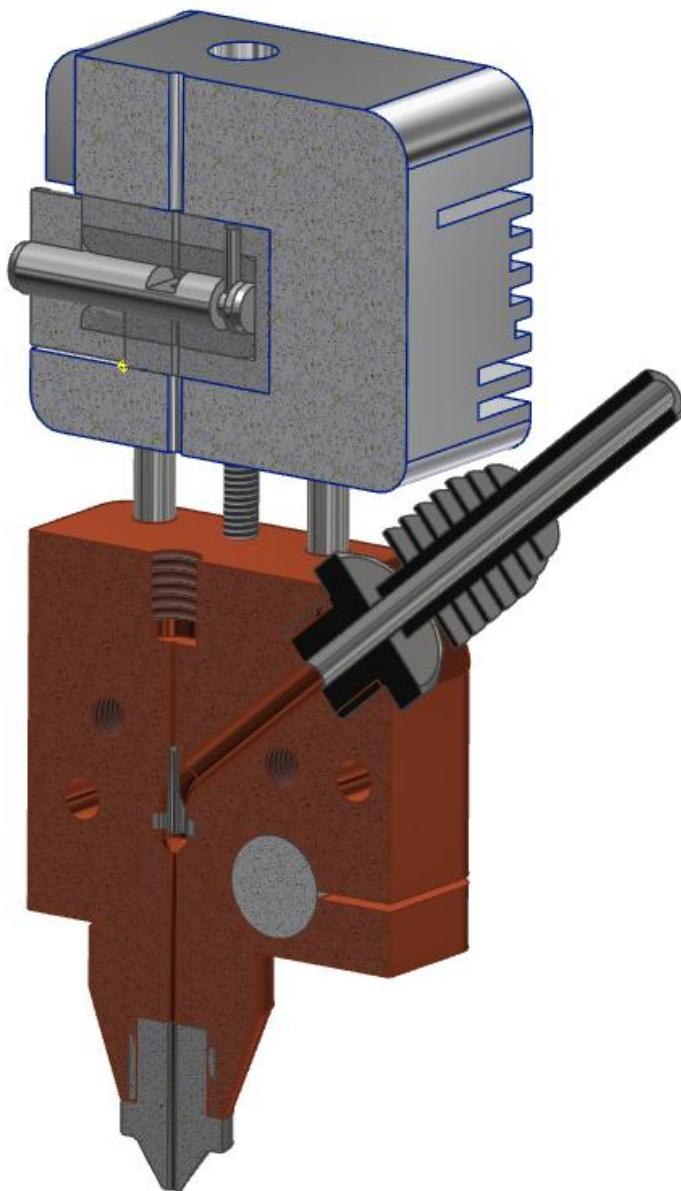


Abb. 52: Druckkopf, Version 6.0.1

Faservorschub

In der überarbeiteten Konstruktion wird ein NEMA-14-Schrittmotor mit den Abmessungen 35,2 mm x 35,2 mm x 20 mm verwendet, der in umgekehrter Ausrichtung zur Vorgängerversion installiert ist. Um die Reibung im Faserkanal zu reduzieren, wird ein PTFE-Schlauch mit einem Außendurchmesser von 1 mm und einem Innendurchmesser von 0,5 mm eingesetzt, der passgenau eingeklebt wird. Dadurch wird ein störungsfreier Transport der Faser ermöglicht und das Risiko von Ablagerungen minimiert. Außerdem bleibt gegenüber dem NEMA-14-Motor ausreichend Bauraum für eine modular aufgebaute Steuerungsplatine mit den Maßen 35 mm x 35 mm x 40 mm, um die Anbindung des Druckkopfes an die Klipper-Steuerung zu realisieren.

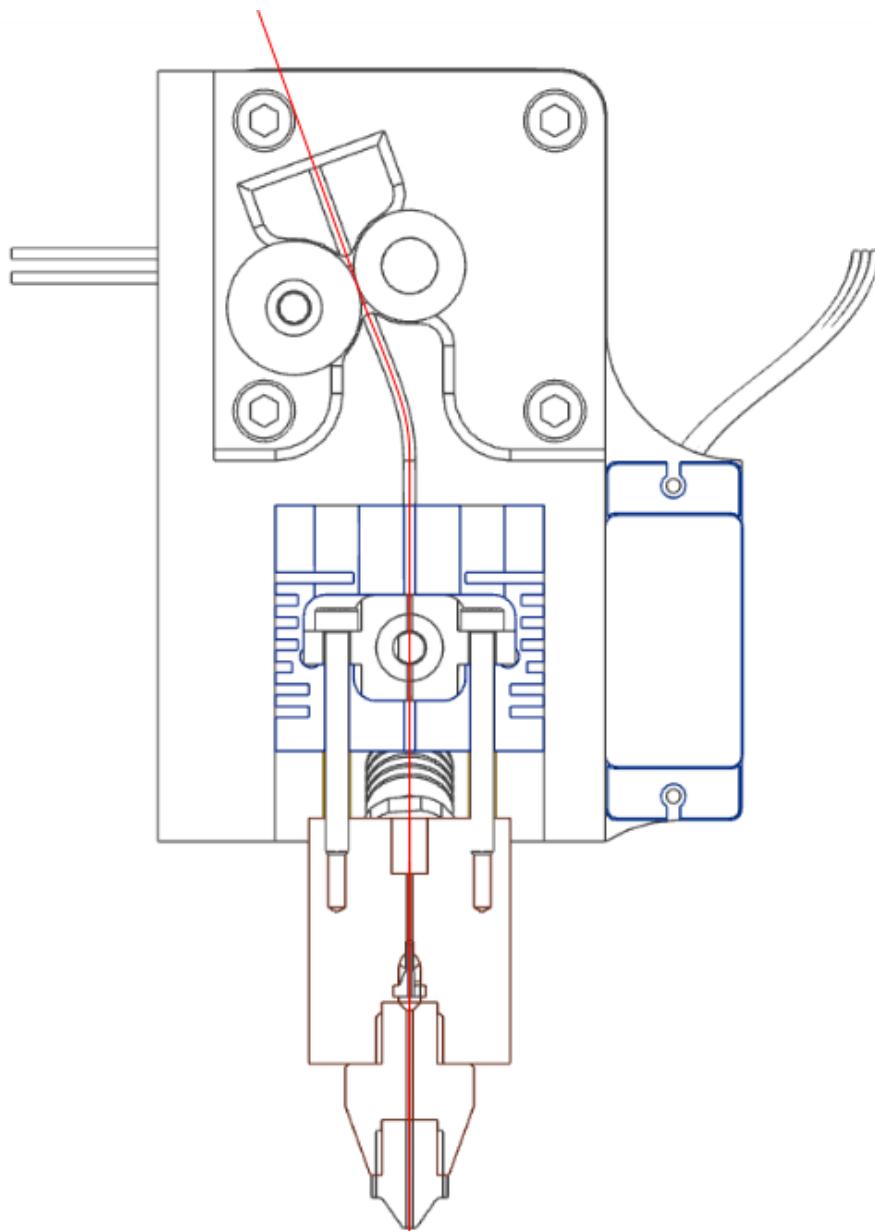


Abb. 53: Faserkanal

Isolationsstifte

Die Isolationsstifte stammen von Slice Engineering und sind speziell für eine thermische Entkopplung ausgelegt. Sie reduzieren die Wärmeübertragung an angrenzende Komponenten signifikant aufgrund der sehr dünnen Wandstärke von nur 0,11 mm. Die Validierung erfolgt durch thermische Simulationen.

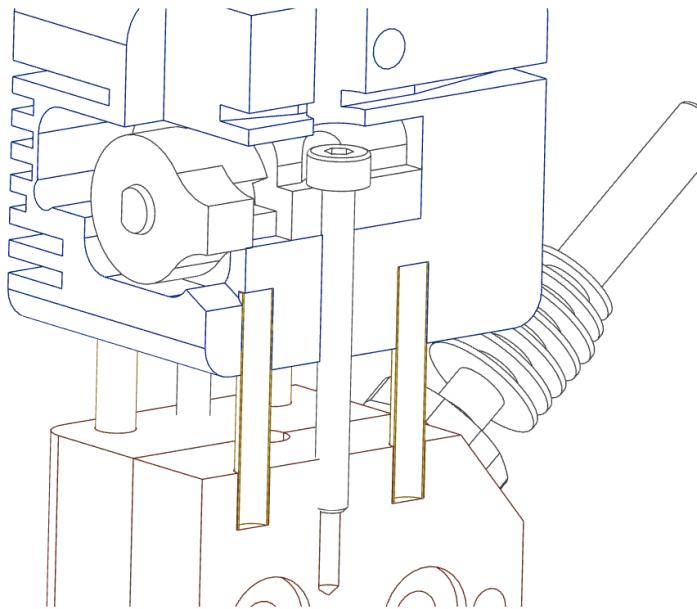


Abb. 54: Montage des Druckkopfes mit Isolationsstiften

Schneidmechanismus (SMN)

Der Schneidmechanismus wird in dieser Version weiter optimiert. Der Schneidstift führt nun eine Drehbewegung zur Schneidfunktion aus, anstelle einer linearen Bewegung. Eine umlaufende Einkerbung sichert den Schneidstift durch einen Sicherungsspin in der Bohrbuchse. In dieser befinden sich zwei Bohrungen – eine für den Sicherungsspin und eine für die Faserführung. Das Stahlgehäuse des SMN wird direkt mit dem Coldend zusammen im Hotend verschraubt (Abbildung 54), wobei präzise Referenzflächen eine exakte Positionierung gewährleisten.

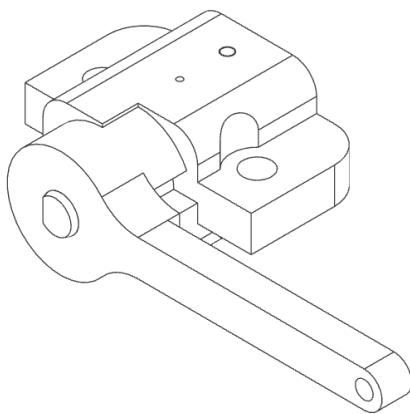


Abb. 55: Schneidmechanismus der neuen Version

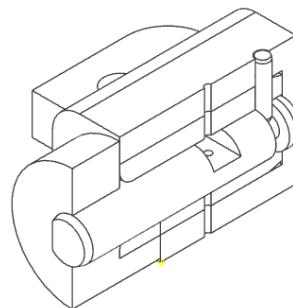


Abb. 56: Halbschnitt des Schneidmechanismus

SMN-Aktor

Die Betätigung des Schneidmechanismus erfolgt über einen Hebelmechanismus, der von einer Kurvenscheibe angetrieben wird. Diese ist mit einem Servomotor gekoppelt. Feste Anschläge am Gehäuse des SMN gewährleisten eine exakte Bewegungsausführung. Die Rückstellung des Hebels nach einem Schnitt erfolgt durch eine Zugfeder, die jedoch in der Konstruktion nicht enthalten ist, da die endgültige Auswahl erst getroffen wird.

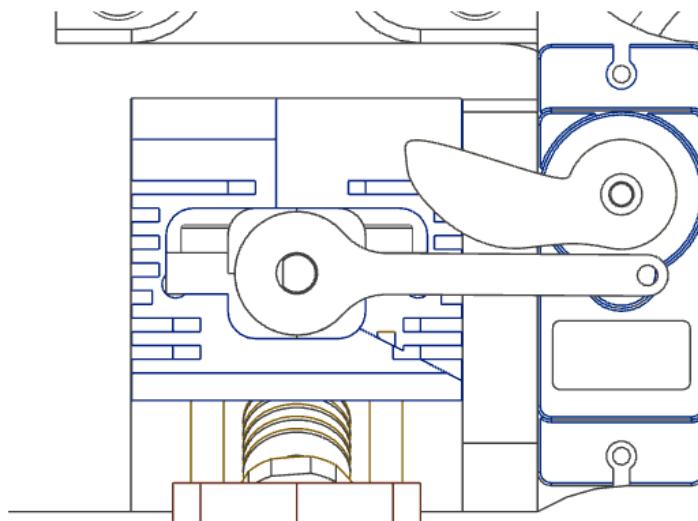


Abb. 57: SMN-Aktor und Hebelmechanismus

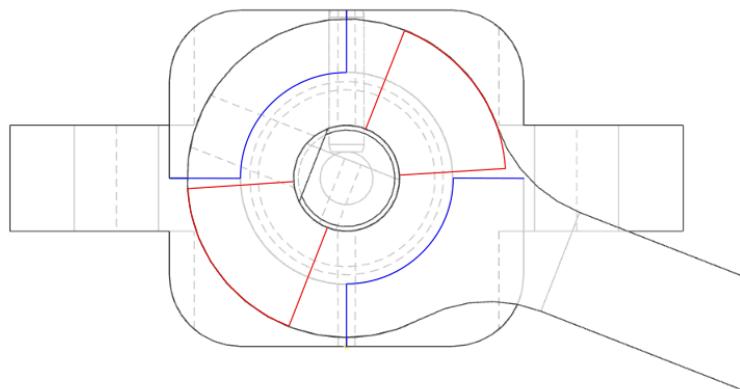


Abb. 58: Hardstops des SMN für exakte Positionierung

Düse

Die Düse erfährt eine konstruktive Anpassung. Der Filamentkanal wird verlängert, sodass die Durchmesserreduzierung auf den endgültigen Düsendurchmesser mehrere Millimeter vor der Austrittsöffnung erfolgt. Vorangegangene Tests zeigen, dass die Faser nicht zuverlässig mitgeführt wird, wenn das Filament nicht gleichmäßig fließt. Durch die Modifikation wird eine optimierte Synchronisation der Fließgeschwindigkeiten von Filament und Faser erreicht.

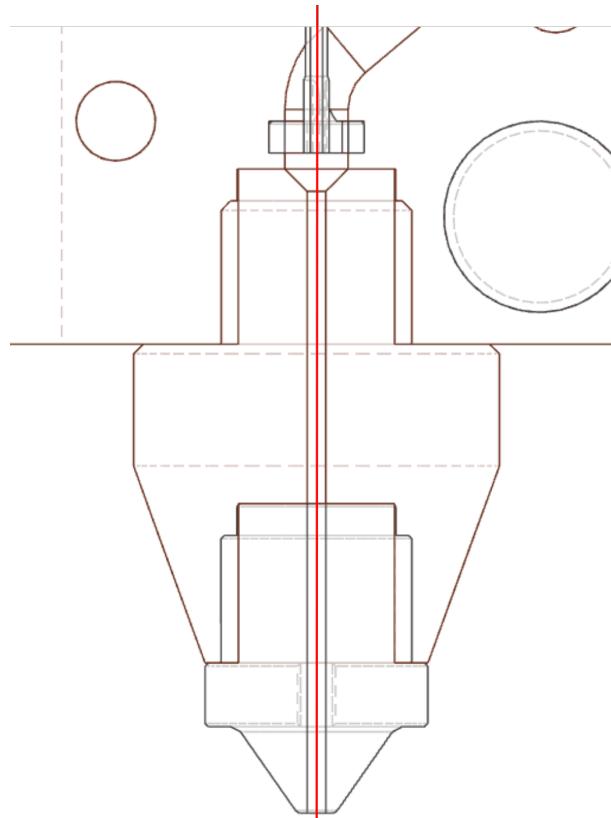


Abb. 59: Optimierte Düse mit verlängertem Filamentkanal

3.11.2 Thermische Simulation

Coldend

Da das Coldend von Slice Engineering bereits für eine effiziente Wärmeableitung optimiert ist, erfolgt keine weitere Simulation. Die Validierung konzentriert sich auf die thermische Performance der Isolationsstifte und des Hotends.

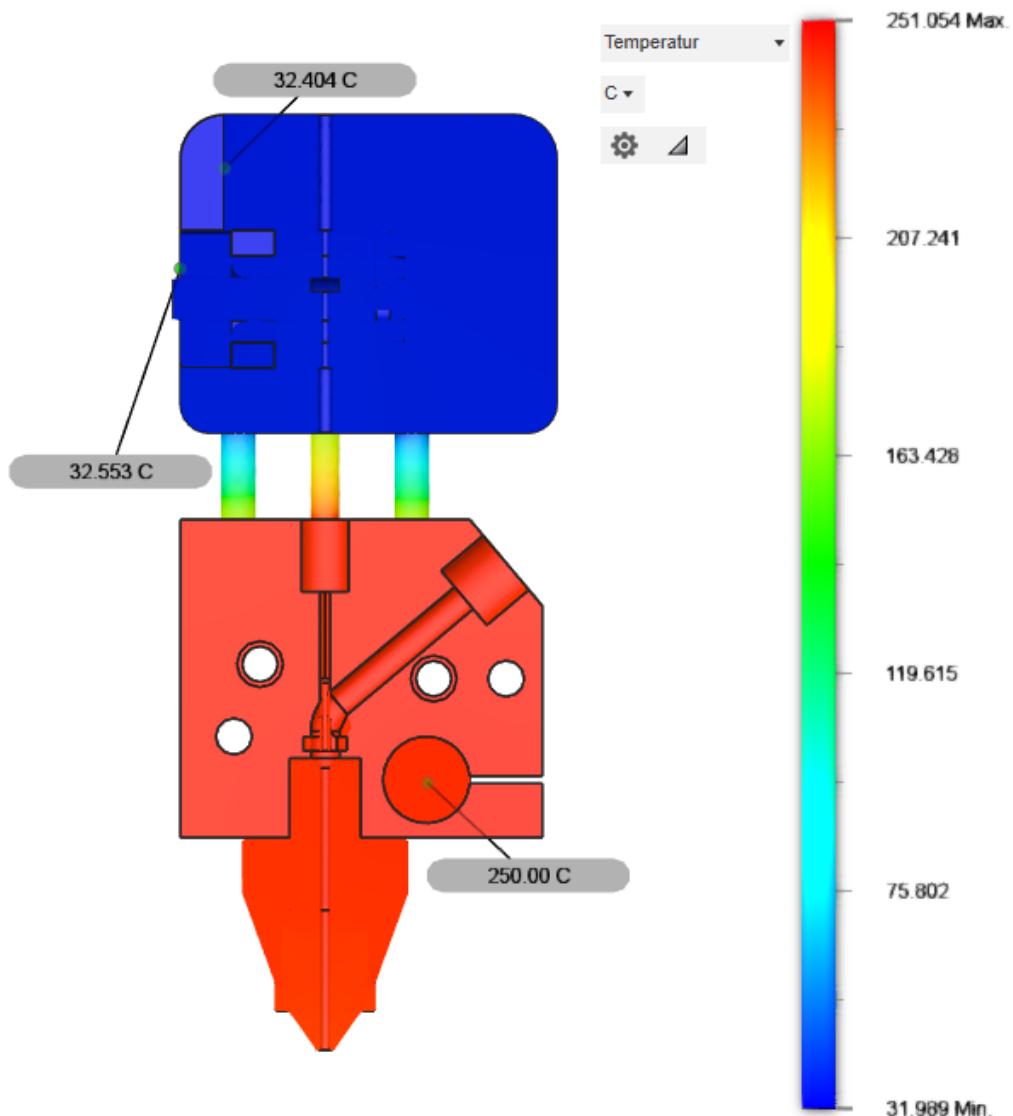


Abb. 60: Thermische Simulation der Isolationsstifte

Die thermische Simulation zeigt einen deutlichen Temperaturgradienten zwischen dem beheizten Druckkopf und dem Kühlkörper. Im Bereich der Düse wurde eine Maximaltemperatur von 250 °C ermittelt wie in den Rahmenbedingungen festgelegt wurde, während der Kühlkörper bei 32,404 °C verbleibt. Der Schneidmechanismus erwärmt sich auf 32,553 °C. Dies stellt keine Probleme in Hinsicht auf die Wärmeausdehnung dar.

SMN

Die thermische Simulation des SMN bestätigt eine effiziente Wärmeableitung, wobei die ermittelten Temperaturen im Bereich des Schneidmechanismus unkritisch bleiben. Die zur Visualisierung verwendete Skala wurde so angepasst, dass die lokale Wärmequelle am SMN deutlich erkennbar ist. Über Schraubverbindungen wird zwar ein Großteil der Wärme übertragen, dieser Effekt bleibt jedoch in tolerierbaren Grenzen. Eine weiterführende Optimierung ist daher nicht zwingend erforderlich, obgleich hierfür noch Potenzial besteht.

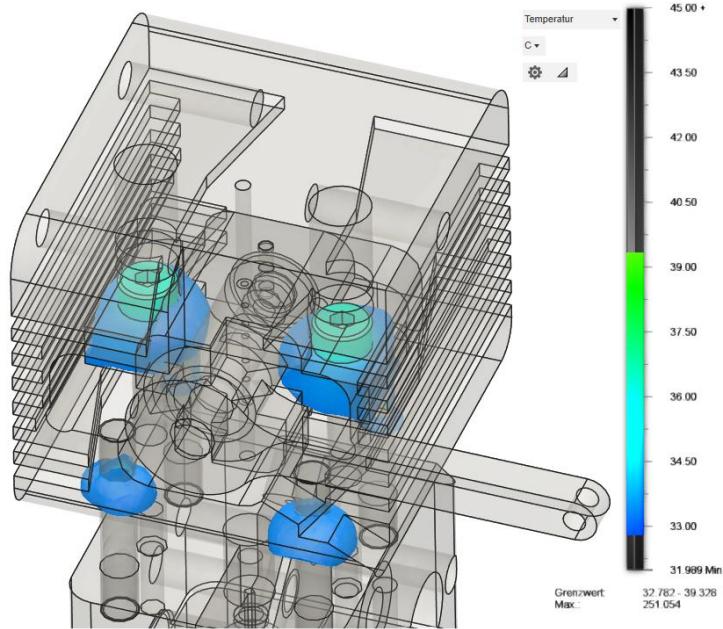


Abb. 61: Temperaturverteilung des SMN

3.11.3 Unterversionen

Der nachfolgende Zeitstrahl zeigt die Entwicklungsschritte von Version 6 des Druckkopfes.

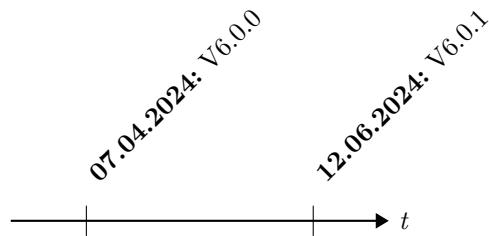


Tabelle 8: Arbeitszeitübersicht der Entwicklungsphasen von Version 6

Version	Beschreibung	Arbeitszeit
V6.0.0	Umfassende Überarbeitung des Druckkopfes mit neuem Coldend von Slice Engineering, Anpassung der Kühlstruktur sowie Gehäuseoptimierungen.	07:46:16
V6.0.1	Einführung von Anschlägen für den SMN, Optimierung des Schneidstiftwinkels, Sicherungspins ergänzt.	05:16:16
Gesamt		13:02:32

3.11.4 Fazit

Die in Version 6 durchgeführten Weiterentwicklungen des Druckkopfes führen zu einer deutlich verbesserten thermischen Stabilität und einer effizienteren Kühlung. Die Integration des Coldends von Slice Engineering sowie die konstruktive Anpassung der Düse tragen maßgeblich zur Reduktion von Wärmeverlusten und zu einer erhöhten Prozesssicherheit bei. Durch die Optimierung des Schneidmechanismus (SMN) konnte zudem die Funktionssicherheit beim Durchtrennen der Faser gesteigert werden. Die durchgeführten Simulationen zeigen, dass trotz unterschiedlicher Wärmequellen und einer teils komplexen Bauteilanordnung keine unzulässig hohen Temperaturen auftreten und somit kein zusätzlicher Optimierungsbedarf besteht.

3.12 Version 7 (aktuelle Version)

In Version 7 kommt ein anderes zugekauftes Heatbreak und Coldend zum Einsatz da diese wesentlich kostengünstiger und leichter zu ersetzen ist. Die Dichtflächen des Split-Hotends werden verringert, um einen höheren Anpressdruck und infolgedessen eine effizientere Abdichtung zu erzielen. Ein Abschnitt einer G22-Kanüle ersetzt das ursprünglich, aufwendig zu fertigende Insert, indem diese mit einem Hochtemperatursilikon eingeklebt wird. Zusätzlich wird ein Bauteilkühler integriert, der das frisch extrudierte Filament direkt kühlt.

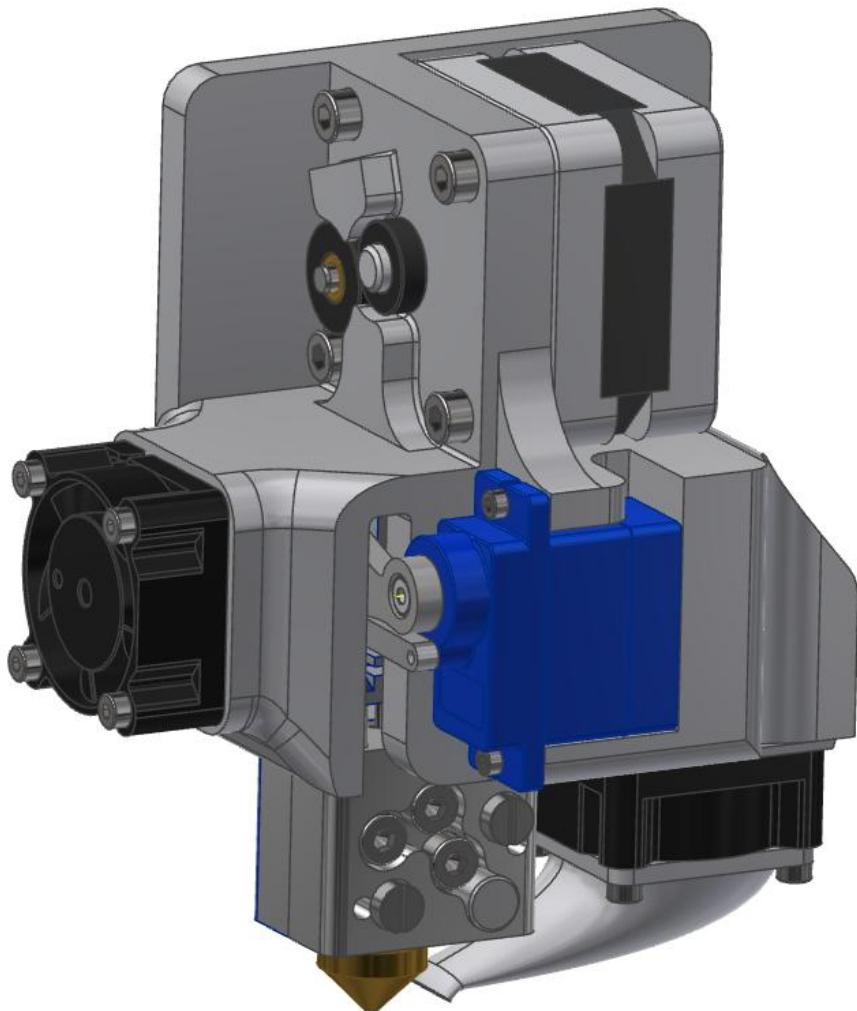


Abb. 62: Druckkopf in Version 7.0.0

3.12.1 Konstruktion der siebten Version

Hotend

Die Dichtflächen des Hotends werden rund um die Kontur vertieft, um die Fläche an denen die beiden Hälften aufeinander treffen zu verringern. Damit entsteht bei der gleichen Kraft ein deutlich höherer Druck auf den kontaktierenden Flächen. Somit entsteht eine bessere Abdichtung.

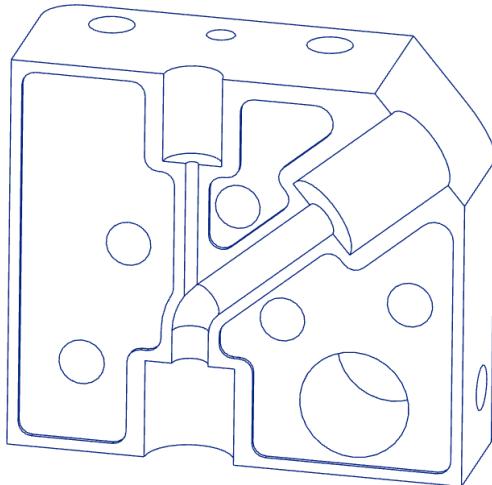


Abb. 63: Verbessertes Hotend

In Version 7.0.0 werden Gewinde zur Befestigung von zwei Temperatursensoren konstruiert. Die Temperatursensoren werden in dafür vorgesehene Positionierbohrungen eingesetzt und die Kabel mittels einer leichten Klemmung fixiert, was im Bereich der 3D-Druck-Hotends üblich ist. In Version 7.0.1 werden die Gewinde durchgängig in einer Hälfte des Hotends ausgeführt, sodass nach erfolgten Tests zwei Schrauben eingedreht werden können, um die beiden Split-Hotend-Hälften kontrolliert auseinanderzudrücken und so die Demontage zu erleichtern.

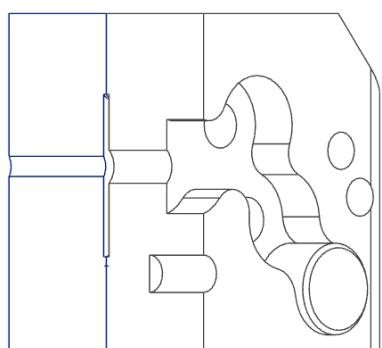


Abb. 64: Hotend, Version 7.0

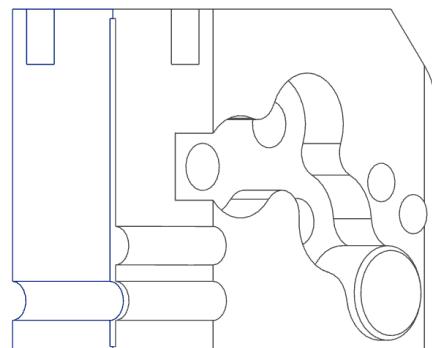


Abb. 65: Hotend, Version 7.0.1

Coldend

Das neue Coldend in Version 7.0.1 besitzt keine Kühlrippen, um Fertigungskosten zu senken. Allerdings zeigt die anschließende thermische Analyse, dass die Wärmeableitung dadurch an ihre Grenzen stößt. Die Temperatur im Bereich der Befestigungsgewinde steigt stärker als erwünscht, was eine erneute Anpassung der Kühlrippengestaltung in künftigen Versionen erfordert.

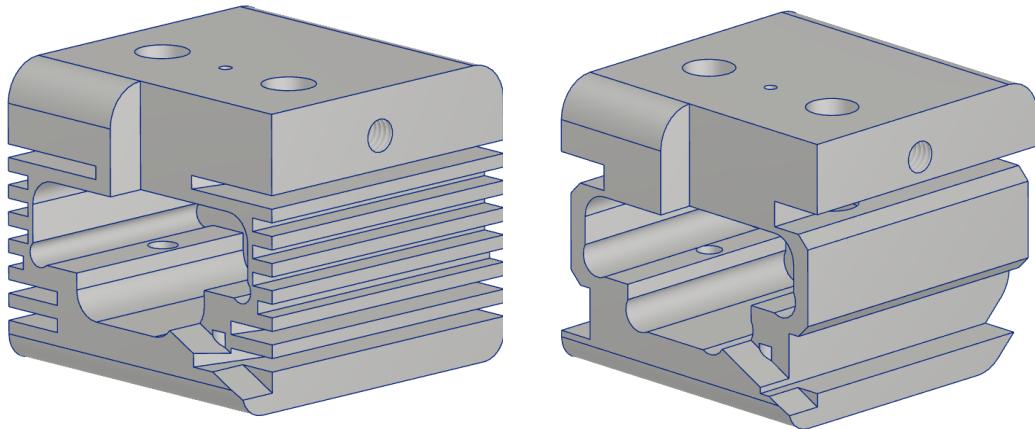


Abb. 66: Coldend Version 6.0.1

Abb. 67: Reduzierte Kühlrippen am Coldend in Version 7.0.1

Insert

Anstelle eines präzise gefertigten Custom-Inserts kommt ein Abschnitt einer handelsüblichen G22-Kanüle, wie auch schon in Version 6, zum Einsatz. Dadurch entfallen aufwendige Fertigungsschritte für sehr kleine Bohrungen. Die Kanüle wird auf die benötigte Länge abgelängt und anschließend in einer 90°-V-Kerbe des Hotends mittels hochtemperaturbeständigem Silikon fixiert. Dies gewährleistet eine präzise Zentrierung und einen minimalen Reibungswiderstand beim Zuführen der Faser. Das Insert stellt dabei ein Verschleißbauteil dar und ist bei jeder Wartung auszutauschen. Aufgrund der geringen Kosten von wenigen Cent pro Kanüle ist dieser Austausch wirtschaftlich unproblematisch.

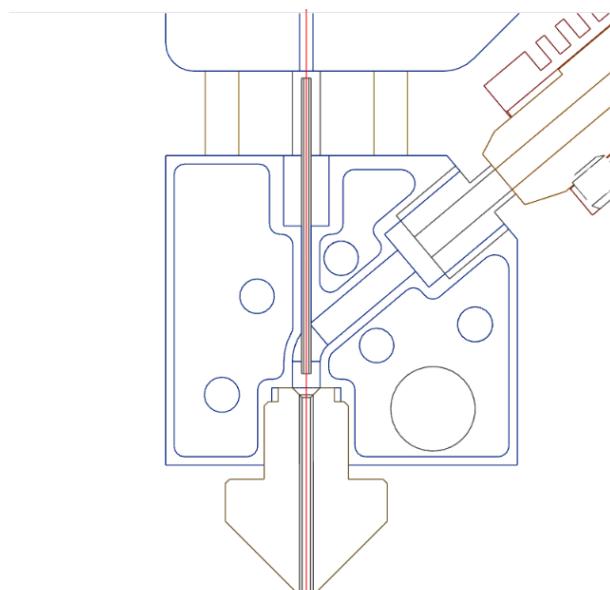


Abb. 68: Kanülenabschnitt als Faserführung im Hotend

Bauteilkühler

Zusätzlich zur Kühlung des Befestigungs-Coldends wird ein separater Bauteilkühler installiert, der das austretende Material unmittelbar nach der Düse mit Luft umströmt. Diese gezielte Kühlung verbessert die Druckergebnisse und vermindert Verzugseffekte, ohne die Temperatur im eigentlichen Schmelzbereich des Hotends stark zu beeinflussen. Dabei kühlt der Bauteillüfter gleichzeitig das Filament-Coldend.

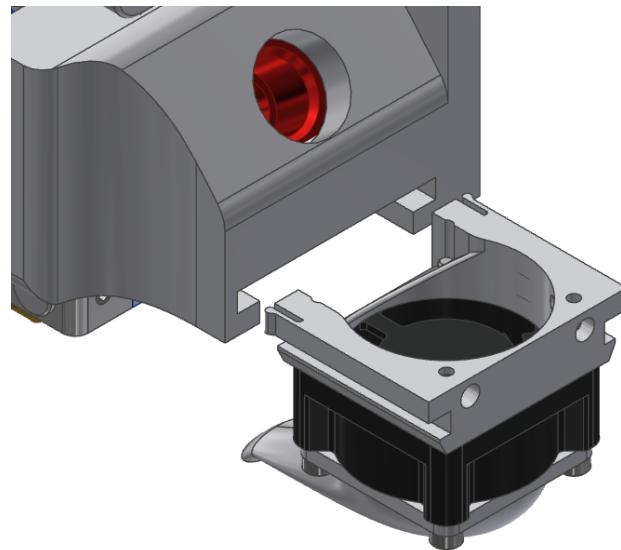


Abb. 69: Zusätzlicher Bauteilkühler, Version 7.1.0

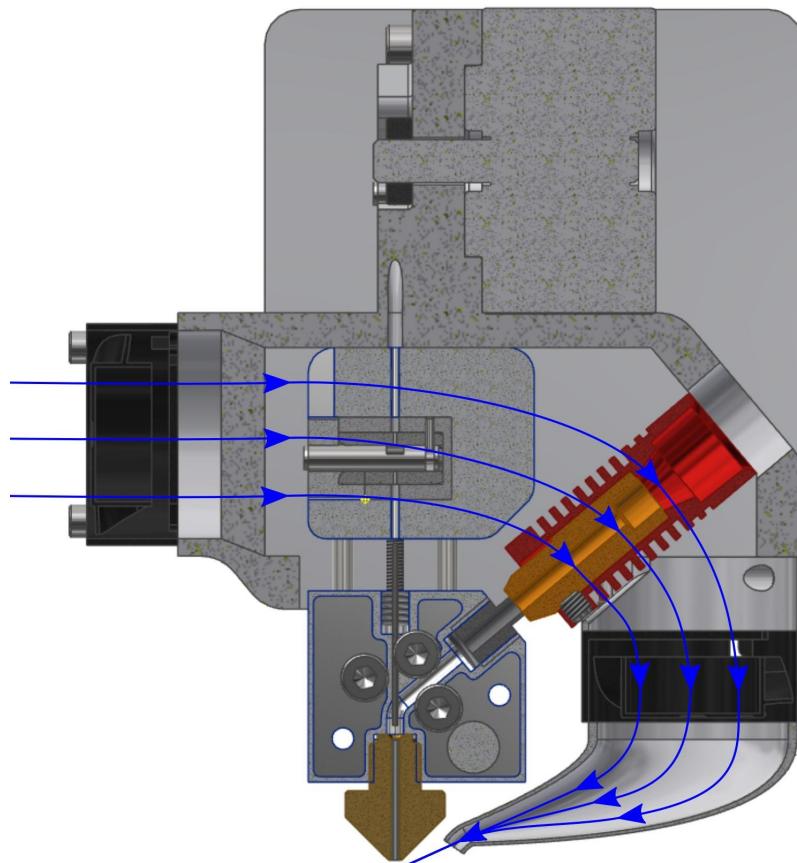


Abb. 70: Luftströmung im Druckkopf, Version 7.1.0

3.12.2 Thermische Simulation

Die Thermische Simualtion wird durchgeföhrte um das neue Kühlrippendesign zu überprüfen bevor dieses gefertigt wird.

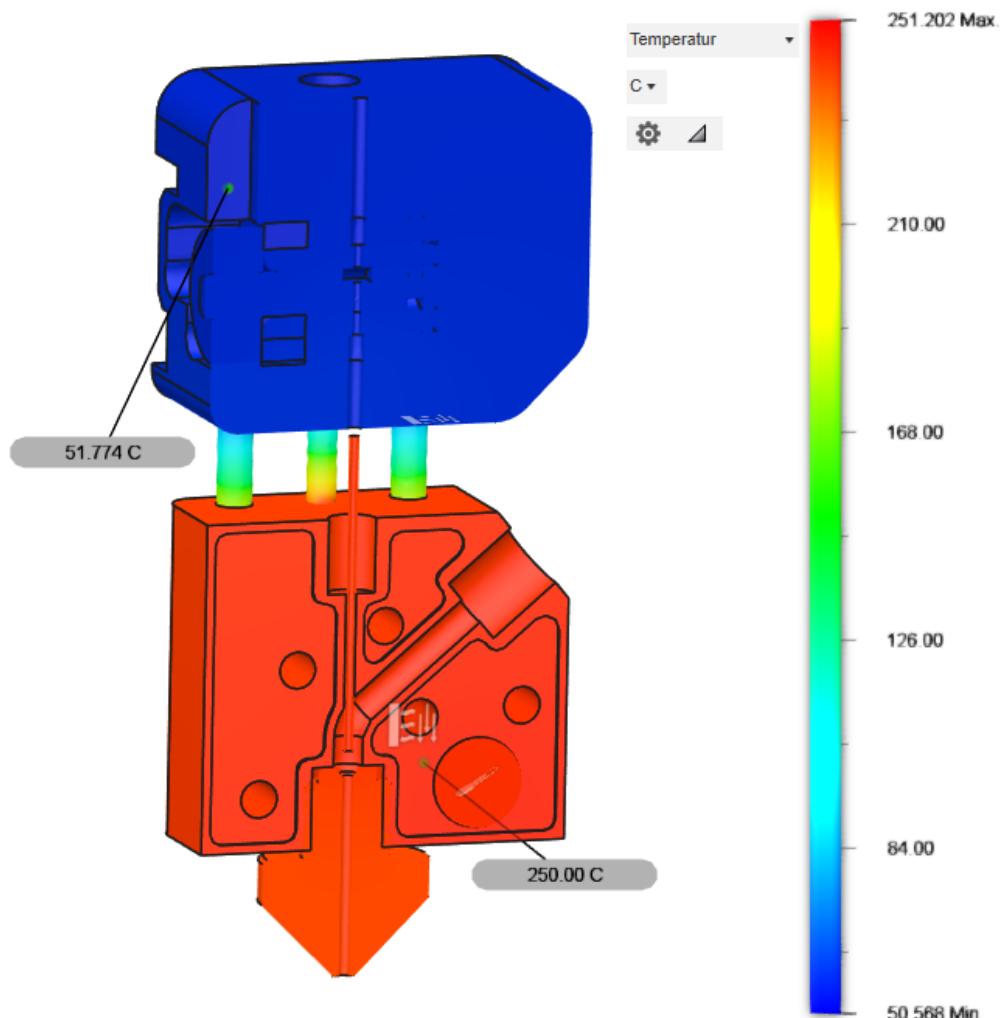


Abb. 71: Thermische Simulation für das vereinfachte Coldend in Version 7.0.1

Die thermische Simulation der Version 7.0.1 zeigt einen deutlichen Temperaturgradienten zwischen Hotend und Coldend. Während die maximale Temperatur im Bereich des Hotends, wie vorgesehen, stabil bei 250 °C liegt, erreicht das Coldend mit einer Temperatur von etwa 51,8 °C deutlich erhöhte Werte. Diese Temperatur übersteigt die zulässigen Grenzwerte für eine sichere thermische Trennung und zuverlässige Filamentkühlung.

Da diese Erwärmung zu Problemen im Filamenttransport föhren kann, ist die derzeitige Kühlrippenauslegung unzureichend. Infolgedessen wird in der nachfolgenden Version wieder das bereits validierte Coldend der Version 6 verwendet, um eine optimale thermische Entkopplung und eine zuverlässige Prozessstabilität sicherzustellen.

3.12.3 Unterversionen

Der nachfolgende Zeitstrahl zeigt die Entwicklungsschritte von Version 7:

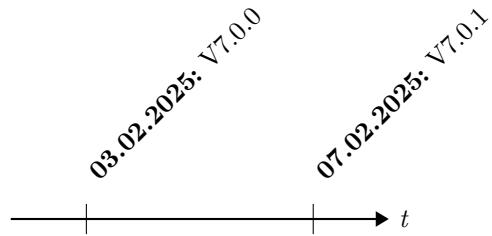


Tabelle 9: Arbeitszeitübersicht der Entwicklungsphasen von Version 7

Version	Beschreibung	Arbeitszeit
V7.0.0	Neues Heatbreak, verbesserte Abdichtung, G22-Kanüle, Bauteilkühler, Simulation	14:50:38
V7.1.0	Fehler behoben, durchgängige Sensorgewinde, Designänderungen, Tests	04:53:33
Gesamt		19:44:11

3.12.4 Fazit

Die siebte Version des Druckkopfes erzielt eine erhebliche Kostenreduktion sowie eine einfache Fertigung durch den Einsatz eines alternativen Heatbreaks und Coldends. Die Abdichtung zwischen den Hotend-Hälften verbessert sich aufgrund der verringerten Dichtfläche, wodurch ein höherer Anpressdruck generiert wird. Der Ersatz des aufwendig gefertigten Inserts durch einen günstigen Abschnitt einer G22-Kanüle reduziert deutlich den Fertigungsaufwand und ermöglicht gleichzeitig eine präzise Faserführung. Der neu hinzugefügte Bauteilkühler optimiert den Materialauftrag unmittelbar nach der Düse, ohne das thermische Gleichgewicht im Hotend negativ zu beeinflussen.

Allerdings zeigt die thermische Analyse, dass die Reduktion der Kühlrippen am Coldend nicht ausreichend Kühlung bietet und zu unerwünschter Temperaturerhöhung führt. Aus diesem Grund erfolgt in der nächsten Version eine Rückkehr zur bewährten Coldend-Konstruktion aus Version 6, um eine zuverlässige thermische Entkopplung sicherzustellen und die Prozessstabilität langfristig zu gewährleisten.

4 Elektronik

Dieses Kapitel beschäftigt sich mit der Entwicklung einer elektronischen Steuerungsplatine für den Druckkopf des Faserdruckers. Das Ziel ist die Konzeption und Umsetzung einer kompakten, zuverlässigen Platine, die alle notwendigen Funktionen für den präzisen Betrieb des Druckkopfes bereitstellt. Im Folgenden werden die einzelnen Schritte des Entwicklungsprozesses detailliert beschrieben: die Auswahl des grundlegenden Konzepts, die Auswahl geeigneter elektronischer Komponenten, die Erstellung des Schaltplans, durchgeführte Berechnungen und Simulationen zur Validierung des Designs, das Layout der Leiterplatte (PCB) sowie der abschließende Beschaffungs- und Lötprozess.

4.1 Elektronische Komponenten des Druckkopfes

Die Abbildung 72 zeigt die wichtigsten elektrischen und mechanischen Komponenten des Druckkopfes.

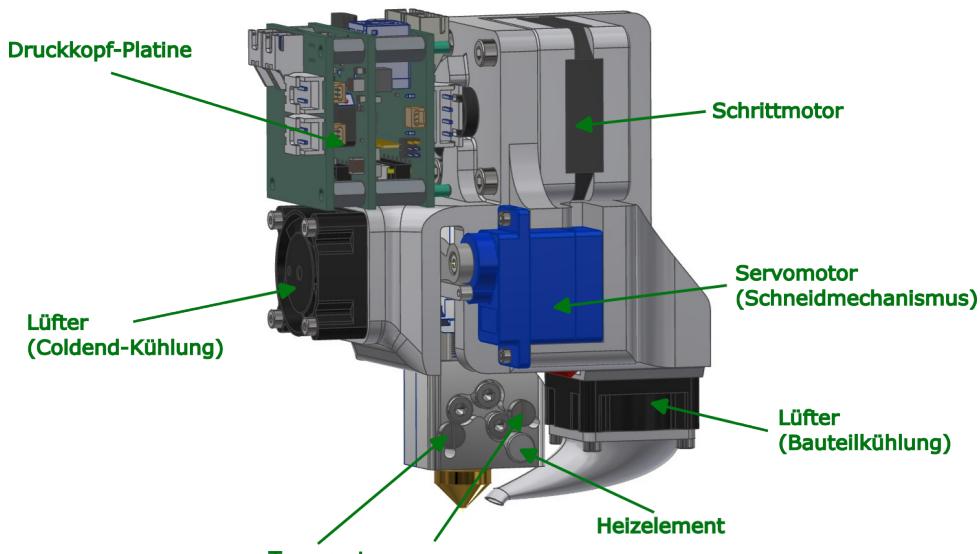


Abb. 72: Darstellung der elektrischen Komponenten im Druckkopf

Zu den zentralen Elementen gehören:

- **Druckkopf-Platine**: Steuereinheit zur Ansteuerung aller elektronischen Komponenten.
- **Schrittmotor**: Verantwortlich für den Vorschub der Faser.
- **Servomotor (Schneidmechanismus)**: Betätigt den Schneidmechanismus für die Fasertrennung.
- **Lüfter für Coldend-Kühlung**: Gewährleistet eine ausreichende Kühlung des Coldends, um ein frühzeitiges Schmelzen des Filaments zu verhindern.
- **Lüfter für Bauteilkühlung**: Unterstützt das schnelle Aushärten des extrudierten Materials zur Verbesserung der Druckqualität.
- **Heizelement**: Erzeugt die notwendige Temperatur für die Verarbeitung des Druckmaterials.
- **Temperatursensoren**: Messen die Temperatur des Heizelements und anderer relevanter Bereiche zur präzisen Steuerung des Druckprozesses.

4.2 Systemarchitektur der Elektronik

Die Abbildung 73 stellt die übergeordneten Verbindungen zwischen den elektronischen Komponenten des Systems dar. Der **Raspberry Pi** übernimmt die Hauptsteuerung und kommuniziert über eine **USB-Verbindung** sowohl mit dem **3D-Drucker-Mainboard** als auch mit der **Platine** des Druckkopfes.

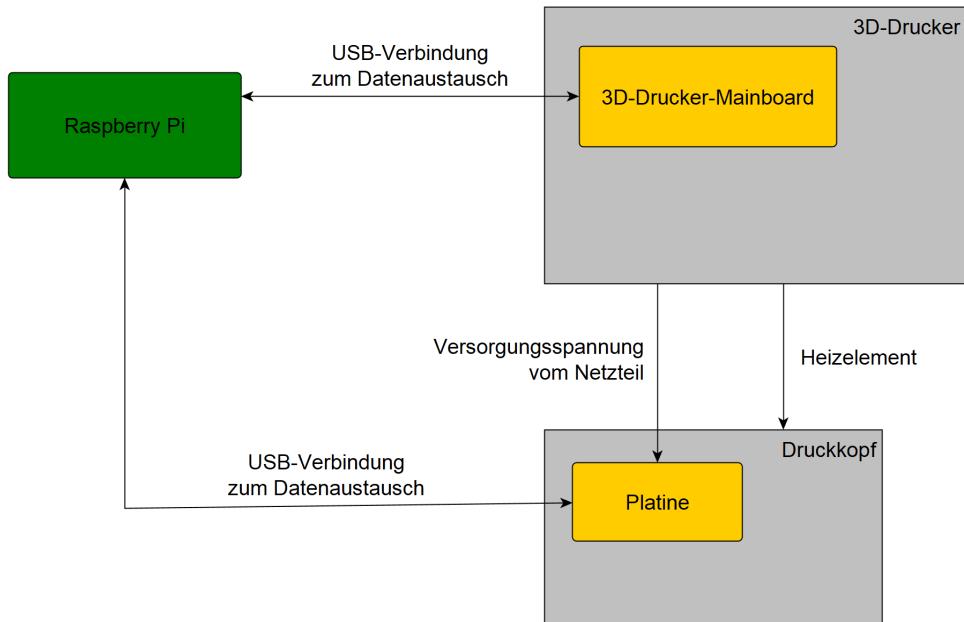


Abb. 73: Systemübersicht der Elektronik und deren Verbindungen

Die wesentlichen Schnittstellen sind:

- **Raspberry Pi – 3D-Drucker-Mainboard:** Die Kommunikation erfolgt über eine USB-Verbindung, wobei der Raspberry Pi Druckbefehle an das Mainboard sendet und Rückmeldungen über den Druckstatus empfängt.
- **Raspberry Pi – Druckkopf-Platine:** Eine zweite USB-Verbindung stellt die Kommunikation mit der speziell entwickelten Platine sicher.
- **Versorgungsspannung:** Die Platine im Druckkopf erhält ihre **Spannungsversorgung direkt vom Netzteil** des 3D-Druckers, um eine stabile und unabhängige Energieversorgung sicherzustellen.
- **Heizelement:** Das Heizelement des Druckkopfes wird direkt vom 3D-Drucker gesteuert, um die notwendige Extrusionstemperatur bereitzustellen.

Diese Architektur ermöglicht eine klare Trennung der Steuerung und erleichtert die Integration in bestehende 3D-Drucker-Systeme. Die Druckkopf-Platine erweitert die Funktionalität des Druckers, ohne in die bestehende Firmware des 3D-Druckers eingreifen zu müssen.

4.3 Anforderungen an die Elektronik

In diesem Abschnitt werden die Anforderungen an die speziell entwickelte Elektronikplatine dargestellt, die für den faserverlegenden 3D-Druckprozess benötigt wird. Folgende Aspekte sind dabei besonders wichtig:

- **Platz- und Gewichtsbeschränkungen:** Durch die engen Einbaubedingungen und die Anforderungen aus 8.1.4 *Plattformunabhängige Integration*, S.199 muss die Elektronik besonders kompakt und leicht ausgeführt werden.
- **Kompatibilität mit handelsüblichen 3D-Druckern:** Um die von gängigen 3D-Druckern bereitgestellten Grundfunktionen (z. B. Temperatursensoren, Lüfter und Heizelement) auch im entwickelten System zu realisieren, muss die Elektronik die entsprechenden Schnittstellen und Steuerungen bereitstellen (8.1.4 *Plattformunabhängige Integration*, S.198).
- **Faservorschub (Extruder):** Für den faserverlegenden Druck wird ein zusätzlicher Motor für den Faservorschub benötigt (8.1.3 *Coldend*, S.197). Die Elektronik übernimmt hierbei die präzise Steuerung.
- **Schneidmechanismus:** Ein **Servo-Motor** betätigt einen Hebel zum Schneiden der Faser (8.1.3 *Coldend*, S.197), was eine zuverlässige und wiederholgenaue Ansteuerung erfordert.
- **Kühlung und Temperaturmanagement:**
 - **Bauteilkühlung:** Zur Gewährleistung einer hochwertigen Druckqualität und Oberflächenbeschaffenheit (8.1.1 *Anforderungen Druckqualität*, S.193) können zwei Lüfter softwareseitig in ihrer Drehrichtung geändert werden. Dies erfordert den Einsatz einer H-Brücke bzw. eines Motorcontrollers.
 - **Kühlung von Coldend und Platine:** Entsprechend 8.1.3 *Coldend*, S.196 ist ein weiteres Lüfterpaar vorgesehen, das über MOSFETs angesteuert wird und bei Bedarf auch die Platine kühlen kann.
- **Systemintegration und Firmware:** Im entwickelten System wird ein **Raspberry Pi** mit der Firmware **Klipper** an den Drucker angeschlossen. Der Raspberry Pi kommuniziert dabei mit der Elektronikplatine am Druckkopf und übernimmt die übergeordnete Steuerung sämtlicher Komponenten (8.1.6 *Firmware*, S.203).

Ziel dieser Entwicklung ist es, eine robuste, platz- und kosteneffiziente Lösung zu schaffen, die sich nahtlos in das Gesamtsystem integriert und alle wesentlichen Aufgaben – von der Sensorik und Aktorik über den Faservorschub bis hin zur Kühlung – in einem kompakten, wartungsarmen und leichtgewichtigen Design vereint.

4.4 Lösungsansätze und Konzepte zur Datenübertragung

In diesem Kapitel werden verschiedene Möglichkeiten zur Umsetzung der Datenübertragung zwischen dem Raspberry Pi und der Platine vorgestellt und bewertet.

4.4.1 Konzept 1 zur Datenübertragung: I²C-Verbindung

Beschreibung

Bei diesem Ansatz kommuniziert der *Raspberry Pi*, auf dem das Drucker-Firmware-System *Klipper* läuft, direkt mit der Platine über I²C. Um die Datenübertragung über eine längere Kabellänge sicherzustellen, wird auf eine differentielle Umsetzung des I²C-Signals gesetzt. Dies bedeutet, dass auf der Platine zusätzliche Transceiver-Chips erforderlich sind, um das I²C-Signal in ein differentielles Signal umzuwandeln (und umgekehrt).

Kosten

Die Kosten für dieses Konzept belaufen sich auf 10.30 € und setzen sich wie folgt zusammen:

- **I²C Bus Extender (PCA9615DPZ):** 3.70 €
- **12-Bit I²C ADC (ADS1015IDGSR):** 2.50 €
- **4-Bit GPIO (PCA9536D):** 2.78 € (2 Stück)
- **Lüftersteuerung (EMC2302-2-AIZL-TR):** 1.32 €

Hinweis

Die angegebenen Kosten beziehen sich ausschließlich auf die spezifischen Komponenten des Konzepts. Kosten, die bei beiden Konzepten gleich sind, wie z.B. Aktoren, Sensoren und Stecker, sind hier nicht mit einbezogen, da es sich um einen Vergleich der beiden Konzepte handelt.

Vorteile

- **Direkte Kommunikation:** Der *Raspberry Pi* kann die Platine unmittelbar ansteuern, ohne dass ein zweiter Mikrocontroller notwendig ist.
- **Einfache Integration mit Klipper:** Da *Klipper* lediglich I²C-Geräte verwalten muss, ist das Setup softwareseitig recht überschaubar.
- **Viele Leitungen zur Platine hin:** Es können mehrere I²C-Komponenten angebunden werden, sodass eine gewisse Flexibilität bei der Anzahl der angeschlossenen Sensoren und Aktoren besteht.

Nachteile

- **Erweiterte Platine beim Raspberry Pi nötig:** Es ist eine zusätzliche Platine oder ein Modul erforderlich, das das normale I²C-Signal in ein differentielles I²C-Signal umwandelt.
- **Hohe Komponentenanzahl:** Unterschiedliche I²C-Chips (Transceiver, Multiplexer etc.) müssen beschafft und verbaut werden. Dadurch steigen sowohl die Materialkosten als auch die Komplexität.
- **Gesteigerter Platzbedarf:** Die vielen Bauteile (Treiber, Steckverbinder, Transceiver) erfordern mehr Leiterplattenfläche, was insbesondere in kompakten Anwendungen nachteilig ist.

4.4.2 Konzept 2 zur Datenübertragung: USB-C-Verbindung

Beschreibung

Auf der entwickelten Platine wird ein zusätzlicher Mikrocontroller integriert, der als zweites Mainboard fungiert. In *Klipper* kann er als zusätzliche Steuerungseinheit eingebunden werden. Die Kommunikation zwischen dem *Raspberry Pi* und der Platine erfolgt über eine *USB-C*-Verbindung.

Der Einsatz eines zweiten Mikrocontrollers bringt mehrere Vorteile mit sich: Zum einen ermöglicht er eine effiziente Aufgabenteilung, indem der *Raspberry Pi* weiterhin für komplexe Berechnungen und die Kommunikation verantwortlich bleibt, während der Mikrocontroller zeitkritische Steuerungsaufgaben mit hoher Präzision übernimmt. Zudem verfügt der Mikrocontroller über dedizierte Hardware-Schnittstellen zur direkten Ansteuerung der Komponenten, was die Implementierung vereinfacht und die Steuerungsgenauigkeit verbessert.

Kosten

Die Kosten für dieses Konzept belaufen sich auf 7.15 € und setzen sich wie folgt zusammen:

- **ATMEGA328P-AU:** 2.55 €
- **USB-Serial-Wandler (FT232RNL):** 4.60 €

Vorteile

- **Platzsparender Aufbau:** Da weniger externe Chips für die differentielle Signalübertragung benötigt werden, kann die Platine kompakter gestaltet werden.
- **Niedrigere Kosten:** Durch Wegfall teurer I²C-Transceiver und Multiplexer reduziert sich das Materialbudget.
- **Einfache Datenübertragung:** Die Kommunikation über *USB-C* ist robust, standardisiert und bietet ausreichend Bandbreite für die Druckeranwendungen.

Nachteile

- **Weiteres Firmware-Management:** Da ein zusätzlicher Mikrocontroller zum Einsatz kommt, müssen sowohl das Hauptsystem (*Raspberry Pi*) als auch das Zweitsystem (Mikrocontroller) gepflegt und aktualisiert werden.

4.4.3 Entscheidungsmatrix für die Platinenwahl

Um die beiden Konzepte objektiv bewerten zu können, wird eine Entscheidungsmatrix herangezogen. Damit lassen sich die wichtigsten Kriterien definieren und vergleichen.

Kriterien für die Auswahl

Die im Folgenden aufgeführten Kriterien werden gewichtet und anschließend für jedes Konzept bewertet:

- **Kosten:** Die Material- und Fertigungskosten sollen möglichst gering gehalten werden, wobei ein gutes Preis-Leistungs-Verhältnis entscheidend ist.
- **Platzbedarf:** Eine kompakte Bauweise ist erforderlich, um die Platine effizient in das bestehende System zu integrieren.
- **Software-Anpassung:** Der Anpassungsaufwand für die Softwareintegration sollte möglichst gering sein, um eine schnelle Implementierung zu ermöglichen.
- **Montage & Verkabelung:** Eine einfache und gut durchdachte Verkabelung minimiert den Installationsaufwand und erhöht die Zuverlässigkeit.
- **Stabilität der Kommunikation:** Eine zuverlässige Datenübertragung ist essenziell für eine stabile und fehlerfreie Funktion des Systems.
- **Wartungsaufwand & Fehlersuche:** Ein geringer Wartungsaufwand erleichtert die Diagnose und Reparatur im Fehlerfall.

Entscheidungsmatrix (Vergleich der beiden Platinenversionen)

Tabelle 10: Bewertung der Datenübertragung nach verschiedenen Kriterien

Kriterium	Gewichtung	I ² C	Gewichtet	USB-C	Gewichtet
Kosten	0.80	2	1.60	4	3.20
Platzbedarf	1.00	3	3.00	5	5.00
Software-Anpassung	0.40	4	1.60	3	1.20
Montage & Verkabelung	0.80	3	2.40	4	3.20
Stabilität d. Kommunikation	1.00	2	2.00	4	4.00
Wartungsaufwand/Fehlersuche	0.60	2	1.20	5	3.00
Gesamtbewertung		16	11.80	25	19.60

Aus der Tabelle 10 ist ersichtlich, dass die zweite Platinenversion (mit separatem Mikrocontroller) eine höhere Gesamtpunktzahl erreicht. Insbesondere bei den Kriterien *Platzbedarf*, *Stabilität der Kommunikation* und *Wartungsaufwand/Fehlersuche* schneidet dieses Konzept deutlich besser ab.

Entscheidung und Begründung

Anfangs ist nicht bekannt, dass Klipper die Konfiguration eines zweiten Mainboards unterstützt. Daher wird zunächst das **I²C-Konzept** umgesetzt, da die direkte Ansteuerung aller Komponenten durch den *Raspberry Pi* als praktikable Lösung erscheint. Erst später wird erkannt, dass eine Integration eines zweiten Mainboards möglich ist. Daraufhin wird eine Entscheidungsmatrix erstellt, um die beiden Konzepte objektiv zu vergleichen und die bessere Lösung zu bestimmen.

Erste Platinenversion (I²C-Version)

- **Einsatz als Prototyp:** Diese Version dient als erste Testplattform zur Untersuchung der Machbarkeit und zur Sammlung von Erfahrungen mit der Hardwareintegration. Die direkte Ansteuerung über den *Raspberry Pi* ermöglicht eine schnelle Inbetriebnahme, bringt jedoch Herausforderungen mit sich.
- **Learnings:** Der hohe Bauteilaufwand (BOM), die komplexe Verdrahtung und die Störanfälligkeit bei längeren Kabeln erhöhen den Integrations- und Wartungsaufwand, wodurch eine alternative Lösung erforderlich wird.

Zweite Platinenversion (2. Mainboard)

- **Geringere Kosten:** Wegfall teurer Transceiver und Multiplexer, kompaktere Platine.
- **Kompaktes Design:** Weniger Bauteile, klar strukturierte Schnittstellen (USB-C).
- **Übernahme der Key-Learnings:** Die bei der ersten Platine gesammelten Erfahrungen flossen in eine effizientere Spannungsversorgung, eine kompaktere Lüfteransteuerung und eine gezielte Wahl besserer Bauformen für Chips ein.

Fazit

Letztlich wurden beide Platinenversionen umgesetzt:

- **Version A (I²C)** diente als erster Prototyp, um die Komponenten schnell in Betrieb zu nehmen und erste Testläufe zu fahren.
- **Version B (USB-C)** wurde danach entwickelt, da sich im praktischen Einsatz zeigte, dass eine kompaktere, kostengünstigere und einfacher zu verdrahtende Lösung vorteilhafter ist.

Die abschließende Empfehlung für den produktiven Einsatz lautet daher, die **zweite Platinenversion (Version B)** mit separatem Mikrocontroller zu nutzen, da sie in puncto **Kompaktheit**, **Kosten** und **Einfachheit der Montage** langfristig überlegen ist.

4.5 Lösungsansätze und Konzepte zur Platinenbauform

In diesem Kapitel werden verschiedene Möglichkeiten zur Umsetzung der Platinenbauform vorgestellt und bewertet.

4.5.1 Konzept 1 zur Platinenbauform: Gestapelte Platine

Beschreibung

Bei diesem Konzept werden zwei oder mehr Platinen übereinander gestapelt und über Steckverbinder miteinander verbunden. Dies ermöglicht eine kompakte Bauweise, da weniger Grundfläche benötigt wird. Die Hauptplatine enthält die Steuerlogik, während zusätzliche Module auf separaten Platinen integriert werden können.

Kosten

Die Kosten für dieses Konzept belaufen sich auf etwa **6.00 €** und setzen sich zusammen aus:

- **Steckverbinder für Platinenstapelung:** ca. 4.00 €
- **Abstandhalter und Befestigungsmaterial:** ca. 2.00 €

Hinweis

Die angegebenen Kosten beziehen sich ausschließlich auf die spezifischen Komponenten für die Bauform. Grundkosten für Bauteile wie Mikrocontroller oder Sensoren sind hier nicht enthalten.

Vorteile

- **Platzsparende Bauweise:** Da mehrere Schichten übereinander gestapelt werden, benötigt das System weniger Grundfläche.
- **Gute Modularität:** Erweiterungen oder Modifikationen sind einfach realisierbar, indem weitere Platinen aufgesteckt werden.
- **Bessere Fehlerbehebung:** Defekte Module können separat ausgetauscht werden, ohne das gesamte System zu ersetzen.

Nachteile

- **Erhöhter Aufbauaufwand:** Die Montage und Demontage erfordert mehr Schritte als bei einer einfachen Platine.
- **Mechanische Stabilität:** Die Steckverbinder müssen robust genug sein, um eine sichere Verbindung über längere Zeit zu gewährleisten.
- **Mögliche Signalstörungen:** Durch die vertikale Struktur können Signallaufzeiten oder Interferenzen eine größere Rolle spielen.

4.5.2 Konzept 2 zur Platinenbauform: Einfache Platine

Beschreibung

Hier wird eine einzige, größere Platine genutzt, auf der alle notwendigen Komponenten integriert sind. Es gibt keine gestapelten Module oder separaten Erweiterungsplatinen.

Kosten

Bei dieser Platinenform sind keine zusätzlichen Kosten für Steckverbinder oder Abstandhalter erforderlich.

Vorteile

- **Einfache Montage:** Keine zusätzlichen Steckverbinder oder modulare Verbindungen nötig.
- **Geringere Signalstörungen:** Da alle Komponenten auf einer Platine sind, können potenzielle Störungen minimiert werden.
- **Geringe Komplexität:** Weniger Bauteile und Verbindungen bedeuten weniger Fehlerquellen und einfache Fertigung.

Nachteile

- **Größerer Platzbedarf:** Da keine vertikale Anordnung genutzt wird, benötigt die Platine mehr Grundfläche.
- **Geringere Modularität:** Erweiterungen oder Änderungen am System erfordern eine Neugestaltung der gesamten Platine.
- **Aufwändige Fehlersuche:** Da alle Komponenten auf einer großen Platine sind, kann die Diagnose und Reparatur komplexer werden.

4.5.3 Entscheidungsmatrix für die Platinenbauform

Um die beiden möglichen Platinenbauformen objektiv zu bewerten, wird eine Entscheidungsmatrix herangezogen. Dabei werden relevante Kriterien definiert, gewichtet und bewertet.

Kriterien für die Auswahl der Bauform

Die folgenden Kriterien wurden zur Bewertung der beiden Platinenbauformen herangezogen:

- **Kosten:** Die Herstellungskosten sollten möglichst gering gehalten werden, wobei ein gutes Preis-Leistungs-Verhältnis angestrebt wird.
- **Gewicht:** Eine leichtere Bauweise ist vorteilhaft für das Gesamtsystem.
- **Platzbedarf:** Eine kompakte Bauweise ermöglicht eine bessere Integration in das bestehende System.
- **Einfachheit:** Eine einfache Konstruktion erleichtert die Fertigung und Montage.
- **Design:** Das Layout der Platine soll funktional und ästhetisch ansprechend sein.
- **Modularität:** Eine modulare Bauweise ermöglicht eine bessere Skalierbarkeit und Flexibilität.
- **Fehlerbehebung:** Eine einfache Fehlersuche und Wartung erleichtert langfristig den Betrieb.

Entscheidungsmatrix (Vergleich der Platinenbauformen)

Tabelle 11: Bewertung der Bauformen nach verschiedenen Kriterien

Kriterium	Gewichtung	Gestapelt	Gewichtet	Einfach	Gewichtet
Kosten	0.30	3	0.90	5	1.50
Gewicht	0.75	5	3.75	4	3.00
Platzbedarf	0.80	4	3.20	2	1.60
Einfachheit	0.70	3	2.10	5	3.50
Design	0.40	4	1.60	4	1.60
Modularität	0.75	5	3.75	1	0.75
Fehlerbehebung	1.00	5	5.00	3	3.00
Gesamtbewertung		29	20.30	24	14.95

Entscheidung und Begründung

Die Entscheidungsmatrix zeigt, dass die **gestapelte Platine** eine deutlich höhere Gesamtpunktzahl erreicht. Besonders in den Kategorien **Platzbedarf, Modularität und Fehlerbehebung** bietet diese Variante erhebliche Vorteile. Daher wird die **gestapelte Platine** umgesetzt, da sie langfristig eine bessere Skalierbarkeit, Wartungsfreundlichkeit und Systemintegration bietet.

4.6 Modulare Aufteilung der Elektronik

Das Gesamtdesign der Platine basiert auf einer modularen Struktur, die bereits in 4.5.1 *Konzept 1 zur Platinenbauform: Gestapelte Platine*, S.62 beschrieben wurde. Die Aufteilung erfolgt in drei Hauptmodule: das Leistungs-Modul, das Lüfter-Modul und das Motoren-Modul. Diese sind mechanisch gestapelt und elektrisch miteinander verbunden. Die Einteilung der Module wird in Abbildung 74 dargestellt.

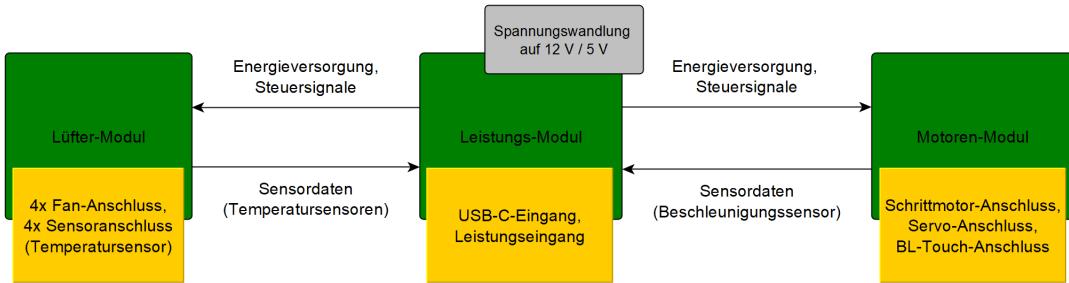


Abb. 74: Funktionale Modulaufteilung der Platine

4.6.1 Mechanische Stabilität

Die Montage der Module erfolgt durch präzise platzierte Befestigungslöcher, die auf die Abstandshalter 971070154 (DigiKey, 2025e) und 971110154 (DigiKey, 2025f) abgestimmt sind. Dies gewährleistet eine robuste und sichere Konstruktion des gesamten Systems.

4.6.2 Leistungs-Modul

Das Leistungs-Modul stellt die zentrale Steuereinheit dar. Es enthält den Mikrocontroller ATmega328P sowie die Spannungswandler zur Bereitstellung der notwendigen Versorgungsspannungen (12 V und 5 V). Zudem werden hier die primären Verbindungsschnittstellen realisiert:

- Anschluss des 3D-Drucker-Netzteils zur Energieversorgung
- USB-Datenverbindung zum Raspberry Pi

4.6.3 Lüfter-Modul

Das Lüfter-Modul ist für die Ansteuerung der Lüfter verantwortlich und enthält:

- MOSFETs zur stufenlosen Steuerung der Lüfter
- Motortreiber zur Lüftersteuerung
- Spannungsteiler zur Auswertung der Temperatursensoren

Die Verbindung zum Leistungs-Modul wird über die in Abbildung 75 (Verbindung zwischen Leistungs- und Lüfter-Modul), S.64 dargestellten Board-zu-Board-Konnektoren hergestellt, welche genauer in der Stückliste Tabelle 23 (Stückliste der Elektronikkomponenten), S.206 untersucht werden können.

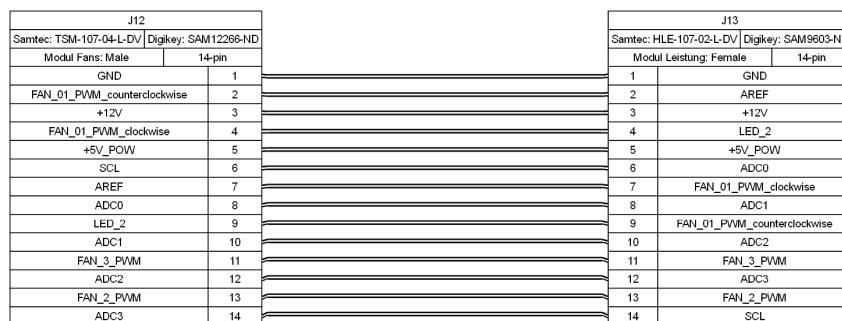


Abb. 75: Verbindung zwischen Leistungs- und Lüfter-Modul

4.6.4 Motoren-Modul

Das Motoren-Modul umfasst alle Steuerungselemente für die Bewegungskomponenten:

- Schrittmotortreiber
- Schutzkondensatoren für Servo-Motoren
- BL-Touch-Sensor für die automatische Bettneivellierung
- Beschleunigungssensor zur Einbindung der Input-Shaping Funktion in Klipper

Die Verbindung zwischen Leistungs- und Motoren-Modul erfolgt über die in Abbildung 76 gezeigten Board-zu-Board-Konnektoren, welche ebenfalls in der Stückliste Tabelle 23 (*Stückliste der Elektronikkomponenten*), S.206 aufgeführt sind.

J24		J15	
Samtec: HW-07-08-L-D-250-SM Mouser: 200-HW0708LD250SM		Samtec: HLE-107-02-L-DV Digikey: SAM9603-ND	
Modul Leistung: Male	14-pin	Modul Motoren: Female	14-pin
GND	1	1	GND
SDA	2	2	SDA
+12V	3	3	+12V
RXD	4	4	RXD
+5V_POW	5	5	+5V_POW
TXD	6	6	TXD
STEP	7	7	STEP
DIR	8	8	DIR
MOSI	9	9	MOSI
BLT_Servo	10	10	BLT_Servo
LED_3	11	11	LED_3
BLT_Switch	12	12	BLT_Switch
NC	13	13	NC
SCL	14	14	SCL

Abb. 76: Verbindung zwischen Leistungs- und Motoren-Modul

4.7 Auswahl der Komponenten

In diesem Abschnitt werden die wichtigsten ausgewählten elektrischen Komponenten beschrieben und dokumentiert. Bei der Auswahl wurde darauf geachtet, dass die Kosten der Bauteile so niedrig wie möglich gehalten werden. Zunächst wurden die platzsparendsten, anschließend die kostengünstigsten Bauteile gesucht, die die Anforderungen erfüllen. Die vollständige Stückliste ist im Anhang (9.3 *Stückliste*, S.206) zu finden.

4.7.1 Spannungsversorgung und Strombedarf

Die Spannungsversorgung des Systems erfolgt über das Netzteil des 3D-Druckers, wie bereits in Abbildung 73 (*Systemübersicht der Elektronik und deren Verbindungen*), S.57 dargestellt. Typischerweise erzeugen 3D-Drucker Netzteile **12 V oder 24 V Gleichspannung** (Filament2Print, 2025). Die am häufigsten verwendeten Netzteile lassen sich folgendermaßen kategorisieren:

- **12 V-Netzteile**, meist zwischen 120 W und 240 W. (All3DP, 2018)
Diese sind oft in Einsteigermodellen oder älteren 3D-Druckern zu finden.
- **24 V-Netzteile**, meist zwischen 240 W und 350 W. (All3DP, 2018)
Moderne 3D-Drucker verwenden vermehrt diese Netzteile, da sie effizienter sind und kürzere Aufheizzeiten ermöglichen.



Abb. 77: Netzteil eines 3D-Druckers (Harald, 2025)

Die minimal bereitgestellten Stromwerte der Netzteile sowie der verbleibende Strom für externe Komponenten lassen sich aus der Leistung berechnen. Dabei wird angenommen, dass der 3D-Drucker etwa 80 % der verfügbaren Leistung nutzt, während 20 % für externe Komponenten zur Verfügung stehen.

- **120 W/12 V-Netzteil:**

$$I_{\text{gesamt}} = \frac{P}{U} = \frac{120 \text{ W}}{12 \text{ V}} = 10 \text{ A} \quad (1)$$

$$I_{\text{externe}} = 0,2 \cdot I_{\text{gesamt}} = 0,2 \cdot 10 \text{ A} = 2 \text{ A} \quad (2)$$

- **240 W/24 V-Netzteil:**

$$I_{\text{gesamt}} = \frac{P}{U} = \frac{240 \text{ W}}{24 \text{ V}} = 10 \text{ A} \quad (3)$$

$$I_{\text{externe}} = 0,2 \cdot I_{\text{gesamt}} = 0,2 \cdot 10 \text{ A} = 2 \text{ A} \quad (4)$$

Die Betriebsspannung der Elektronik wird über Step-Down-Wandler auf die benötigten Werte reduziert. In diesem Fall sind dies zwei Hauptspannungen: **5 V** und **12 V**. Die 5 V-Schiene versorgt unter anderem den Mikrocontroller und Sensoren, während 12 V für bestimmte Aktoren benötigt werden. Eine detaillierte Übersicht über die Komponenten, deren Stromaufnahme und Spannungsniveau zeigt die Abbildung 78:

Bezeichnung	Seriennummer	Stromverbr. in A	Spannung	Stückzahl
Microchip	ATMEGA328P-AU	0.02	5	1
USB-Serial Wandler	FT232RNL-TUBE	0.1	5	1
Schrittmotor-Treiber	DRV8825	1.5	12	1
Servo	MG90S	0.5	5	1
Nivellierungssensor	BLTouch Nivellierungssensor	0.5	5	1
Lüfter	CFM-2507CF-1140-313	0.17	12	2
Lüfter	CFM-2507CF-0140-313	0.35	5	2
12-40V zu 40V Buck	TPS56837HRPAR	0.05	5	1
Fan-Controller mit Richtungswechsel	TPM16050-S6TR	0.02	5	1

Strom in A bei 12 V
1.84
Strom in A bei 5 V
1.89

Abb. 78: Übersicht der Komponenten mit Strom- und Spannungsbedarf

Basierend auf den gegebenen Stromwerten für die 12V- und 5V-Versorgung wird der Gesamtstrombedarf ermittelt. Dabei wird der Strom für die 12V-Versorgung unter Berücksichtigung der Leistungsaufnahme der 5V-Komponenten berechnet. Anschließend wird der Strom für die 24V-Versorgung bestimmt.

$$P_5 = I_5 \cdot V_5 = 1.89 \text{ A} \cdot 5 \text{ V} = 9.45 \text{ W} \quad (5)$$

$$I_{12ges} = I_{12} + \frac{P_5}{V_{12}} = 1.84 \text{ A} + \frac{9.45 \text{ W}}{12 \text{ V}} = 2.63 \text{ A} \quad (6)$$

$$P_{12} = I_{12ges} \cdot V_{12} = 2.63 \text{ A} \cdot 12 \text{ V} = 31.53 \text{ W} \quad (7)$$

$$I_{24} = \frac{P_{12}}{V_{24}} = \frac{31.53 \text{ W}}{24 \text{ V}} = 1.31 \text{ A} \quad (8)$$

Daraus ergibt sich ein Gesamtstrombedarf von:

- **12 V-Versorgung:** $I_{12ges} = 2.63 \text{ A}$

- **24 V-Versorgung:** $I_{24} = 1.31 \text{ A}$

Die berechneten Stromwerte zeigen, dass der benötigte Strom von 1,31 A bei 24 V problemlos von der verfügbaren Leistung des Netzteils für externe Komponenten (2 A, wie in Gleichung 2 und 4 berechnet) bereitgestellt werden kann. Dies gilt sowohl für 12V- als auch für 24 V-Netzteile, was die Kompatibilität mit verschiedenen 3D-Druckernmodellen sicherstellt.

4.7.2 Übersicht der Teilbereiche

Elektronische Steuerung

Das System basiert auf einem ATmega328-Mikrocontroller zur Echtzeitsteuerung der peripheren Komponenten. Für rechenintensive Aufgaben kommt ein Raspberry Pi zum Einsatz.

Kommunikationsschnittstellen

Die Hauptkommunikation erfolgt über:

- UART für die Kommunikation zwischen Raspberry Pi und ATmega328
- Digitale Ausgänge für den Schrittmotor
- PWM zur Steuerung von Servos und Lüftern
- Digitale Eingänge für den Endschalter des BL-Touch
- ADC zur Verarbeitung analoger Sensordaten
- I²C für die Kommunikation mit einem Beschleunigungssensor

Sensorik und Aktoren

Zur Überwachung und Regelung des Systems werden verschiedene Sensoren und Aktoren eingesetzt:

- Temperaturüberwachung mit Temperatursensoren
- BL-Touch zur Druckbettkalibrierung (PWM + digitales Signal)
- Beschleunigungssensor zur Vibrationsanalyse und Input-Shaping
- 4 Lüfter (5 V/12 V, PWM-gesteuert)
- Schrittmotor für Faserextrusion (12 V)
- Servomotor für Faserschnitt (5 V)

Input-Shaping

Input Shaping ist eine Technik zur Vibrationskompensation in 3D-Druckern, die unerwünschte Schwingungen reduziert und so die Druckqualität verbessert. In Klipper ist Input Shaping einfach integrierbar, da es direkt in der Firmware unterstützt wird und über eine einfache Konfiguration mit einem Beschleunigungssensor optimal eingestellt werden kann. („Wikipedia“, 2024)

Mechanische Integration

Die Platine wird robust in das Gesamtsystem integriert, wobei folgende Aspekte berücksichtigt werden:

- Leistungs-Buchse für Spannungsversorgung
- JST-Buchsen für Lüfter und Motoren
- Thermische Maßnahmen für Leistungskomponenten
- Befestigungsbohrungen zur sicheren Montage

Schutzschaltungen und Maßnahmen

Um die Betriebssicherheit zu gewährleisten und die elektronischen Komponenten zu schützen, werden folgende Schutzmechanismen implementiert:

- TVS-Dioden zur Absicherung gegen Überspannungen
- Pufferkondensatoren nahe den Verbrauchern zur Stabilisierung der Spannungsversorgung
- Verpolungsschutzschaltung für die Hauptversorgung

4.7.3 Einplatinencomputer – Raspberry Pi 3 Model B+

Beschreibung Funktion

Der **Raspberry Pi 3 Model B+** fungiert als Hauptsteuerungseinheit des 3D-Druckers, auf dem die Firmware **Klipper** läuft. Er verarbeitet die Druckaufträge, kommuniziert mit dem Drucker-Mainboard über USB und kommuniziert mit einer eigenentwickelten Druckkopf-Platine. Dabei übernimmt er die Aufgaben der Berechnung der Druckbewegungen sowie die Echtzeitkommunikation zwischen den Steuerkomponenten.



Abb. 79: Raspberry Pi 3 Model B+ (nach Digikey, 2025)

Anforderungen

- Kompatibilität mit der **Klipper**-Firmware
- Ausreichend Rechenleistung für die Steuerung des Druckers
- Weit verbreitet in der 3D-Druck-Community für gute Software-Unterstützung
- Stabile und zuverlässige USB-Schnittstellen für Kommunikation mit dem Mainboard und der Druckkopf-Platine

Technische Details

Tabelle 12: Technische Details des Raspberry Pi 3 Model B+ und der Alternative Raspberry Pi 4 Model B (aus Digikey, 2025)

Parameter	Raspberry Pi 3 Model B+	Alternative: Raspberry Pi 4 Model B
Prozessor	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-Bit SoC @ 1,4GHz	Broadcom BCM2711, Cortex-A72 (ARMv8) 64-Bit SoC @ 1,5GHz
Arbeitsspeicher	1GB LPDDR2 SDRAM	2GB, 4GB oder 8GB LPDDR4-3200 SDRAM
USB-Ports	4 × USB 2.0	2 × USB 3.0, 2 × USB 2.0
Videoausgang	HDMI	2 × Micro-HDMI (4K@60Hz)
Abmessungen	85,6 mm × 56,5 mm	85,6 mm × 56,5 mm
Preis/Stk.	33,37 €	ca. 60,00 € (je nach RAM-Variante)
Bezugsquelle	Digi-Key	Offizielle Raspberry Pi Händler (z. B. BerryBase, Pi-Shop)

Fazit

Der **Raspberry Pi 3 Model B+** ist eine bewährte und gut unterstützte Plattform für den Einsatz mit Klipper. Die Alternative **Raspberry Pi 4 Model B** bietet mehr Rechenleistung und bessere Schnittstellen, ist jedoch teurer.

4.7.4 Mikrocontroller – ATMEGA328P-AU

Beschreibung Funktion

Der **ATMEGA328P-AU** dient als zentrales Steuerungselement für die Druckkopf-Platine. Er übernimmt die Ansteuerung von Komponenten wie Lüfern, dem BL Touch, einem Servo, einem Schrittmotor und Temperatursensoren. Auf den **ATMEGA328P-AU** wird die **Klipper**-Firmware aufgespielt, die eine effiziente Steuerung der Hardware-Komponenten ermöglicht. In der **Klipper**-Konfiguration wird der **ATMEGA328P-AU** als zweites Mainboard definiert. Die Kommunikation zwischen dem **ATMEGA328P-AU** und dem **Raspberry Pi** erfolgt über die serielle Schnittstelle, wodurch eine zuverlässige und schnelle Datenübertragung gewährleistet wird. Aufgrund der weit verbreiteten AVR-Architektur und der umfassenden Unterstützung durch die Open-Source-Community bietet er eine zuverlässige Entwicklungs-Umgebung.

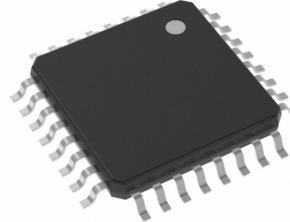


Abb. 80: ATMEGA328P-AU Mikrocontroller aus (nach DigiKey, 2025h)

Anforderungen

- 8-Bit AVR-Architektur für effiziente Verarbeitung
- Mindestens 16 MHz Taktfrequenz für eine stabile Echtzeitsteuerung
- Ausreichend digitale und analoge I/O-Pins für die erforderlichen Schnittstellen
- Unterstützung von SPI und I2C zur Kommunikation mit weiteren Systemkomponenten

Technische Details

Tabelle 13: Technische Details des ATMEGA328P-AU, ATmega2560 und STM32 (Cortex-M3) (aus DigiKey, 2025h)

Parameter	ATMEGA328P-AU	ATmega2560	STM32 (Cortex-M3)
Architektur	8-Bit AVR	8-Bit AVR	32-Bit ARM Cortex-M3
Taktfrequenz	20 MHz	16 MHz	72 MHz
Flash-Speicher	32 KB	256 KB	64 KB - 1 MB
Digitale I/O-Pins	23	54	50+
PWM-Pins	6	15	10+
Analoge Eingänge	6	16	10+
I2C/SPI Unterstützung	Ja	Ja	Ja
Kompatibel mit Klipper	Ja	Ja	Ja
Preis/Stk.	2,55 €	6,80 €	5.42 €
Bezugsquelle	DigiKey	DigiKey	DigiKey

Fazit

Der **ATMEGA328P-AU** wurde aufgrund seiner etablierten AVR-Architektur, seiner niedrigen Kosten sowie der umfangreichen Unterstützung innerhalb der Open-Source-Community für das Projekt ausgewählt. Als mögliche Alternative bietet der **ATmega2560** eine größere Anzahl an I/O-Pins und zusätzlichen Speicher, benötigt jedoch mehr Platz auf der Platine. Der **STM32 (Cortex-M3)** überzeugt durch eine höhere Rechenleistung und **Klipper**-Kompatibilität, erfordert jedoch eine aufwendigere Implementierung.

4.7.5 Schrittmotor – 14HM08-0504S

Beschreibung Funktion

Der **14HM08-0504S** ist ein bipolarer Schrittmotor, der für den Faservorschub verwendet wird. Aufgrund seiner kompakten Bauform ist er besonders gut für das begrenzte Platzangebot geeignet. Zudem ist er kompatibel mit der Klipper-Firmware und ermöglicht eine präzise, kontinuierliche Steuerung.

Anforderungen

- Bipolarer Schrittmotor für präzise Positionierung
- Drehmoment: 5 Ncm
- Baugröße: 35x35x20 mm
- Betriebsspannung: 5 V
- Maximaler Strom: 0.5 A

Technische Details

Tabelle 14: Technische Details des Schrittmotors 14HM08-0504S und der Alternative 17HS10-0704S (aus Stepperonline, 2024)

Parameter	14HM08-0504S	Alternative: 17HS10-0704S
Motortyp	Bipolar	Bipolar
Drehmoment	5 Ncm	10 Ncm
Baugröße	35x35x20 mm	42x42x23 mm
Schrittwinkel	0,9 °	1,8 °
Betriebsspannung	5 V	5 V
Maximaler Strom	0.5 A	0.7 A
Gewicht	90 g	120 g
Preis/Stk.	9,91 €	12,50 €
Bezugsquelle	StepperOnline	StepperOnline

Fazit

Der **14HM08-0504S** wurde aufgrund seiner kompakten Bauweise und der ausreichenden Leistung für den Extruder ausgewählt. Er bietet eine präzise Steuerung bei gleichzeitig geringer Größe und geringem Gewicht, was für den Einsatz in einem platzkritischen System entscheidend ist.



Abb. 81: Schrittmotor 14HM08-0504S aus Stepperonline, 2024

Schrittmotor-Treiber – DRV8825

Beschreibung Funktion

Der **DRV8825** ist ein Schrittmotor-Treiber, der zur präzisen Steuerung bipolarer Schrittmotoren dient. Er unterstützt Mikroschritt-Auflösungen bis zu 1/32-Schritten und verfügt über einen einstellbaren Stromregler.

Anforderungen

- Unterstützung für bipolare Schrittmotoren
- Mikroschritt-Auflösung: bis zu 1/32-Schritte
- Maximaler Ausgangsstrom: 2,5 A
- Versorgungsspannung: 8,2 V bis 45 V
- Integrierter Überstrom- und Überhitzungsschutz

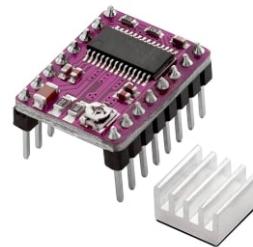


Abb. 82: DRV8825 Schrittmotor-Treiber aus (nach AZ-Delivery, 2025a)

Technische Details

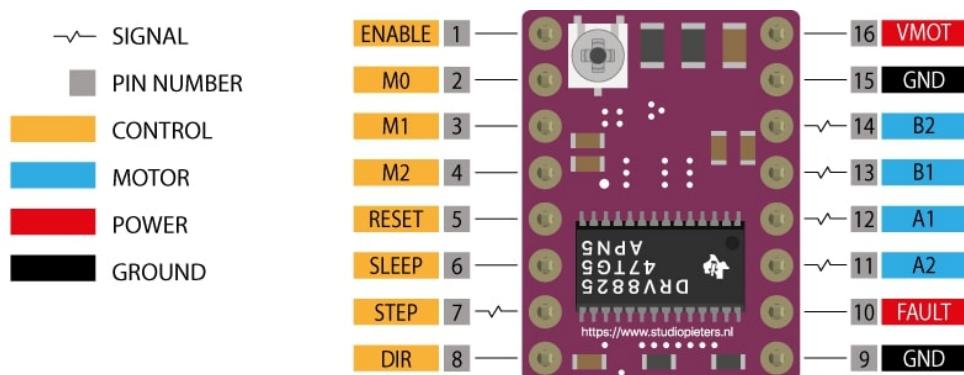
Tabelle 15: Technische Details des Schrittmotor-Treibers DRV8825 und der Alternative TMC2209 (aus AZ-Delivery, 2025a)

Parameter	DRV8825	Alternative: TMC2209
Versorgungsspannung	8,2 V – 45 V	4,75 V – 29 V
Max. Ausgangsstrom	2,5 A	2 A RMS (2,8 A Spitze)
Mikroschritt-Auflösung	Bis zu 1/32-Schritte	Bis zu 1/256-Schritte
Schutzfunktionen	Überstrom-, Kurzschluss-, Unterspannungssperre und Übertemperaturschutz	Überstrom-, Kurzschluss-, Unterspannungssperre und Übertemperaturschutz
Besondere Merkmale	Einfache STEP/DIR-Schnittstelle	UART-Schnittstelle für Strommessung
Preis/Stk.	6,99 €	6,99 €
Bezugsquelle	AZ-Delivery	3D-Jake

Fazit

Der **DRV8825** wurde aufgrund seiner einfachen Implementierung und seiner Kompatibilität mit gängigen Steuerungssystemen ausgewählt. Wenn eine höhere Mikroschritt-Auflösung oder eine UART-Schnittstelle für die Strommessung erforderlich ist, kann der **TMC2209** als Alternative verwendet werden.

Pinout des DRV8825



StudioPieters®
<https://www.studiopieters.nl>

Abb. 83: Pinout des Schrittmotor-Treibers **DRV8825** (nach Pieters, 2022)

4.7.6 Servo – MG90S

Beschreibung Funktion:

Der **MG90S** ist ein digitaler Mikroservo, der für präzise Steuerungsaufgaben in kompakten Anwendungen entwickelt wurde. Aufgrund seiner robusten Konstruktion und schnellen Reaktionszeit eignet er sich besonders für Projekte mit begrenztem Bauraum und Anforderungen an hohe Zuverlässigkeit.

Anforderungen:

- Betriebsspannung: 5 V
- Drehmoment: mindestens 2.0 kg/cm
- Abmessungen: maximal 23 mm x 12 mm x 29 mm
- Gewicht: maximal 14 g
- Rotationswinkel: 180 °



Abb. 84: Servo MG90S (nach AZ-Delivery, 2025c)

Technische Details

Tabelle 16: Technische Details des Servos MG90S und der Alternative Lynxmotion Micro 9G (aus AZ-Delivery, 2025c)

Parameter	MG90S	Alternative: Lynxmotion Micro 9G
Motortyp	Digitaler Servo	Digitaler Servo
Drehmoment	2.0 kg/cm (4.8 V)	1.8 kg/cm (4.8 V)
Abmessungen	22.8 mm x 12.2 mm x 28.5 mm	22.8 mm x 12.2 mm x 28.5 mm
Rotationswinkel	180 °	180 °
Betriebsspannung	4.8 V bis 6 V	4.8 V bis 6 V
Betriebsstrom	120 mA bis 250 mA	120 mA bis 250 mA
Gewicht	13.4 g	13.4 g
Preis/Stk.	4.99 €	10.14 €
Bezugsquelle	Smart Prototyping	RobotShop

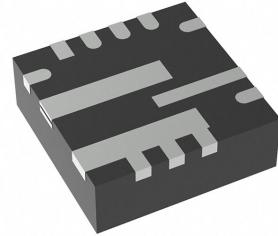
Fazit

Der **MG90S** wurde aufgrund seiner kompakten Abmessungen ausgewählt. Er ist ideal für Anwendungen, die eine zuverlässige und präzise Bewegung erfordern, insbesondere in beengten Platzverhältnissen.

4.7.7 Spannungswandler 24 V auf 12 V – TPS56837HRPAR

Beschreibung Funktion

Der **TPS56837HRPAR** regelt die Eingangsspannung, die vom 3D-Drucker-Netzteil bereitgestellt wird und entweder 24V oder 12V betragen kann, auf eine fixe Ausgangsspannung von 12V.



Anforderungen

- Eingangsspannung: 12 V bis 24 V
- Ausgangsspannung: 12V
- Maximaler Ausgangsstrom: 4 A
- Schaltfrequenz: 500 kHz
- Effizienz: > 90 %

Abb. 85: TPS56837HRPAR Spannungswandler aus (nach DigiKey, 2025ad)

Technische Details

Tabelle 17: Technische Details der Spannungswandler TPS56837HRPAR und der Alternative LM2679SX-12/NOPB (aus DigiKey, 2025ad)

Parameter	TPS56837HRPAR	LM2679SX-12/NOPB
Eingangsspannung	12-40 V	12-40 V
Ausgangsspannung	0,6 V – 40 V	12 V fix
Max. Ausgangsstrom	8 A	5 A
Schaltfrequenz	500 kHz	260 kHz
Effizienz	95 %	92 %
Bauform	VQFN-22	TO-263
Preis/Stk.	2,34 €	5,79 €
Bezugsquelle	DigiKey	DigiKey

Fazit

Der **TPS56837HRPAR** wurde als leistungsstarke Alternative zum **LM2679SX-12/NOPB** ausgewählt. Durch seine hohe Schaltfrequenz, den größeren Ausgangsstrom und die kompakte Bauform ist er besonders gut für platzkritische Anwendungen mit hohen Leistungsanforderungen geeignet.

Anwendungsschaltung

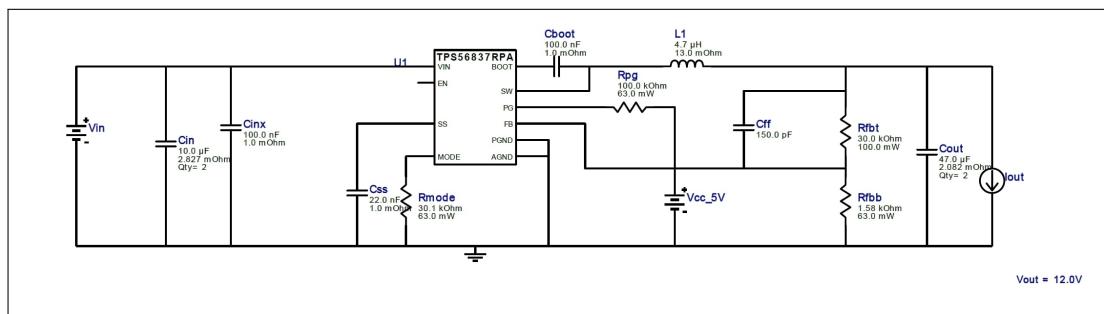


Abb. 86: Anwendungsschaltung für den Spannungswandler **TPS56837HRPAR** aus 9.7 Layout Empfehlungen aus dem Datenblatt des **TPS56387**, S.216

4.7.8 Spannungswandler 12 V auf 5 V – TPS565208DDCR

Beschreibung Funktion

Der **TPS565208DDCR** regelt die 12V-Ausgangsspannung, die vom **TPS56837HRPAR** bereitgestellt wird auf eine fixe Ausgangsspannung von 5V.

Anforderungen

- Eingangsspannung: 12 V
- Ausgangsspannung: 5 V
- Maximaler Ausgangsstrom: 2 A
- Schaltfrequenz: 500 kHz
- Effizienz: > 90 %

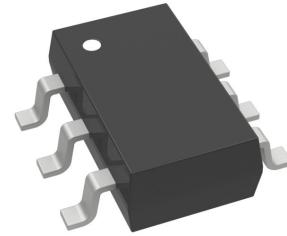


Abb. 87: TPS565208DDCR Spannungswandler aus (nach DigiKey, 2025ac)

Technische Details

Tabelle 18: Technische Details der Spannungswandler TPS565208DDCR und der Alternative LM22676MRX-5.0/NOPB (aus DigiKey, 2025ac)

Parameter	TPS565208DDCR	LM22676MRX-5.0/NOPB
Eingangsspannung	4,5-17 V	4,5-42 V
Ausgangsspannung	5 V fix	5 V fix
Max. Ausgangsstrom	5 A	3 A
Schaltfrequenz	500 kHz	500 kHz
Effizienz	90 %	85 %
Bauform	SOT-23-6	SOIC-8
Preis/Stk.	1,17 €	4,14 €
Bezugsquelle	DigiKey	DigiKey

Fazit

Der **TPS565208DDCR** wurde als leistungsstarke Alternative zum **LM22676MRX-5.0/NOPB** ausgewählt. Durch seine höhere Strombelastbarkeit, kompakte Bauform und bessere Effizienz eignet er sich besonders für platzkritische Anwendungen mit hohen Leistungsanforderungen.

Anwendungsschaltung

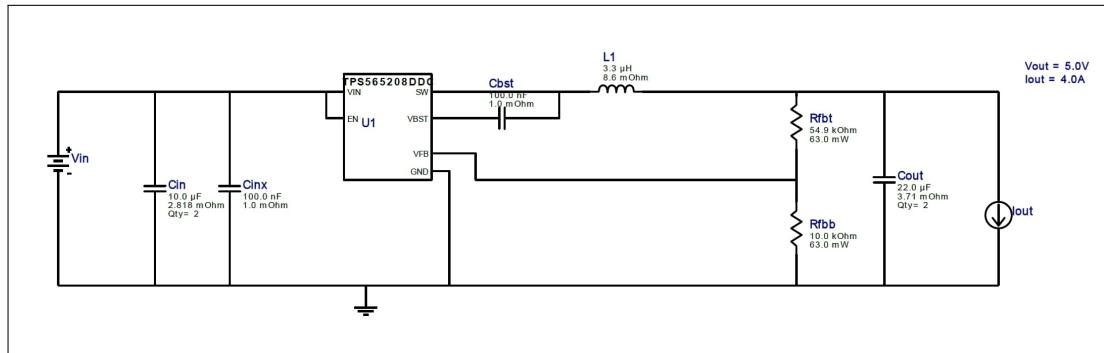


Abb. 88: Anwendungsschaltung für den Spannungswandler **TPS565208DDCR** aus 9.8 *Layout Empfehlungen aus dem Datenblatt des TPS565208*, S.217

4.8 Schaltungsdesign

Im Folgenden Abschnitt werden die wichtigsten Teile des entworfenen Schaltplans, sowie die wichtigsten Verkabelungen erläutert. Die Schaltung ist modular aufgebaut und in verschiedene Funktionsblöcke unterteilt.

4.8.1 Leistungseingang und Schutzschaltung

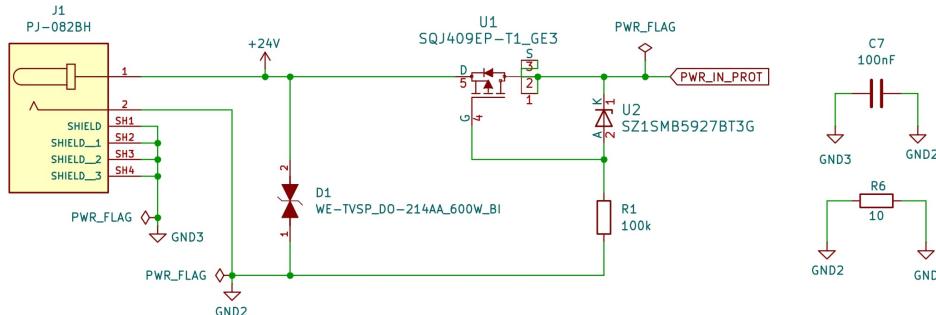


Abb. 89: Schaltplan des Leistungseingangs und der Schutzschaltung

Der Leistungseingang des Systems erfolgt über eine DC-Buchse (J1, Typ PJ-082BH), die das Netzteil des 3D-Druckers mit der Schaltung verbindet. Die Versorgungsspannung beträgt 24 V DC und wird über ein geschirmtes, flexibles Steuerkabel (Igus CF881-15-02) vom Netzteil (J2) zur Buchse J1 geführt. Die folgende Abbildung zeigt den schematischen Aufbau der Verbindung zwischen Netzteil und Platine:

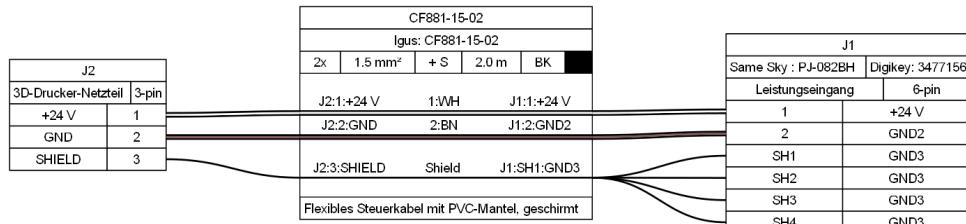


Abb. 90: Verbindung zwischen 3D-Drucker-Netzteil und Platine

Das Kabel verfügt über zwei Adern mit je $1,5 \text{ mm}^2$ Querschnitt für +24 V und GND sowie eine zusätzliche Abschirmung, die mit den Massepunkten verbunden ist. Die Masseleitungen sind über verschiedene Pins der Buchse (J1) mit der Platine verbunden, um eine stabile und störungsfreie Stromversorgung sicherzustellen.

Schutzmechanismen

Die Versorgungsspannung beträgt 24 V DC, welche über verschiedene Schutz- und Stabilisierungsmaßnahmen weiterverarbeitet wird.

Überspannungsschutz

Eine TVS-Diode (D1, WE-TVSP_D0-214AA_600W_BI) ist in Sperrrichtung gegen Masse geschaltet. Sie schützt die nachfolgenden Schaltungsteile vor Spannungsspitzen, indem sie bei einer Überschreitung der spezifizierten Durchbruchspannung leitend wird und die überschüssige Energie ableitet.

Verpolungsschutz

Ein P-Kanal-MOSFET (U1, SQJ409EP-T1_GE3) ist als Verpolungsschutz konfiguriert. Der Source-Anschluss ist direkt mit der Eingangsspannung verbunden, während Drain die geregelte Ausgangsspannung zur weiteren Verwendung liefert. Falls eine falsch gepolte Spannung angelegt wird, bleibt der MOSFET gesperrt und schützt die Schaltung. Die Funktion des Verpolungsschutzes wird in 4.9.1 *Simulation Verpolungsschutz*, S.81 simuliert.

Gate-Source-Spannungsstabilisierung

Eine Zener-Diode (U2, SZ1SMB5927BT3G) ist zwischen Gate und Source des MOSFETs angebracht. Sie stellt sicher, dass die Gate-Source-Spannung (V_{GS}) nicht über den sicheren Bereich hinaus ansteigt, wodurch eine zuverlässige Schaltfunktion gewährleistet wird.

Spannungsstabilisierung und Masseverbindungen

- Ein Pull-Down-Widerstand (R1, 100 k Ω) entlädt das Gate des MOSFETs, um sicherzustellen, dass der MOSFET bei Fehlen einer Eingangsspannung sicher sperrt.
- Es kann ein Entkopplungskondensator (C7, 100 nF) eine kapazitive Verbindung zwischen GND3 (Shield-Masse) und GND2 herstellen. Dadurch wird für hochfrequente Störsignale ein niedrigimpedanter Rückstrompfad geschaffen, wodurch Masseschleifen vermieden und EMV-Störungen reduziert werden. Dieser Kondensator kann bei Bedarf verbaut werden.
- Der Widerstand R6 (10 Ω) verbindet unterschiedliche Massepotenziale (GND2 und GND (System-Masse)) und verhindert unerwünschte Masseschleifen.

4.8.2 USB-Datenverbindung und Schnittstellenwandler

Die Datenverbindung zwischen dem Raspberry Pi und der Platine erfolgt über ein USB-C-Kabel. Die Signale für die Datenübertragung werden vom Raspberry Pi bereitgestellt und an den USB-C-Eingang der Platine weitergeleitet. Abbildung 91 zeigt die schematische Verdrahtung dieser Verbindung. Die USB-Datenleitungen (USBD+ und USBD-) werden von der USB-C-Buchse direkt zum USB-Seriell-Wandler (FT232RL) geführt, wie in Abbildung 92 dargestellt.

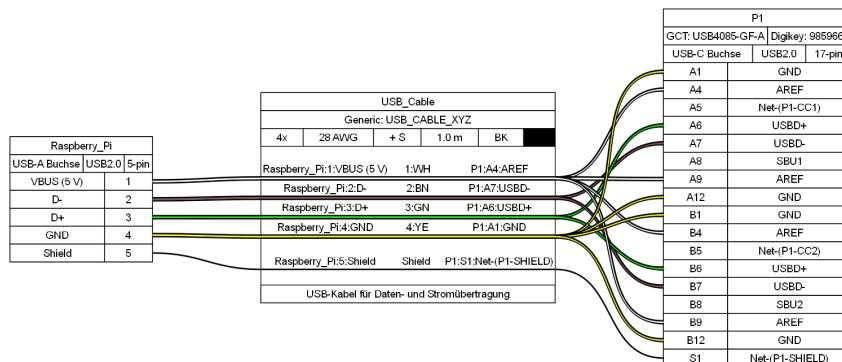


Abb. 91: Verbindung zwischen Raspberry Pi und Platine über USB-C

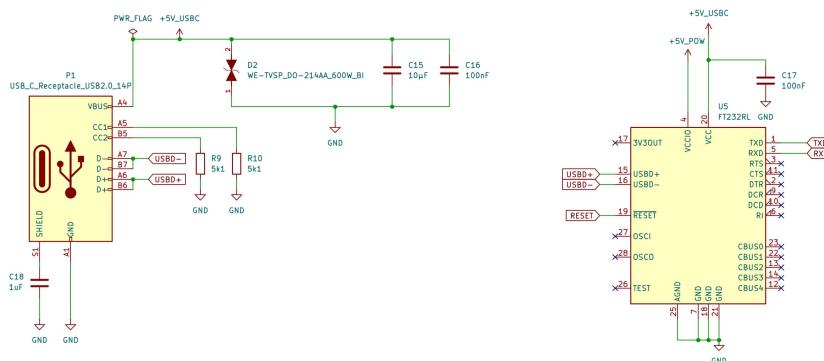


Abb. 92: Schaltplan des USB-C-Eingangs und des Schnittstellenwandlers

Überspannungsschutz

Eine TVS-Diode (D2, WE-TVSP_D0-214AA_600W_BI) ist parallel zu den USB-Versorgungsspannungen geschaltet. Sie schützt die Schaltung vor transienten Spannungsspitzen und sorgt für eine stabile Spannungsversorgung.

Widerstände für USB-Identifikation

Die CC-Pins (CC1 und CC2) sind mit Pull-Down-Widerständen (R9 und R10, jeweils $5.1\text{ k}\Omega$) belegt, um sicherzustellen, dass der USB-C-Anschluss korrekt als Gerät erkannt wird. Diese Widerstände sind gemäß der USB-C-Spezifikation erforderlich. (Matthew, 2021)

Signalstabilisierung

- Ein Entkopplungskondensator (C18, $1\text{ }\mu\text{F}$) zwischen Shield und GND verhindert hochfrequente Störungen.
- Die USB-Datenleitungen werden direkt an den FT232RL weitergeleitet, der die USB-Signale in serielle UART umwandelt.
- Der FT232RL ist mit einem zusätzlichen Entkopplungskondensator (C17, 100 nF) ausgestattet, um eine stabile Versorgungsspannung sicherzustellen.

Datenübertragung und UART-Schnittstelle

Die USB-Daten werden über den FT232RL in ein serielles UART-Protokoll konvertiert, das zur Kommunikation mit dem ATmega328P genutzt wird. Der TXD-Pin des FT232RL sendet Daten an den RXD-Pin des ATmega328P, während der RXD-Pin des FT232RL Daten vom TXD-Pin des ATmega328P empfängt. Die Baudrate kann softwareseitig angepasst werden.

4.8.3 24 V auf 12 V Spannungswandler

Funktionsweise der Schaltung

Die in 93 *Schaltplan des Step-Down-Wandlers*, S.77 dargestellte Schaltung ist ein Step-Down-Schaltregler (Buck-Converter) auf Basis des TPS56837HRPAR von Texas Instruments. Beim Schaltdesign wurde das vorgeschlagene Design von TI Power Systems verwendet. Das Programm hat die Größen aller benötigten Widerstände, Kondensatoren etc. dimensioniert und vorgegeben. Der vorliegende Schaltplan zeigt die Implementierung der Vorgabe in das Vorhaben. Dieser wandelt eine höhere Eingangsspannung (24 V) in eine niedrigere Ausgangsspannung (12 V) um.

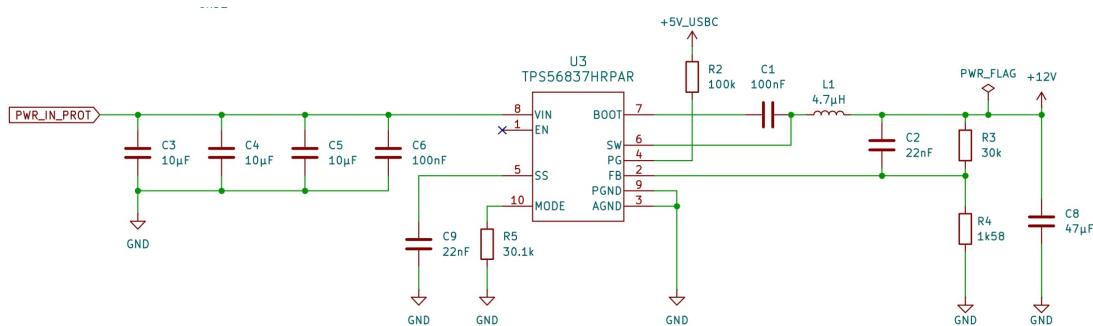


Abb. 93: Schaltplan des Step-Down-Wandlers

Eingangsfilterung und Stabilisierung

Die Eingangsspannung (PWR_IN_PROT, 12 V) wird über mehrere Eingangskondensatoren (C3, C4, C5; jeweils $10\text{ }\mu\text{F}$, C6: 100 nF) stabilisiert. Diese Kondensatoren reduzieren Spannungsschwankungen und filtern hochfrequentes Rauschen heraus, um Störungen durch schnelles Schalten zu minimieren.

Spannungsregelung durch den TPS56837HRPAR

- VIN (Pin 8): Eingangsspannung (24 V) wird an den Regler angelegt.
- EN (Pin 1): Aktivierung des Reglers (dauerhaft aktiv).
- SS (Soft-Start, Pin 5, mit $C9 = 22\text{ nF}$): Verhindert Einschaltstromstöße.
- MODE (Pin 10, mit $R5 = 30,1\text{ k}\Omega$): Legt den Betriebsmodus fest (PWM).

Schalt- und Energieübertragungseinheit

- SW (Switching Node, Pin 6): Hier findet das schnelle Schalten statt.
- BOOT (Pin 7, mit $C_1 = 100 \text{ nF}$ und $L_1 = 4,7 \mu\text{H}$): Versorgt den High-Side MOSFET-Treiber.
- Induktivität $L_1 (4,7 \mu\text{H})$: Speichert Energie und glättet den Stromfluss.
- Kondensator $C_2 (22 \text{ nF})$: Stabilisiert die Regelungsschleife.

Feedback und Spannungsregelung

- FB (Feedback, Pin 2): Spannungsteiler mit $R_3 (30 \text{ k}\Omega)$ und $R_4 (1,58 \text{ k}\Omega)$ regelt die Ausgangsspannung.
- PWR_FLAG: Kennzeichnet eine anliegende Versorgungsspannung.

Ausgangsfilterung

Die gepulste Spannung wird durch den Kondensator $C_8 (47 \mu\text{F})$ geglättet, um eine stabile 12V-Versorgung zu gewährleisten.

Simulation und Berechnung

Die Funktion des Spannungswandlers wird in 4.9.2 *Simulation der 12 V-Spannungsregelung*, S.83 simuliert.

4.8.4 12 V auf 5 V Spannungswandler

Funktionsweise der Schaltung

Die in 94 *Schaltplan des Step-Down-Wandlers von 12 V auf 5 V*, S.78 dargestellte Schaltung ist ein Step-Down-Schaltregler (Buck-Converter) auf Basis des TPS565208 von Texas Instruments. Beim Schaltungsdesign wurde das von TI Power Systems empfohlene Referenzdesign berücksichtigt. Dieses Design gibt die Dimensionierung der Bauelemente wie Widerstände, Kondensatoren und Induktivitäten vor. Der vorliegende Schaltplan zeigt die Implementierung dieser Vorgaben in das Projekt. Der Spannungsregler reduziert die Eingangsspannung von 12 V auf eine stabile Ausgangsspannung von 5 V.

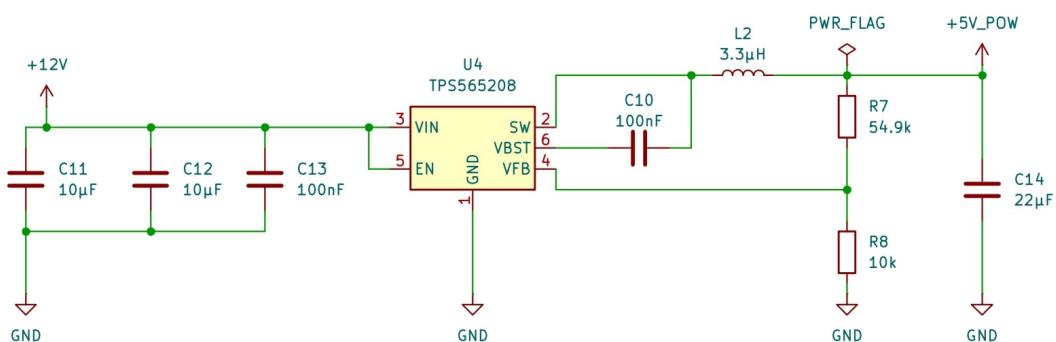


Abb. 94: Schaltplan des Step-Down-Wandlers von 12 V auf 5 V

Eingangsfilterung und Stabilisierung

Die Eingangsspannung (+12 V) wird durch die Kondensatoren C_{11} und C_{12} (jeweils $10 \mu\text{F}$) sowie C_{13} (100 nF) stabilisiert. Diese Komponenten dienen dazu, Spannungsschwankungen zu minimieren und hochfrequente Störungen, die durch das schnelle Schalten entstehen, herauszufiltern.

Spannungsregelung durch den TPS565208

- VIN (Pin 3): Die Eingangsspannung von 12 V wird hier angelegt.
- EN (Pin 5): Aktivierung des Reglers (im Betrieb dauerhaft aktiv).
- GND (Pin 1): Bezugspotenzial für das gesamte System.

Schalt- und Energieübertragungseinheit

- SW (Switching Node, Pin 2): Hier findet der schnelle Schaltvorgang statt.
- VBST (Boost-Kondensator, Pin 6, mit $C_{10} = 100 \text{ nF}$): Unterstützt den High-Side-MOSFET-Treiber.
- Induktivität L2 ($3,3 \mu\text{H}$): Speichert Energie und glättet den Stromfluss.

Feedback und Spannungsregelung

- VFB (Feedback, Pin 4): Spannungsteiler bestehend aus R7 ($54,9 \text{ k}\Omega$) und R8 ($10 \text{ k}\Omega$) bestimmt die Ausgangsspannung.
- PWR_FLAG: Markierung zur Signalisierung der Betriebsspannung.

Ausgangsfilterung

Die durch das Schalten entstehende Restwelligkeit wird durch den Ausgangskondensator C14 ($22 \mu\text{F}$) geglättet, um eine stabile 5 V-Versorgung sicherzustellen.

Simulation und Berechnung

Die Funktionalität des Spannungswandlers wird in 4.9.3 *Simulation der 5 V-Spannungsregelung*, S.85 simuliert.

4.8.5 Lüftersteuerung

Die Steuerung der Lüfter erfolgt über den ATmega328P, welcher die PWM-Signale zur Drehzahlregelung bereitstellt. Die Versorgungsspannung der Lüfter kann über einen 3-Pin-Jumper auf der Platine zwischen 5 V und 12 V umgeschalten werden, wie in folgender Abbildung dargestellt.

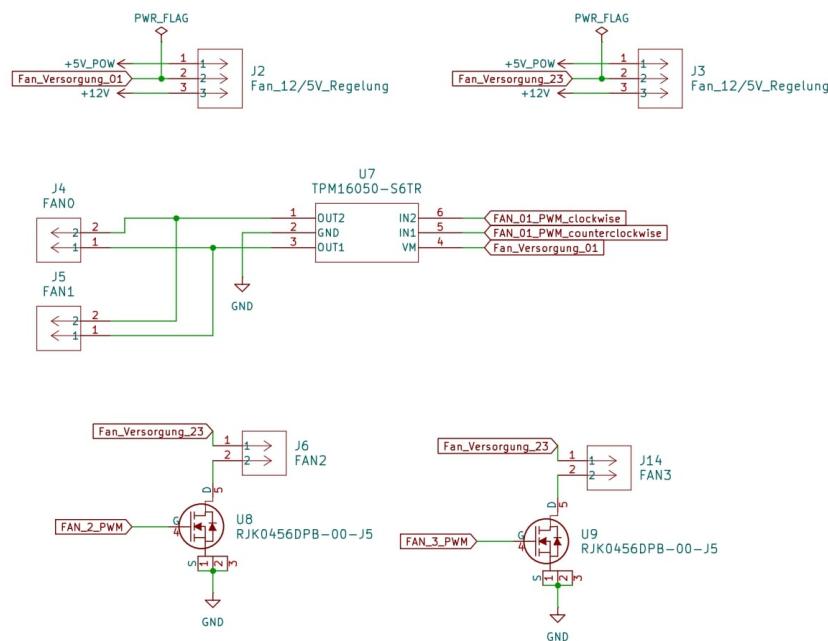


Abb. 95: Schaltplan der Lüftersteuerung mit PWM-Signalen vom ATmega328P

Energieversorgung und MOSFET-Schaltung

Da die PWM-Ausgänge des ATmega328P nicht ausreichend Leistung für die Lüfter liefern können, erfolgt die Ansteuerung über **MOSFETs** (RJK0456DPB) und DC-Motor-Controller. Diese erlauben es, die Lüfter mit einer externen Energiequelle zu betreiben. Die MOSFETs schalten die Versorgungsspannung der Lüfter in Abhängigkeit von den PWM-Signalen durch, sodass eine effiziente Leistungsregelung möglich ist.

Umschaltbare Spannungsversorgung

Die Versorgungsspannung der Lüfter kann über einen 3-Pin-Jumper auf der Platine zwischen 5 V und 12 V umgeschaltet werden. Dies ermöglicht es, je nach eingesetzten Lüftermodellen die geeignete Spannung zu wählen und die Leistungsaufnahme flexibel zu gestalten.

Drehzahl- und Richtungssteuerung

Die Steuerung von FAN0 und FAN1 erfolgt über den DC-Motor-Controller **TPM16050** (9.6 *Datenblatt des DC-Motor-Controllers TPM16050*, S.215). Dieser ermöglicht es, softwareseitig die Drehrichtung der Lüfter zu beeinflussen. Dazu werden die PWM-Signale `FAN_01_PWM_clockwise` und `FAN_01_PWM_counterclockwise` vom ATmega328P ausgegeben. Abhängig davon, welches Signal aktiv ist, rotiert der Lüfter in die eine oder andere Richtung.

PWM-Steuerung für FAN2 und FAN3

Die Lüfter FAN2 und FAN3 werden ebenfalls über PWM-Signale vom ATmega328P angesteuert. Allerdings erfolgt hier keine Richtungssteuerung. Stattdessen wird die Drehzahl über die PWM-Signale `FAN_2_PWM` und `FAN_3_PWM` reguliert.

4.9 Simulationen und Berechnungen

In diesem Abschnitt werden die wichtigsten Berechnungen und Simulationen für die Elektronik des FiberPrinters dargestellt.

4.9.1 Simulation Verpolungsschutz

Elektronische Schaltungen müssen vor einer falschen Polarität der Versorgungsspannung geschützt werden, da eine Verpolung zu einer Beschädigung von Bauteilen führen kann. Eine bewährte Methode hierfür ist der Einsatz eines PMOS-FETs in Kombination mit einer Zener-Diode (Alex, 2021). Diese Simulation zeigt eine solche Schaltung und analysiert das Verhalten bei korrekter sowie umgekehrter Polung der Eingangsspannung.

Schaltungsaufbau

Die simulierte Schaltung besteht aus folgenden Hauptkomponenten:

- **PMOS-FET (SQJ409EP):** Dieser dient als Schalter und blockiert den Stromfluss bei negativer Eingangsspannung.
- **Zener-Diode (1SMB5927BT3G):** Sie stabilisiert die Gate-Source-Spannung des PMOS-FETs.
- **Widerstände (R1 = 100 kΩ, R2 = 4 Ω):** R1 sorgt für eine sichere Gate-Ladung und Entladung, während R2 die Last simuliert.
- **Spannungsquelle (V1):** Eine gesteuerte Spannungsquelle, die von +24 V auf -24 V umschaltet, um den Effekt der Verpolung zu analysieren.

Die folgende Abbildung zeigt den Schaltungsaufbau:

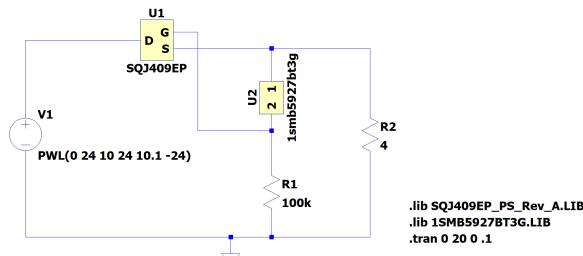


Abb. 96: Schaltung der Simuation des Verpolungsschutzes mit PMOS-FET und Zener-Diode

Simulationsergebnisse

Verhalten der Ausgangsspannung bei Verpolung

Die erste Simulation zeigt die Eingangsspannung (grün) und die Spannung an der Last (blau).

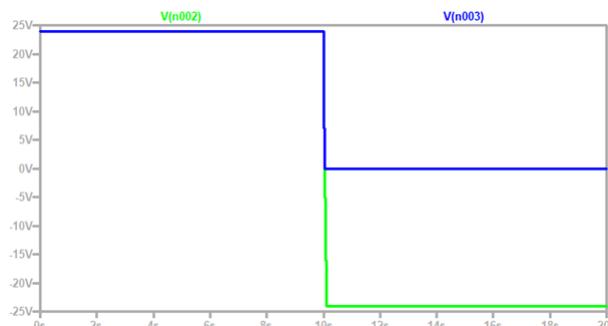


Abb. 97: Simulation der Eingangsspannung (grün) und Ausgangsspannung an der Last (blau)

Analyse

- Solange die Eingangsspannung positiv ist (+24 V), wird der PMOS-FET durchgeschaltet, und die Last erhält die volle Eingangsspannung.
- Sobald die Eingangsspannung auf -24 V wechselt (Verpolung), sperrt der PMOS-FET, und die Spannung an der Last fällt auf 0 V.
- Der Verpolungsschutz funktioniert einwandfrei, da die Last vor der negativen Spannung geschützt wird.

Stabilisierung der Gate-Source-Spannung

In der zweiten Simulation wurde die Eingangsspannung (grün) und die Gate-Source-Spannung des PMOS-FETs (blau) aufgezeichnet.

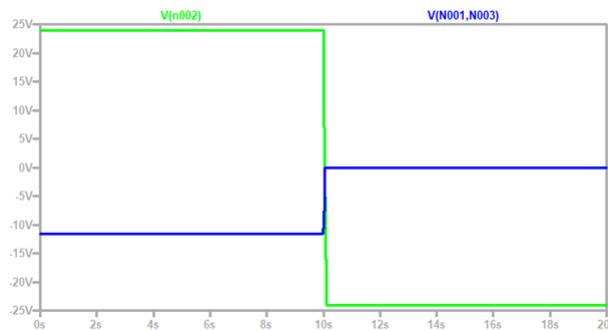


Abb. 98: Simulation der Eingangsspannung (grün) und Gate-Source-Spannung des PMOS-FETs (blau)

Analyse

- Im normalen Betrieb wird die Gate-Source-Spannung durch die Zener-Diode auf einen stabilen Wert begrenzt.
- Bei Verpolung bleibt die Gate-Source-Spannung im sicheren Bereich, sodass der FET vollständig gesperrt und keinen Stromfluss zulässt.
- Die Zener-Diode sorgt somit für eine zuverlässige Spannungsbegrenzung und verhindert eine ungewollte Durchschaltung des PMOS-FETs.

Fazit

Die Simulation bestätigt, dass die implementierte Schaltung zuverlässig als Verpolungsschutz fungiert. Der PMOS-FET bleibt bei negativer Eingangsspannung gesperrt, wodurch die Last vor schädlichen Verpolungseffekten geschützt wird. Die Zener-Diode stabilisiert die Gate-Source-Spannung und gewährleistet eine sichere Funktion des FETs. Diese Schaltung ist somit eine effektive Lösung für den Schutz von empfindlichen elektronischen Komponenten vor Verpolung.

4.9.2 Simulation der 12 V-Spannungsregelung

Die folgende Simulation bildet die in 4.8.3 *24 V auf 12 V Spannungswandler*, S.77 dargestellte Schaltung nach und wurde mit dem Programm TI Power Systems (1.8 *Verwendete Software*, S.4) durchgeführt. Sie kann vollständig im Anhang 9.9 *12 V-Schaltregler Simulation*, S.218 gefunden werden. Die Simulation stellt das Verhalten des Schaltreglers unter verschiedenen Betriebsbedingungen grafisch dar. Im folgenden Abschnitt wird der Betriebspunkt mit einer Eingangsspannung von 24V, einer Ausgangsspannung von 12V und einem Ausgangstrom von 4A (entnommen aus dem Strombedarf Abbildung 78 (*Übersicht der Komponenten mit Strom- und Spannungsbedarf*), S.66) analysiert. Die entsprechenden Werte für diesen Betriebspunkt werden der Simulation entnommen.

Duty Cycle

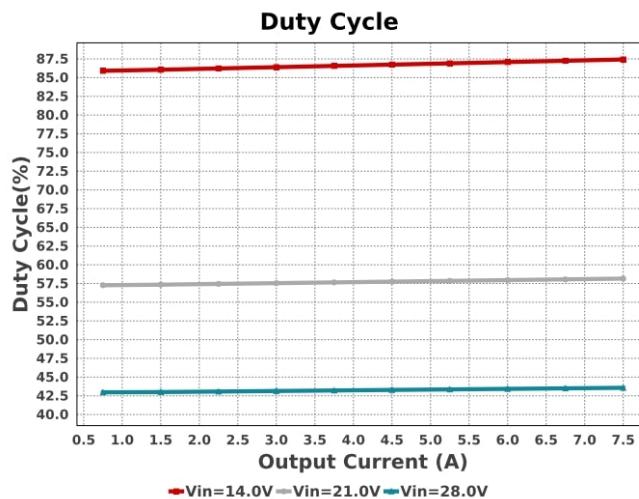


Abb. 99: Duty Cycle in Abhängigkeit des Ausgangsstroms. (aus 9.7 *Layout Empfehlungen aus dem Datenblatt des TPS56387*, S.216)

Der Duty-Cycle beschreibt das Verhältnis der Einschaltzeit zur gesamten Periodendauer eines Schaltzyklus des Schaltreglers. In der Abbildung 99 ist zu erkennen, dass der Duty-Cycle mit zunehmendem Ausgangstrom leicht ansteigt. Für den gegebenen Anwendungsfall ergibt sich ein Duty-Cycle von 50.46%.

Effizienz

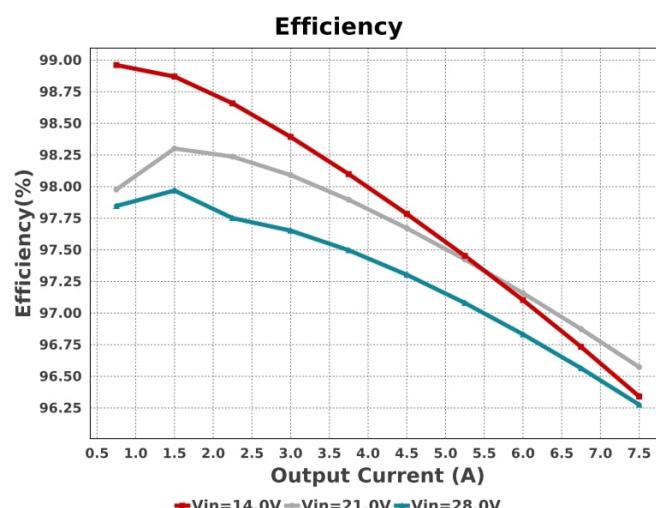


Abb. 100: Effizienz in Abhängigkeit des Ausgangsstroms. (aus 9.7 *Layout Empfehlungen aus dem Datenblatt des TPS56387*, S.216)

Die Effizienz gibt das Verhältnis der Ausgangsleistung zur Eingangsleistung an und beschreibt den Energieverlust durch Schaltverluste und Widerstandsverluste in der Schaltung. Wie in Abbildung 100 dargestellt, nimmt die Effizienz mit steigendem Ausgangsstrom ab. Für die gegebenen Betriebsbedingungen resultiert eine Effizienz von 97.60%.

IC-Junction-Temperatur

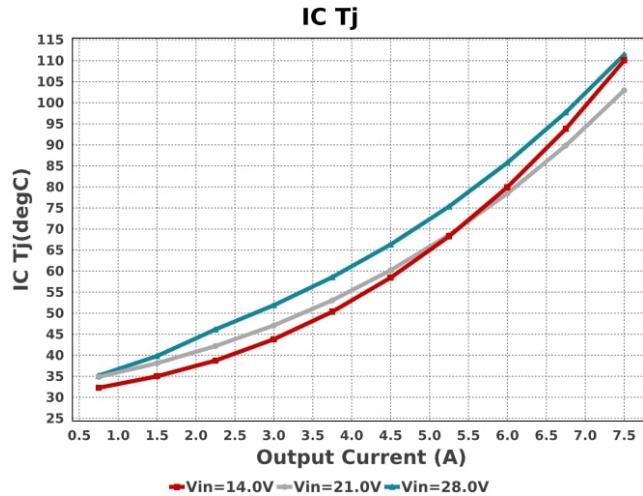


Abb. 101: Junction-Temperatur des ICs in Abhängigkeit des Ausgangsstroms. (aus 9.7 *Layout Empfehlungen aus dem Datenblatt des TPS56387*, S.216)

Die Junction-Temperatur (T_j) beschreibt die Temperatur der aktiv arbeitenden Region des Halbleiters. Sie ist ein entscheidender Parameter für die thermische Belastung der Schaltung. Abbildung 101 zeigt, dass die Temperatur mit zunehmendem Ausgangsstrom ansteigt. Unter den gegebenen Betriebsbedingungen resultiert eine Junction-Temperatur von $58.28^\circ C$.

Ripple Voltage

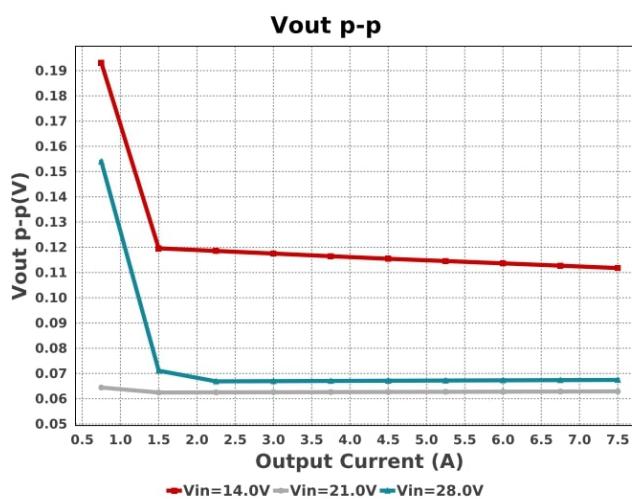


Abb. 102: Ripple-Voltage in Abhängigkeit des Ausgangsstroms. (aus 9.7 *Layout Empfehlungen aus dem Datenblatt des TPS56387*, S.216)

Die Ripple-Voltage bezeichnet die Restwelligkeit der Ausgangsspannung eines Schaltnetzteils. Sie entsteht durch das periodische Laden und Entladen der Ausgangskapazität sowie durch den Schaltbetrieb der Spannungsregelung. Eine geringe Ripple-Spannung ist entscheidend für eine stabile Spannungsversorgung empfindlicher Schaltungsteile. In Abbildung 102 ist ersichtlich, dass die Ripple-Voltage mit zunehmendem Strom abnimmt. Für den gegebenen Anwendungsfall beträgt die Spitzen-Spitzen-Welligkeit der Ausgangsspannung $63.43 mV$.

4.9.3 Simulation der 5 V-Spannungsregelung

Die folgende Simulation bildet die in 4.8.4 *12 V auf 5 V Spannungswandler*, S.78 dargestellte Schaltung nach und wurde mit dem Programm TI Power Systems (1.8 *Verwendete Software*, S.4) durchgeführt. Sie kann vollständig im Anhang 9.10 *5 V-Schaltregler Simulation*, S.223 gefunden werden. Die Simulation stellt das Verhalten des Schaltreglers unter verschiedenen Betriebsbedingungen grafisch dar. Im folgenden Abschnitt wird der Betriebspunkt mit einer Eingangsspannung von 12 V, einer Ausgangsspannung von 5 V und einem Ausgangsstrom von 2 A (entnommen aus dem Strombedarf Abbildung 78 (*Übersicht der Komponenten mit Strom- und Spannungsbedarf*), S.66) analysiert. Die entsprechenden Werte für diesen Betriebspunkt werden der Simulation entnommen.

Duty Cycle

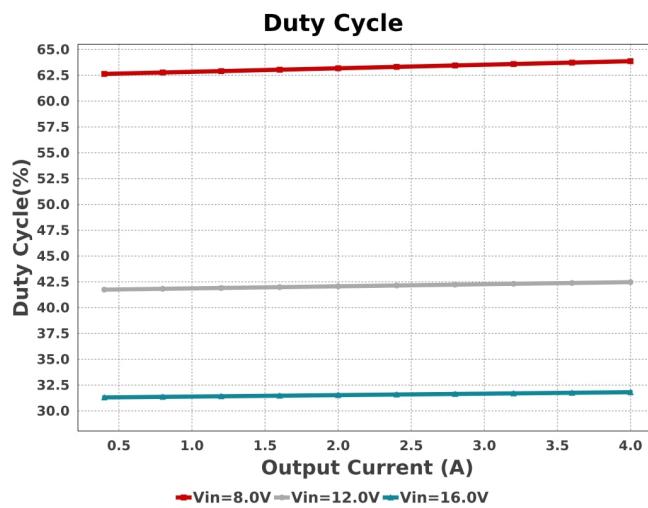


Abb. 103: Duty Cycle in Abhängigkeit des Ausgangsstroms. (aus 9.8 *Layout Empfehlungen aus dem Datenblatt des TPS565208*, S.217)

In der Abbildung 103 ist zu erkennen, dass der Duty-Cycle mit zunehmendem Ausgangsstrom leicht ansteigt. Für den gegebenen Anwendungsfall ergibt sich ein Duty-Cycle von 42.07%.

Effizienz

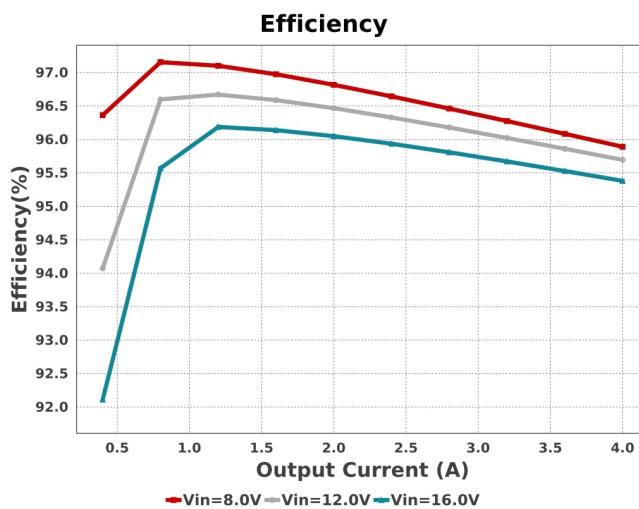


Abb. 104: Effizienz in Abhängigkeit des Ausgangsstroms. (aus 9.8 *Layout Empfehlungen aus dem Datenblatt des TPS565208*, S.217)

Abbildung 104 zeigt, dass die Effizienz mit steigendem Ausgangsstrom abnimmt. Für die gegebenen Betriebsbedingungen resultiert eine Effizienz von 96.5%.

IC-Junction-Temperatur

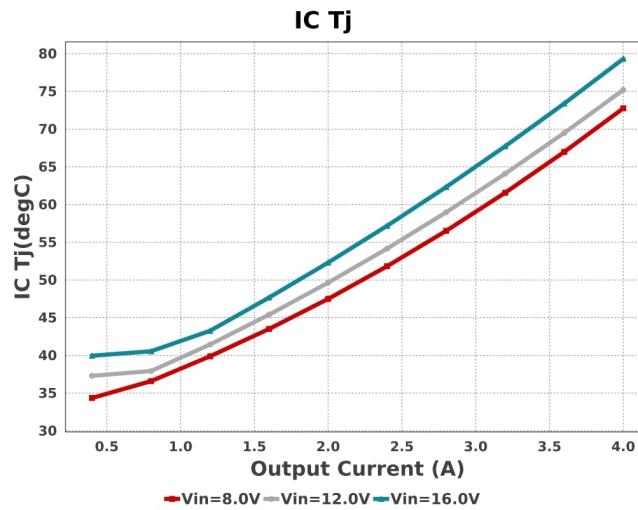


Abb. 105: Junction-Temperatur des ICs in Abhängigkeit des Ausgangsstroms. (aus 9.8 *Layout Empfehlungen aus dem Datenblatt des TPS565208*, S.217)

Abbildung 105 zeigt, dass die Temperatur mit zunehmendem Ausgangstrom ansteigt. Unter den gegebenen Betriebsbedingungen resultiert eine Junction-Temperatur von $49.64^{\circ}C$.

Ripple Voltage

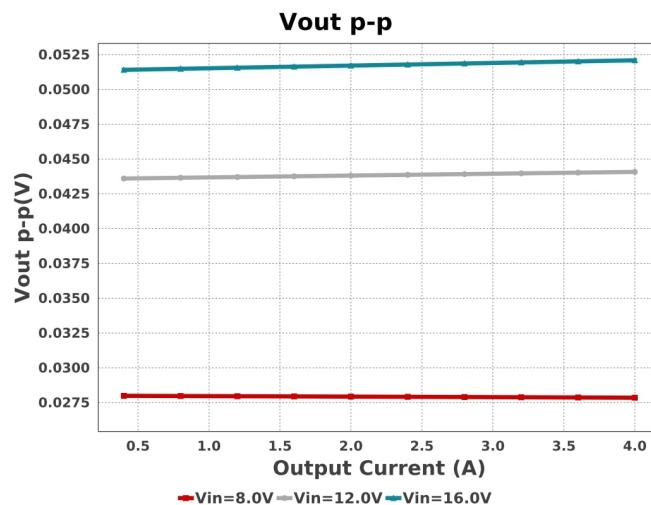


Abb. 106: Ripple-Voltage in Abhängigkeit des Ausgangsstroms. (aus 9.8 *Layout Empfehlungen aus dem Datenblatt des TPS565208*, S.217)

In Abbildung 106 ist ersichtlich, dass die Ripple-Voltage mit zunehmendem Strom zunimmt. Für den gegebenen Anwendungsfall beträgt die Spitzen-Spitzen-Welligkeit der Ausgangsspannung 43.81 mV .

4.10 Leiterplatten-Design (PCB-Layout)

In diesem Abschnitt wird das Leiterplatten-Design der Druckkopf-Platine beschrieben. Dabei wird genauer auf das Layout des 12 V-Spannungsreglers und des 5 V-Spannungsreglers eingegangen. Das gesamte Leiterplatten-Design kann im Anhang 9.5 *Vollständiges Layout*, S.212 analysiert werden. Das endgültige Design wird bei einem Online-Hersteller gefertigt. Die Designregeln des Herstellers sowie die Layout-Vorgaben aus den Datenblättern relevanter Komponenten werden berücksichtigt.

4.10.1 Allgemein

Modularer Aufbau

Die Bauteile sind auf funktionale Baugruppen aufgeteilt, um Leitungslängen zu minimieren und die Signalintegrität zu optimieren. Das Gesamtlayout ist in drei Module gegliedert, wie bereits in 4.6 *Modulare Aufteilung der Elektronik*, S.64 beschrieben. Beim Layout wird sichergestellt, dass die Bauteile entsprechend ihrer Funktionalität dem korrekten Modul zugewiesen und dort platziert werden.

Designoptimierung

Zur Verbesserung der elektrischen Eigenschaften werden folgende Maßnahmen umgesetzt:

- **Minimierung der Vias:** Signale werden mit einer minimalen Anzahl an Vias verdrahtet. Die Leitungen sind so kurz wie möglich ausgeführt, um Widerstände und parasitäre Effekte zu reduzieren.
- **Optimierung der Leiterbahnen:** Nach der vollständigen Verdrahtung wird die Breite der Leiterbahnen maximiert und der Abstand zwischen den Bahnen vergrößert, um Signalstörungen zu minimieren. Diese Maßnahmen werden so weit wie möglich umgesetzt, auch wenn in bestimmten Bereichen Einschränkungen bestehen.

Designprozess

Bei der PCB-Layout-Entwicklung wurde ein systematischer Ansatz implementiert, unter strikter Einhaltung der Hersteller-spezifischen Designregeln. Die Bauteilplatzierung folgte dem funktionalen Modulkonzept mit besonderem Augenmerk auf Signalintegrität und EMV-Charakteristik.

Layoutfertigung

Aufgrund der hohen Integrationsdichte und der erforderlichen Präzision wurde die Fertigung an den Dienstleister JLCPCB ausgelagert. Die Designparameter wurden an die Fertigungsspezifikationen adaptiert, wobei folgende technische Limitierungen maßgebend waren:

Tabelle 19: Designvorgaben gemäß JLCPCB (JLCPCB, 2025)

Beschreibung	Wert
Mindestleiterbahnbreite	0,15 mm
Mindestabstand zwischen Leiterbahnen	0,15 mm
Mindestabstand zwischen Leiterbahn und Via-Pad	0,15 mm
Mindestabstand zwischen Leiterbahn und Massefläche	0,15 mm
Mindestabstand zwischen Via-Pad und Massefläche	0,15 mm
Mindestabstand zwischen Via-Pad und BGA-Pad	0,15 mm
Mindestabstand zwischen Massefläche und Massefläche	0,15 mm
Mindestabstand zwischen PCB-Kante und Leiterbahn	0,2 mm
Mindestabstand zwischen PCB-Kante und Pad	0,2 mm
Mindestabstand zwischen PCB-Kante und Massefläche	0,2 mm
Mindestbreite des Annular Rings	0,15 mm

4.10.2 Layout des 12 V-Spannungsreglers

Das Layout des 12V-Spannungsreglers ist in Abbildung 109 dargestellt. Sämtliche Bauteile befinden sich auf dem Leistungsmodul gruppiert und vorwiegend auf der Rückseite des Moduls platziert, um eine thermische Entkopplung vom Mikrocontroller zu gewährleisten.

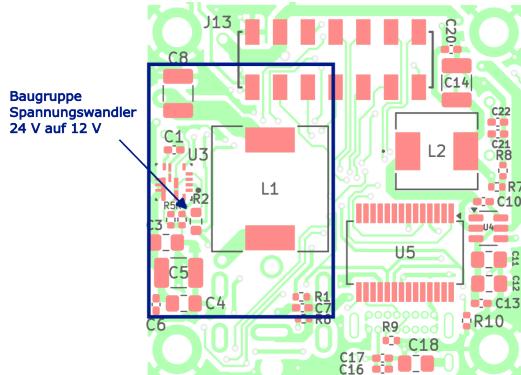


Abb. 107: Layout Unterseite

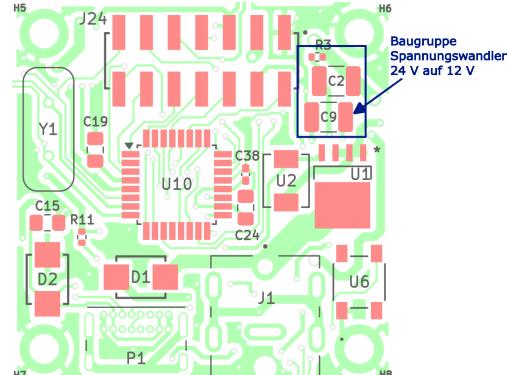


Abb. 108: Layout Oberseite

Abb. 109: Layout des 12 V-Spannungsreglers

Berücksichtigung der Layout-Guidelines

Das Design des Layouts folgt den relevanten Empfehlungen aus dem Datenblatt des **TPS56387** (siehe 9.7 *Layout Empfehlungen aus dem Datenblatt des TPS56387*, S.216). Insbesondere berücksichtigt es folgende Aspekte:

- Maximale Masseflächenteilung zur Verbesserung der thermischen Leistung.
- Breiteste mögliche VIN- und PGND-Leiterbahnen zur Reduzierung des Bahnwiderstands und zur Wärmeableitung.
- Eingangs- und Ausgangskondensator in direkter Nähe zum Bauteil zur Minimierung des Bahnwiderstands und parasitärer Effekte.
- Kurze und breite SW (Switching Node)-Leiterbahn zur Reduzierung abgestrahlter Störungen.
- Breite PGND-Leiterbahnen zwischen Ausgangskondensator und PGND-Pin zur Minimierung des Bahnwiderstands.

4.10.3 Layout des 5V-Spannungsreglers

Das Layout des 12 V-auf-5 V-Spannungsreglers ist in Abbildung 110 dargestellt. Die Baugruppe ist auf der Leiterplatte klar abgegrenzt und umfasst den **TPS565208** als Spannungsregler (DigiKey, 2025ac) (siehe 9.8 *Layout Empfehlungen aus dem Datenblatt des TPS565208*, S.217).

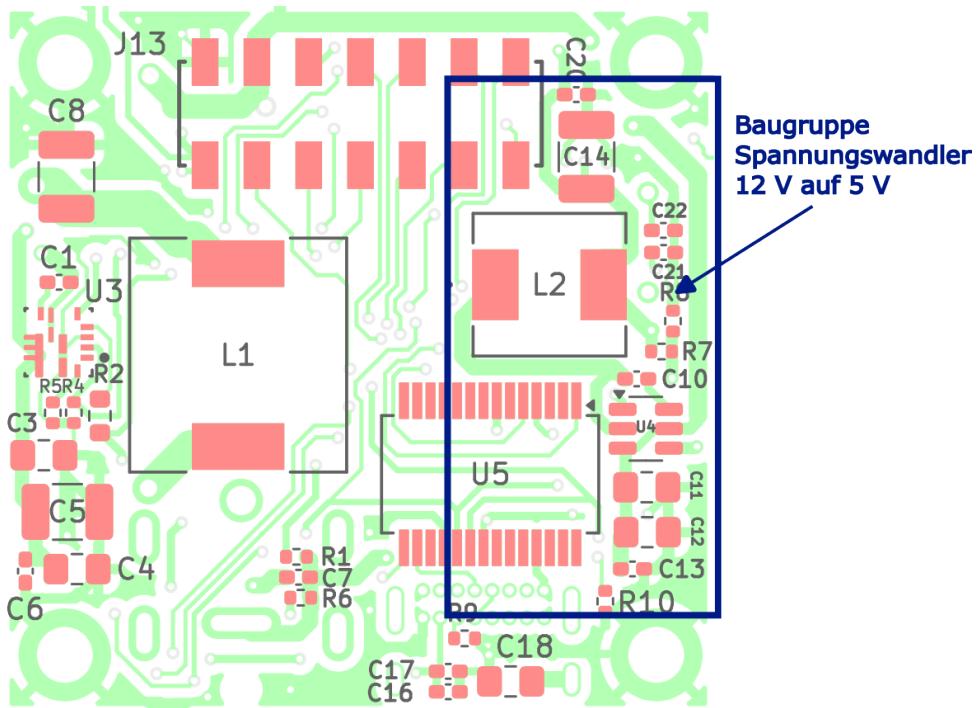


Abb. 110: Layout des 5 V-Spannungsreglers

Berücksichtigung der Layout-Guidelines

Das Layout wurde gemäß den Empfehlungen aus dem Datenblatt des **TPS565208** umgesetzt und folgt insbesondere den folgenden Designrichtlinien:

- Breiteste mögliche VIN- und GND-Leiterbahnen zur Reduzierung des Bahnwiderstands und Verbesserung der Wärmeableitung.
- Eingangs- und Ausgangskondensator in direkter Nähe zum Bauteil zur Minimierung parasitärer Effekte.
- Ausreichende Vias für Eingangs- und Ausgangskondensator zur stabilen Verbindung mit der Massebene.
- Kurze und breite SW-Leiterbahn zur Reduzierung abgestrahlter Störungen.
- Vermeidung hochfrequenter Schaltströme unterhalb des Bauteils.
- Separater VOUT-Pfad zur Reduzierung von Störeinflüssen auf die Regelung.
- Minimale VFB-Leiterbahn zur Vermeidung von Störungskopplungen.
- Breite GND-Leiterbahn zwischen Ausgangskondensator und GND-Pin zur Reduzierung des Bahnwiderstands.

4.10.4 Zusammenführung der Module

Die einzelnen Module sind auf einer Platine zusammengeführt, um die Beschaffungskosten zu reduzieren und die Integration zu vereinfachen. Dabei sind die Module nahezu vollständig ausgefräst, wobei kleine Brücken zur Implementierung der vorgesehenen Testpunkte erhalten bleiben.

Nach erfolgreicher Überprüfung der Funktionalität über die Testpunkte und der Programmierung des ATmega lassen sich die einzelnen Module aus der Gesamtplatine herauslösen und final montieren.

Das gesamte Design weist eine Abmessung von 100×100 mm auf, um die Herstellungskosten der Platine zu minimieren. Das finale PCB-Design ist in den folgenden Abbildungen dargestellt:

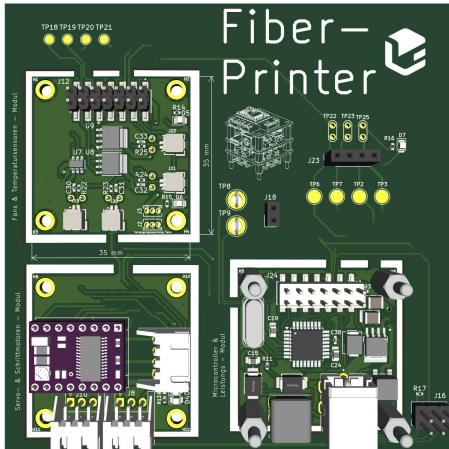


Abb. 111: Oberseite des fertigen PCB-Designs

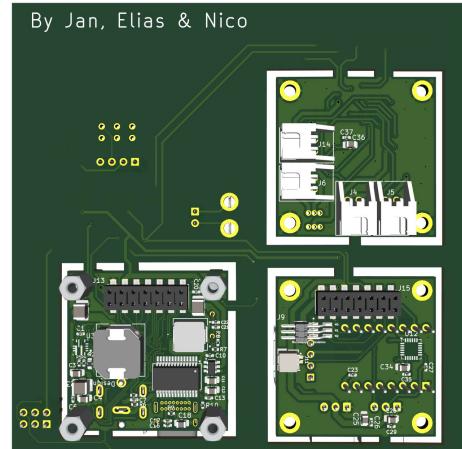


Abb. 112: Unterseite des fertigen PCB-Designs

Nach finaler Montage sieht das gestapelte Design wie folgt aus:

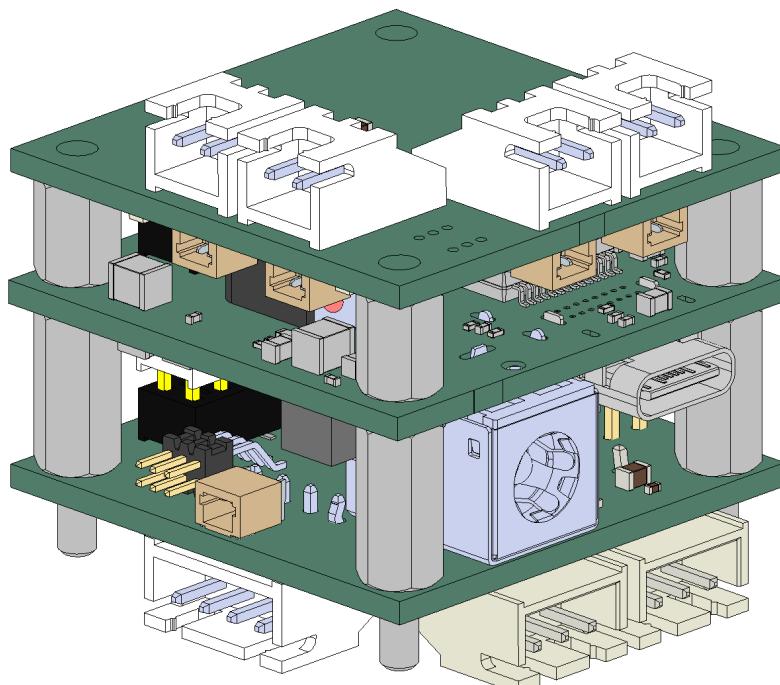


Abb. 113: Gestapeltes PCB-Design

4.10.5 Beschaffung

Um die Platine zu bestellen, wird das fertige Layout zunächst als Gerber-Datei exportiert und auf der Plattform JLCPCB hochgeladen. Basierend auf diesen Daten werden die Herstellungskosten berechnet. Die voraussichtlichen Kosten für die Fertigung und den Versand der Platinen lassen sich wie folgt zusammenfassen:

Kosten der Platinenfertigung

Die Herstellung der Mindestbestellmenge von 5 Stück ohne Bestückung kostet etwa 2 €. Hinzu kommen Versandkosten von ca. 7 €, sodass sich die Gesamtkosten für die unbestückten Platinen auf rund 9 € belaufen.

Für die Bestellung wurden folgende Spezifikationen gewählt:

- **Anzahl der Lagen:** 2
- **Platinenfarbe:** Schwarz
- **Platinenstärke:** 1,6 mm
- **Oberflächenfinish:** HASL
- **Kupferdicke:** 1 oz
- **Mindestanzahl:** 5 Stück
- **Abmessungen:** 100 x 100 mm

Bestellung des Stencils Zur Erleichterung des Lötprozesses (siehe 4.11 *Lötprozess*, S.92) wird ein Stencil mitbestellt. Dieser kostet etwa 7 €. Durch den zusätzlichen Versand steigen die Versandkosten auf insgesamt ca. 21 €, wodurch sich die Gesamtkosten der Bestellung auf rund 30 € belaufen.

Für den Stencil wurden folgende Parameter gewählt:

- **Framework:** Nein
- **Größe:** 380 x 280 mm
- **Stencil-Seite:** Ober- und Unterseite auf einem Stencil
- **Anzahl:** 1

Lieferzeit

Die geschätzte Lieferzeit beträgt etwa zwei Wochen.

4.11 Lötprozess

Aufgrund der geringen Größe der Leiterbahnen und Bauteile erfolgt die Fertigung der Platinen durch JLCPCB. Die Bestückung wird jedoch manuell durchgeführt. Zur Sicherstellung einer hohen Lötqualität sind die folgenden Aspekte zu berücksichtigen:

Reflow-Lötverfahren für SMD-Komponenten

Für eine effiziente und zuverlässige Verarbeitung der SMD-Bauteile kommt ein Reflow-Ofen zum Einsatz. Besonders die Bauform des TPS56837HRPAR erfordert dieses Verfahren. Wann immer möglich, erfolgt die Lötzung aller SMD-Komponenten mittels Reflow-Lötverfahren, da dieses eine gleichmäßige und zuverlässige Lötverbindung gewährleistet. Hierfür wird das bestellte Stencil verwendet, um die Lötpaste präzise auf die Pads der Platine aufzutragen.

Herausforderungen bei doppelseitiger Bestückung

Da die Platine doppelseitig bestückt ist, besteht die Möglichkeit, dass beim erneuten Erhitzen der zweiten Seite bereits verlötete Bauteile wieder aufschmelzen. Zur Vermeidung dieses Problems kommt eine bewährte Strategie zum Einsatz:

- Bleifreie Lötpaste besitzt eine höhere Schmelztemperatur von etwa 217–220 °C.
- Bleihaltige Lötpaste schmilzt bereits bei ca. 183 °C.
- Die erste Seite wird mit bleifreier Lötpaste verlötet, da diese höheren Temperaturen standhält.
- Die zweite Seite wird mit bleihaltiger Lötpaste verarbeitet, um ein erneutes Aufschmelzen der bereits gelöteten Komponenten auf der Unterseite zu verhindern.

Temperaturanforderungen für hitzeempfindliche Bauteile

Bestimmte Bauteile der Platine sind besonders empfindlich gegenüber hohen Temperaturen. Dazu gehören:

- ATMEGA328P-AU
- FT232RL
- Sunfounder GY-521 (MPU-6050)

Diese Komponenten weisen einen zulässigen Betriebstemperaturbereich von -40 °C bis 85 °C auf. Insbesondere der ATMEGA328P-AU kann bei Temperaturen über 85 °C in seiner Funktion beeinträchtigt werden. Der MPU-6050 kann außerhalb seines spezifizierten Temperaturbereichs Messungenauigkeiten oder Funktionsausfälle aufweisen. Deshalb werden diese Bauteile nach dem Reflow-Lötprozess manuell verlötet, um die Temperaturkontrolle zu gewährleisten.

5 Firmware

5.1 Einleitung

Firmware bezeichnet die Software, die direkt auf den Steuerplatten (Mainboards) von 3D-Druckern ausgeführt wird. Sie ist essenziell für die Steuerung der Motoren, Sensoren und weiterer Peripherie und ermöglicht die Umsetzung von Druckbefehlen (8.1.6 *Firmware*, S.203). Dabei werden Eingaben, die beispielsweise von einem Slicer oder einer höheren Softwareebene (z. B. einem Webinterface) stammen, in hardwarekompatible Signale übersetzt. Somit stellt die Firmware eine zentrale Komponente zur Regelung und Steuerung aller Druckprozesse dar.

5.2 Verfügbare Firmware-Optionen

Zur Steuerung von 3D-Druckern stehen verschiedene Firmware-Lösungen zur Verfügung, die jeweils spezifische Vor- und Nachteile aufweisen. Zu den gängigsten Firmwares zählen:

- **Marlin:** Eine etablierte und weit verbreitete Firmware, die sich durch hohe Zuverlässigkeit und umfangreiche Anpassungsmöglichkeiten auszeichnet. Sie wird in einer Vielzahl von 3D-Druckern verwendet und bietet umfassende Unterstützung für unterschiedliche Hardwarekonfigurationen.
- **Klipper:** Eine moderne Firmware, die durch die parallele Verarbeitung von G-Code-Befehlen auf einem Raspberry Pi eine gesteigerte Druckgeschwindigkeit und Präzision ermöglicht. Diese Architektur reduziert die Berechnungsanforderungen an das Drucker-Mainboard und optimiert die Steuerung der Bewegungsabläufe.
- **RepRapFirmware:** Eine Firmware, die speziell für RepRap-Drucker entwickelt wurde und eine einfache Anpassung an verschiedene Hardwarekonfigurationen erlaubt. Sie verfügt über eine benutzerfreundliche Web-Oberfläche zur Steuerung und Überwachung des Druckprozesses.

5.3 Auswahl der Firmware

Für die Steuerung des 3D-Druckkopfs wurde **Klipper** gewählt, da diese Firmware zahlreiche technische Vorteile bietet:

- **Benutzerfreundlichkeit:** Durch die Web-Oberfläche **Mainsail** ermöglicht Klipper eine intuitive Steuerung, einfache Kalibrierung und Echtzeit-Anpassung des Druckprozesses.
- **Hohe Flexibilität ohne Mainboard-Eingriff:** Die Hauptberechnungen erfolgen auf einem Raspberry Pi, wodurch Hardware-Änderungen bequem über eine Konfigurationsdatei vorgenommen werden können, ohne die Firmware neu flashen zu müssen.
- **Einfache Erweiterbarkeit:** Plugins und Skripte lassen sich problemlos integrieren, wodurch beispielsweise die Ansteuerung eines Servomotors ohne erheblichen Mehraufwand realisiert werden kann.

5.4 Klipper: Definition und Funktionsweise

Klipper ist eine alternative Firmware für 3D-Drucker, die eine Aufgabenverteilung zwischen zwei Einheiten vorsieht:

- Ein Raspberry Pi (oder ein vergleichbarer Einplatinencomputer) übernimmt komplexe Berechnungen, wie die Berechnung der Bewegungsprofile.
- Ein Mikrocontroller (z. B. das vorhandene 3D-Drucker-Mainboard) dient als Echtzeit-Steuereinheit und führt die Befehle aus.

Diese Architektur entlastet den Mikrocontroller erheblich, da rechenintensive Aufgaben auf den Raspberry Pi ausgelagert werden. Dadurch wird eine höhere Druckgeschwindigkeit erreicht, da Beschleunigungs- und Geschwindigkeitsberechnungen effizienter durchgeführt werden können.

5.5 Installation von Klipper

Zur Inbetriebnahme von Klipper wurde das vorgefertigte **Klipper Mainsail Image** verwendet. Dieses Image enthält bereits eine vorkonfigurierte Umgebung mit allen notwendigen Komponenten, wodurch keine manuelle Installation einzelner Softwarebestandteile erforderlich war. Nach dem Flashen des Images auf eine microSD-Karte und dem Einlegen in den Raspberry Pi konnte die Klipper-Umgebung direkt gestartet werden.

5.5.1 Verwaltung und Aktualisierung mit KIAUH

Zur Verwaltung und Aktualisierung von Klipper wurde das Tool **KIAUH** (*Klipper Installation And Update Helper*) eingesetzt. KIAUH bietet eine benutzerfreundliche Möglichkeit zur Installation und Aktualisierung von Klipper sowie zugehöriger Software wie **Moonraker**, **Mainsail** oder **Fluidd**. Dies war insbesondere relevant, da ein eigener **Fork** von Klipper genutzt wurde, welcher regelmäßig aktualisiert und gewartet werden musste.

Die Installation von KIAUH erfolgte über das Terminal des Raspberry Pi, wie in Listing 1 dargestellt:

Listing 1: Installation von KIAUH

```
1 git clone https://github.com/th33xitus/kiauh.git
2 cd kiauh
3 ./kiauh.sh
```

Nach dem Start des Skripts wurde eine interaktive Benutzeroberfläche im Terminal angezeigt, über die sich Klipper, Moonraker, Mainsail und weitere Komponenten effizient verwalten ließen. Dadurch konnte der eigene Klipper-Fork problemlos hinzugefügt, aktualisiert und getestet werden, ohne manuelle Kompilierungen oder komplexe Konfigurationsanpassungen vornehmen zu müssen.

5.6 Klippy: Die zentrale Steuerlogik von Klipper

Klippy ist die Hauptsoftwarekomponente der Klipper-Firmware, die auf einem Raspberry Pi ausgeführt wird und für die Verarbeitung von G-Code-Befehlen verantwortlich ist. Sie übernimmt die Berechnungen zur Bewegungssteuerung und übermittelt die resultierenden Anweisungen an das Drucker-Mainboard, das als Echtzeitsteuerung fungiert.

5.6.1 Interaktion zwischen Klippy und dem Mainboard

Die Kommunikation zwischen Klippy und dem Mainboard erfolgt über eine serielle Schnittstelle. Klippy verarbeitet die Befehle, berechnet die optimalen Bewegungsabläufe und gibt diese in einer optimierten Form an den Mikrocontroller weiter. Dadurch wird eine präzisere Steuerung und eine Erhöhung der Druckgeschwindigkeit ermöglicht, da rechenintensive Prozesse auf dem leistungsstärkeren Steuerrechner ausgeführt werden, anstatt den begrenzten Mikrocontroller zu belasten.

5.7 Mainsail: Webinterface zur Steuerung von Klipper

Mainsail ist eine für Klipper optimierte Weboberfläche, die eine benutzerfreundliche Steuerung und Überwachung des 3D-Druckprozesses ermöglicht. Die Software wird auf einem Raspberry Pi betrieben und bietet eine intuitive Bedienoberfläche. Zu den wichtigsten Funktionen zählen:

- Echtzeitüberwachung von Temperaturverläufen, Druckfortschritt und Systemstatus.
- Direkte Eingabe von G-Code-Befehlen und Anpassung der Druckparameter.
- Verwaltung und Start von G-Code-Dateien über die Weboberfläche.
- Nutzung von Makros zur Automatisierung wiederkehrender Abläufe.

5.8 Moonraker: API-Server für Mainsail

Moonraker fungiert als Backend von Mainsail und stellt die Schnittstelle zwischen der Weboberfläche und der Klipper-Firmware dar. Es ermöglicht die Datenübertragung zwischen Drucker und Steuerungsebene und bietet eine API für erweiterte Funktionen:

- Bereitstellung von Druckerstatus, Sensorwerten und Systeminformationen.
- Unterstützung von Push- oder E-Mail-Benachrichtigungen.
- Mehrbenutzerbetrieb zur gleichzeitigen Nutzung durch mehrere Clients.

5.9 Klipper Display: Lokale Steuerung von Klipper

Klipper Display stellt eine Benutzeroberfläche für den direkten Zugriff auf den 3D-Drucker bereit und ermöglicht grundlegende Steuerfunktionen ohne ein externes Webinterface.

- Anpassung von Druckparametern, Homing-Funktionen und Temperaturregelung.
- Anzeige von Druckerstatus, Druckfortschritt und Sensordaten.
- Ausführung vordefinierter Makros über das Display.

5.10 Konfiguration von Klipper

Die Konfiguration von Klipper erfolgt über eine textbasierte Konfigurationsdatei, die auf dem Steuerrechner (z. B. einem Raspberry Pi) gespeichert wird. Diese Datei enthält sämtliche Parameter zur Ansteuerung der Hardware und zur Definition des Druckeraufbaus.

5.10.1 Aufbau der Konfigurationsdatei

Die Konfigurationsdatei ist in Abschnitte gegliedert, die jeweils spezifische Hardwarekomponenten oder Funktionen definieren:

- **[mcu]**: Definiert das Mainboard und die zugehörige Kommunikationsschnittstelle.
- **[stepper_x], [stepper_y], [stepper_z]**: Konfiguration der Schrittmotoren, inklusive Pins, Schrittauflösung und maximaler Geschwindigkeit.
- **[extruder]**: Einstellungen für den Extruder, einschließlich Heizpatrone, Temperatursensor und Vorschubrate.
- **[heater_bed]**: Definition der Heizbettsteuerung mit PID-Regelung und Sicherheitsgrenzen.
- **[sensor]**: Anbindung von Temperatursensoren, Endschaltern und anderen Sensoren.

5.10.2 Anpassung der Konfiguration

Die Konfigurationsdatei kann direkt editiert werden, um Anpassungen an der Hardware vorzunehmen. Änderungen werden nach einem Neustart der Firmware wirksam, ohne dass eine erneute Kompilierung erforderlich ist. Dadurch lassen sich Druckerparameter flexibel anpassen und optimieren.

5.10.3 Makros und erweiterte Funktionen

Zusätzlich zu den grundlegenden Hardwareeinstellungen erlaubt Klipper die Definition von Makros zur Automatisierung wiederkehrender Befehlssequenzen. Diese werden in der Konfigurationsdatei unter **[gcode_macro]** definiert und können zur Optimierung von Druckprozessen genutzt werden. Beispiele dafür sind:

- Automatisiertes Homing und Kalibrierungsabläufe.
- Spezielle Start- und Endsequenzen für den Druckprozess.
- Steuerung zusätzlicher Komponenten wie Servos oder externe Sensoren.

5.10.4 Erweiterung der Funktionalität

Während viele Funktionen durch Makros realisiert werden können, sind für spezialisierte Anwendungen, wie die Steuerung eines Faserextruders, erweiterte Anpassungen erforderlich. Hierfür werden Plugins entwickelt, die neue Funktionen in Klippy integrieren (8.1.6 *Firmware*, S.203). In Fällen, in denen die Standardfunktionalität nicht ausreicht, sind direkte Modifikationen an Klippy notwendig.

5.11 Klipper-Plugin zur Steuerung des Schneidervos

Das Klipper-Plugin zur Steuerung des Schneidervos ermöglicht das präzise Abschneiden der Fasern während des Druckprozesses. Hierzu wird ein PWM-Signal genutzt, um den Servomotor auf eine definierte Winkelposition zu bewegen.

5.11.1 Konfiguration des Plugins

Die Konfiguration erfolgt in der Klipper-Konfigurationsdatei innerhalb des Abschnitts `[fiberextruder]`. Wie in Listing 2 zu sehen ist, werden Parameter wie der Initialwinkel, der Schneidwinkel sowie der zugeordnete Steuerpin festgelegt:

Listing 2: Beispielhafte Konfigurationsdatei für den Schneidervos

```

1 [fiberextruder]
2 initial_angle: 0.0
3 cutting_angle: 20.0
4 pin: !PB2

```

5.11.2 PWM-Steuerung des Servos

Der Servomotor wird mittels eines PWM-Signals angesteuert, wobei die Impulsweite des Signals die Winkelposition des Servos bestimmt. Das Plugin definiert hierzu folgende Parameter:

- `SERVO_SIGNAL_PERIOD` = 20 ms: Periodendauer des PWM-Signals.
- `MIN_WIDTH` = 0.5 ms: Minimale Pulsweite für die kleinste Servoposition.
- `MAX_WIDTH` = 2.51 ms: Maximale Pulsweite für die größte Servoposition.
- `ANGLE_SPEED` = 0.005: Geschwindigkeit der Winkelbewegung.

Die Umrechnung des Winkels in eine PWM-Pulsweite erfolgt nach Gleichung (9). Anschließend wird die resultierende Pulsweite mithilfe von Gleichung (10) in den entsprechenden PWM-Wert umgerechnet:

$$\text{width} = \text{MIN_WIDTH} + \text{angle} \times \left(\frac{\text{MAX_WIDTH} - \text{MIN_WIDTH}}{\text{max_angle}} \right) \quad (9)$$

$$\text{pwm_value} = \frac{\text{width}}{\text{SERVO_SIGNAL_PERIOD}} \quad (10)$$

In der in Tabelle 5.11.2 dargestellten Variablenbeschreibung werden die relevanten Variablen für die PWM-Steuerung zusammengefasst.

Variablenbeschreibung

- `angle`: Gewünschter Winkel des Servos.
- `max_angle`: Maximaler Winkel des Servos.
- `width`: Pulsweite des PWM-Signals.
- `pwm_value`: PWM-Wert zur Ansteuerung des Servos.

5.11.3 Implementierung des Plugins

Das Plugin registriert mehrere G-Code-Befehle zur Steuerung des Servos:

- **CUT**: Bewegt den Servo zum Schneidwinkel und zurück.
- **M751**: Alternative Bezeichnung für den **CUT**-Befehl.
- **CUT_STATUS**: Gibt den aktuellen Winkel des Servos aus.

Listing 3 zeigt die Implementierung der Funktion `cmd_CUT_SERVO`, welche den Servo auf den definierten Winkel setzt:

Listing 3: Python-Implementierung zur Servo-Steuerung

```

1 def cmd_CUT_SERVO(self, gcmd):
2     print_time = self.printer.lookup_object('toolhead').get_last_move_time()
3     angle = gcmd.get_float('ANGLE', None)
4
5     if angle is not None:
6         self._set_pwm(print_time, self._get_pwm_from_angle(angle))
7     else:
8         self._set_pwm(print_time, self._get_pwm_from_angle(self.initial_angle + self.cutting_angle))
9         self._set_pwm(print_time + self.cutting_angle * ANGLE_SPEED,
10                     self._get_pwm_from_angle(self.initial_angle))

```

5.12 Klipper-Plugin zur Steuerung des Faserextruders

Das Klipper-Plugin zur Steuerung des Faserextruders ermöglicht eine gezielte Extrusion der Fasern während des Druckprozesses. Die Extrusion wird in die Bewegungssteuerung integriert und kann durch spezifische G-Code-Befehle aktiviert oder deaktiviert werden.

5.12.1 Konfiguration des Faserextruders

Die Konfiguration erfolgt über den Abschnitt `[fiberextruder]` in der Klipper-Konfigurationsdatei. Listing 4 zeigt hierzu eine beispielhafte Konfiguration, in der unter anderem `e_factor` sowie die Stepper-Pins definiert werden:

Listing 4: Beispielhafte Konfigurationsdatei für den Faserextruder

```

1 [fiberextruder]
2 e_factor: 1.0
3
4 # Stepper configuration
5 step_pin: !PB0
6 dir_pin: !PB1
7 enable_pin: !PB2
8 rotation_distance: 1.0
9 microsteps: 16
10 ...

```

5.12.2 Integration in den Toolhead von Klippy

Zur Steuerung des Faserextruders werden Funktionszeiger (`callbacks`) verwendet, die Ereignisse wie `move`, `check_move` und `update_move_time` registrieren. In Listing 5 ist die Registrierung dieser Callback-Funktionen dargestellt:

Listing 5: Registrierung der Callback-Funktionen

```

1 self.printer.register_event_handler("move", self.move)
2 self.printer.register_event_handler("check_move", self.check_move)
3 self.printer.register_event_handler(
4     "update_move_time", self.update_move_time)

```

5.12.3 Bewegungssteuerung

Die Klasse `FiberExtruder` koordiniert die Extrusion basierend auf den Druckbewegungen. Hierbei kommt eine Trapezbewegungswarteschlange (`trapq`) zum Einsatz, die eine effiziente Verarbeitung der Bewegungsdaten sicherstellt. In Listing 6 wird deren Initialisierung veranschaulicht:

Listing 6: Initialisierung der Trapezbewegungswarteschlange

```

1 ffi_main, ffi_lib = chelper.get_ffi()
2 self.trapq_append = ffi_lib.trapq_append
3 self.trapq_finalize_moves = ffi_lib.trapq_finalize_moves
4 self.trapq = ffi_main.gc(ffi_lib.trapq_alloc(), ffi_lib.trapq_free)

```

Die Funktion `move` (siehe Listing 7) wird bei jeder Bewegungsausführung des Toolheads aufgerufen und entscheidet, ob eine Faserextrusion erforderlich ist:

Listing 7: Bewegungssteuerung in `move`

```

1 def move(self, print_time, move):
2     if self.enabled:
3         if move.is_kinematic_move:
4             d = move.move_d
5             axis_r = move.axes_r[3]
6             accel = move.accel
7             start_v = move.start_v
8             cruise_v = move.cruise_v
9
10            self.trapq_append(self.trapq, print_time, move.accel_t, move.cruise_t, move.decel_t,
11                           d, 0., 0., 1., False, 0., start_v, cruise_v, accel)

```

6 Slicing-Software

Die Slicing-Software spielt eine große Rolle im 3D-Druckprozess, indem sie digitale Modelle in maschinenlesbare Anweisungen umwandelt. Dabei werden nicht nur die einzelnen Druckschichten berechnet, sondern auch Parameter wie Druckgeschwindigkeit, Materialfluss und Temperatursteuerung festgelegt.

6.1 Einleitung

Die Entwicklung einer spezialisierten Slicing-Software für die Endlosfaserverlegung stellt besondere Herausforderungen dar. Herkömmliche Slicer sind auf die Verarbeitung von standardisierten Druckprozessen ausgelegt und bieten keine native Unterstützung für die gezielte Steuerung der Faserverlegung. Daher wird ein spezieller Slicer entwickelt, der alle relevanten Aspekte der Endlosfaserverarbeitung berücksichtigt und eine optimale Druckqualität gewährleistet.

6.2 Lösungsansätze

Zur Entwicklung eines spezialisierten Slicers für die Endlosfaser-Technologie gibt es mehrere mögliche Ansätze. Die beiden Varianten werden im Folgenden vorgestellt und miteinander verglichen:

6.2.1 Weiterentwicklung eines bestehenden Open-Source-Slicers

Eine Möglichkeit besteht darin, eine vorhandene Open-Source-Slicing-Software zu erweitern, um die Funktionalität der Faserverlegung zu integrieren. Dies kann durch die Entwicklung von Plugins oder Erweiterungen erfolgen.

Vorteile :

- Nutzung bestehender, erprobter Softwarearchitektur.
- Reduzierter Entwicklungsaufwand, da grundlegende Funktionen bereits vorhanden sind.
- Community-Support und vorhandene Dokumentation.

Nachteile :

- Eingeschränkte Kontrolle über die Architektur der Software.
- Mögliche Kompatibilitätsprobleme mit spezifischen Hardwareanforderungen.
- Erschwerter Optimierungsgrad durch bestehende Code-Strukturen.

6.2.2 Entwicklung eines Slicers von Grund auf ("Ground-up")

Eine alternative Herangehensweise ist die komplette Neuentwicklung eines Slicers, der von Anfang an auf die spezifischen Anforderungen des Endlosfaser-Drucks zugeschnitten ist.

Vorteile :

- Maximale Kontrolle über die Softwarearchitektur.
- Optimale Anpassung an die Hardware und spezifische Anforderungen.
- Hoher Lernfaktor durch die Auseinandersetzung mit allen relevanten Technologien.

Nachteile:

- Höherer Entwicklungsaufwand und längere Implementierungszeit.
- Keine direkte Nutzung bestehender Community-Lösungen.
- Erfordert umfassende Kenntnisse in verschiedenen Bereichen der Softwareentwicklung.

6.2.3 Vergleich der Ansätze

Beide Ansätze bieten spezifische Vorteile und Herausforderungen. Die Weiterentwicklung eines bestehenden Open-Source-Slicers ist effizienter und bietet schnelle Ergebnisse, während die Neuentwicklung mehr Kontrolle und Optimierungsmöglichkeiten bietet.

6.3 Gründe für die komplette Neuentwicklung eines Slicers

Die Entscheidung für die Neuentwicklung eines eigenen Slicers wird getroffen, um maximale Freiheit bei der Implementierung neuer Features zu haben und so viele Aspekte der Softwareentwicklung wie möglich zu erlernen. Bestehende Open-Source-Lösungen bieten zwar eine Basis, sind jedoch oft durch bestehende Strukturen limitiert und lassen nicht die Flexibilität zu, innovative Funktionen nahtlos zu integrieren.

6.4 Verwendung von Rust

Die Wahl der Programmiersprache für die Entwicklung eines neuen Slicers ist entscheidend für dessen Wartbarkeit und Stabilität. Rust bietet mehrere Aspekte, die es besonders geeignet für dieses Projekt machen.

6.4.1 Sicherheit und Fehlervermeidung

Rust ist eine Programmiersprache, die durch strenge Kompilierungsregeln häufige Fehler wie Speicherlecks und Datenrinnen verhindert. Sobald der Code erfolgreich kompiliert wurde, funktioniert er in den meisten Fällen korrekt. Diese Eigenschaft macht Rust besonders für Entwickler attraktiv, die auf Sicherheit und Stabilität großen Wert legen.

6.4.2 Leistung und Effizienz

Rust bietet eine Performance, die mit Sprachen wie C und C++ vergleichbar ist. Dadurch eignet sich Rust hervorragend für Systeme, bei denen Geschwindigkeit und Ressourcenoptimierung entscheidend sind. Sie kombiniert die Effizienz von Low-Level-Sprachen mit moderner Syntax und Sicherheitsmechanismen.

6.4.3 Lernen und moderne Programmierung

Neben technischer Effizienz bietet Rust auch die Gelegenheit, sich mit einer modernen, stark typisierten und sicherheitsorientierten Programmiersprache vertraut zu machen. Dies macht Rust nicht nur zu einer leistungsstarken Wahl für Projekte, sondern auch zu einer lohnenden Investition in die eigene Weiterentwicklung als Entwickler.

6.5 Entwicklung mit Rust

Dieses Kapitel behandelt die Entwicklung mit Rust, die Wahl der Entwicklungsumgebung, den Paketmanager Cargo und die Nutzung von Crates. Zudem werden Beispiele gezeigt, wie Projekte mit Cargo und Rust aufgesetzt werden.

6.5.1 Entwicklungsumgebungen

Für die Entwicklung mit Rust gibt es mehrere integrierte Entwicklungsumgebungen (IDEs) und Editoren, die Unterstützung bieten:

- Visual Studio Code (VS Code)
- RustRover Jetbrains
- Zed

Visual Studio Code wird als Entwicklungsumgebung gewählt, da es eine äußerst vielseitige und anpassbare IDE ist. Dank der großen Auswahl an Plugins und Erweiterungen kann die Funktionalität gezielt erweitert werden. Insbesondere die Rust-Analyzer-Erweiterung bietet eine umfassende Unterstützung für die Programmiersprache Rust. Sie umfasst unter anderem Autovervollständigung, Fehlerhervorhebung und Code-Navigation, wodurch die Entwicklung in Rust effizienter und benutzerfreundlicher gestaltet wird.

6.5.2 Rust-Bibliotheken und der Paketmanager Cargo

Rust verwendet *Cargo* als seinen Standard-Paketmanager und Build-System. Cargo ist eine essentielle Komponente des Rust-Ökosystems und wird verwendet, um Projekte zu erstellen, Abhängigkeiten zu verwalten und den Code zu kompilieren. Sobald ein Rust-Projekt gestartet wird, erstellt Cargo eine grundlegende Verzeichnisstruktur und eine `Cargo.toml`-Datei, die als zentrales Konfigurationsfile dient. In Rust werden Bibliotheken und Pakete als *Crates* bezeichnet. Eine *Crate* ist die grundlegende Einheit für Kompilierung und Verteilung. Es gibt zwei Arten von Crates:

- Binary Crates: Diese erzeugen ausführbare Programme und enthalten immer eine `main`-Funktion.
- Library Crates: Diese bieten Funktionen, Typen und andere Elemente, die in anderen Crates verwendet werden können. Sie enthalten keine `main`-Funktion.

Wie funktionieren Crates?

Crates werden in Rust über das öffentliche Repository (Developers, 2014a) verteilt. Entwickler können hier veröffentlichte Crates durchsuchen und ihre Abhängigkeiten in die `Cargo.toml`-Datei ihres Projekts integrieren. Die `Cargo.toml`-Datei enthält eine Liste der Abhängigkeiten, deren Versionen und weitere Metadaten. Ein einfaches Beispiel ist in Listing 8 zu sehen:

Listing 8: cargo.toml

```
1 [package]
2 name = "mylib"
3 version = "0.1.0"
4 edition = "2024"
```

Nach dem Hinzufügen von Abhängigkeiten lädt Cargo automatisch die benötigten Crates von *crates.io* (Developers, 2014a) herunter und integriert sie in das Projekt. Dies wird durch den Befehl `cargo build` oder `cargo check` ausgelöst.

Ein Crate wird durch ein eigenes `lib.rs` (für Libraries) oder `main.rs` (für Binaries) im `src`-Verzeichnis definiert. Listing 9 zeigt als Beispiel eine Library-Crate mit einer `greet`-Funktion:

Listing 9: lib.rs

```
1 pub fn greet(name: &str) -> String {
2     format!("Hello, {}!", name)
3 }
```

Diese Funktion könnte von einer anderen Crate importiert und verwendet werden, indem sie die entsprechende Abhängigkeit in der `Cargo.toml` definiert. In Listing 10 wird gezeigt, wie die Funktion aus `mylib` in einer `main.rs`-Datei verwendet wird:

Listing 10: main.rs

```
1 use mylib::greet;
2
3 fn main() {
4     let message = greet("Jan");
5     println!("{}", message);
6 }
```

Vorteile des Crate-Systems

- Wiederverwendbarkeit: Rust fördert die modulare Entwicklung durch die klare Trennung von Funktionalitäten in Crates.
- Versionierung und Sicherheit: Durch die Integration in Cargo und *crates.io* (Developers, 2014a) wird eine einfache Versionierung und das Prüfen auf potenzielle Sicherheitsprobleme ermöglicht.
- Open-Source-Community: Die meisten Crates auf *crates.io* (Developers, 2014a) sind Open Source, was die Zusammenarbeit und den Wissensaustausch fördert.

6.5.3 Rust-Book und Dokumentation

Die offizielle Dokumentation für Rust ist das sogenannte *Rust Book* (Developers, 2014b), welches kostenfrei verfügbar ist. Dieses umfassende Werk richtet sich sowohl an Einsteiger als auch an fortgeschrittene Rust-Entwickler und deckt alle wichtigen Aspekte der Sprache ab, von grundlegenden Syntax- und Typkonzepten bis hin zu komplexen Themen wie Concurrency und Memory Safety. Neben dem Rust Book existieren weitere offizielle Dokumentationen wie *The Rust Reference* (Developers, 2015) und *Rust by Example* (Developers, 2013), die jeweils unterschiedliche Schwerpunkte setzen:

- *The Rust Reference*: Ein detailliertes Nachschlagewerk für die Spezifikation und das Verhalten der Sprache.
- *Rust by Example*: Eine Sammlung interaktiver Beispiele, die typische Rust-Programme Schritt für Schritt aufschlüsseln und erläutern.

6.6 Bibliotheken

In diesem Abschnitt werden die verschiedenen Bibliotheken und Crates vorgestellt, die in diesem Projekt verwendet werden. Diese umfassen Bibliotheken für die Benutzeroberfläche, Rendering, mathematische Berechnungen, Fenster- und Eingabeverwaltung, asynchrone Verarbeitung sowie weitere spezifische Funktionen. Die Auswahl der Bibliotheken basiert auf Kriterien wie Plattformunabhängigkeit, Performance, einfache Integration und aktive Community-Unterstützung.

6.6.1 Benutzeroberfläche

Die Anforderungen an die Benutzeroberfläche eines Slicers umfassen eine intuitive Bedienung, Visualisierung der 3D-Modelle, Anpassungsmöglichkeiten für Slicing-Parameter und gezielte Steuerung der Faserverlegung. In der Praxis gibt es zwei wesentliche Ansätze, die je nach Projektziel und Zielgruppe gewählt werden können:

- Desktop-Anwendungen: Desktop-basierte UIs bieten eine native Nutzererfahrung und direkten Zugriff auf Systemressourcen. Diese Option eignet sich besonders für performante Anwendungen, die lokale Hardware ansteuern, wie ein Slicer für 3D-Druck.
- Webbasierte Benutzeroberflächen: Eine Web-UI mit Technologien wie WebAssembly (Wasm) ermöglicht eine plattformunabhängige Nutzung. Dieser Ansatz ist ideal für cloudbasierte Lösungen, bei denen der Benutzer ohne Installation auf die Software zugreifen kann.

Crates zur UI-Entwicklung

Die Wahl der richtigen Library ist entscheidend für Qualität der Software. Rust bietet verschiedene Libraries für die Entwicklung von Benutzeroberflächen, die unterschiedliche Stärken und Schwächen aufweisen. Wie in Tabelle 20 zu sehen ist, unterscheiden sie sich vor allem in Plattformunterstützung, Komplexität und Reifegrad:

- *Tauri*: Eine übermäßig komplexe Lösung, die Rust-Backends mit Webtechnologien kombiniert. Ideal für plattformübergreifende Anwendungen mit moderner UI.
- *Iced*: Deklaratives Framework für Desktop- und Web-UIs, einfach zu bedienen, aber eingeschränkt bei komplexen Layouts.
- *egui*: Leichtgewichtige Library für schnelles Prototyping und einfache UIs.

Tabelle 20: Vergleich der Libraries zur UI-Entwicklung in Rust

Library	Plattformen	Rust-Nativ	Komplexität der UIs	Reifegrad
Tauri	Desktop	Nein	Hoch	Mittel
Iced	Desktop, Web	Ja	Gering	Mittel
egui	Desktop, Web	Ja	Gering	Sehr Hoch

Warum egui?

Für die Entwicklung der Benutzeroberfläche wird `egui` (emilk, 2020) gewählt. Diese Entscheidung basiert auf mehreren Faktoren:

- Aktive Community: `egui` verfügt über eine engagierte und stetig wachsende Entwicklergemeinschaft, die regelmäßige Erweiterungen und Unterstützung bereitstellt.
- Reifegrad: `egui` ist eine ausgereifte Bibliothek mit Stabilität, die sich für schnelles Prototyping eignet.
- Einfache Integration: Die nahtlose Einbindung in Rust-Projekte erleichtert die Entwicklung und reduziert den Aufwand für Konfiguration und Setup.

6.6.2 Rendering

Für die Visualisierung der Werkzeugpfade und CAD-Modelle wird `wgpu` verwendet, eine moderne und plattformunabhängige API für Grafik-Rendering. Sie stellt eine sichere und leistungsfähige Schnittstelle zu aktuellen Grafik-Backends wie Vulkan, DirectX und Metal bereit und ermöglicht eine effiziente Nutzung der Hardware-Ressourcen.

Vergleich von Rendering-APIs

Wie in Tabelle 21 dargestellt, bietet `wgpu` ein hohes Maß an Hardware-Nähe und Flexibilität:

Tabelle 21: Vergleich gängiger Rendering-APIs

Library	Plattformen	Rust-Nativ	Hardware-Nähe	Flexibilität
bevy	Multi-Plattform	Ja	Niedrig	Hoch
rend3d	Multi-Plattform	Ja	Mittel	Mittel
three-d	Desktop	Ja	Mittel	Niedrig
wgpu	Multi-Plattform	Ja	Hoch	Sehr Hoch

Warum `wgpu`?

Die Entscheidung für `wgpu` (gfx-rs, 2019) basiert auf mehreren Faktoren. Es stellt eine moderne Alternative zu älteren Rendering-APIs dar und abstrahiert plattformübergreifend auf Vulkan, DirectX, Metal und OpenGL. Zu den wesentlichen Vorteilen gehören:

- Plattformunabhängigkeit: Funktioniert auf Windows, Linux, macOS und WebAssembly.
- Einfache Integration: Leicht mit `egui` und anderen Grafikbibliotheken kombinierbar.
- Hohe Performance: Ideal für performante Visualisierungen im Slicer.

6.6.3 Mathematik und Vektorrechnungen

Für Berechnungen in 3D kommt `glam` (Hart, 2022) zum Einsatz, eine hochoptimierte Mathebibliothek für lineare Algebra in Rust.

Warum `glam`?

- Schnelle Vektorrechnungen: SIMD-Optimierungen für hohe Performance.
- Einfache API: Intuitive Nutzung für Transformationen und Geometrie.
- Breite Anwendung: Unterstützt 2D, 3D und 4D Vektoren sowie Matrizen.

6.6.4 Fenster- und Eingabeverwaltung

Die Verwaltung von Fenstern und Benutzereingaben erfolgt über `winit` (Source, 2015), eine robuste Bibliothek für plattformunabhängige GUI-Anwendungen.

Warum `winit`?

- Plattformunabhängig: Unterstützt Windows, macOS, Linux und WebAssembly.
- Eventbasiert: Reagiert effizient auf Eingaben und Fensterereignisse.
- Gut integriert: Funktioniert nahtlos mit egui und wgpu.

6.6.5 Asynchrone Verarbeitung

Zur Verwaltung asynchroner Operationen kommt `tokio` (Source, 2016) zum Einsatz.

Warum `tokio`?

- Effizient: Hochperformante asynchrone Laufzeitumgebung.
- Skalierbar: Unterstützt parallele Verarbeitung von Netzwerk- und IO-Operationen.
- Etablierte Lösung: Eine der am weitesten verbreiteten async-Bibliotheken in Rust.

6.6.6 Weitere verwendete Crates

Neben den großen Kernbibliotheken egui, wgpu, glam, winit und tokio kommen weitere Bibliotheken zum Einsatz, die spezifische Funktionen im Projekt übernehmen.

- `serde` (1.0.217), `serde_json` (1.0.134), `serde_yaml` (0.9.25): Diese Bibliotheken ermöglichen das effiziente Serialisieren und Deserialisieren von Datenformaten wie JSON und YAML, was für die Konfigurationsverwaltung hilfreich ist.
- `toml` (0.8.19): Dient zur Verarbeitung von TOML-Dateien für Konfigurationszwecke, insbesondere für benutzerdefinierte Einstellungen und Parameter.
- `thiserror` (2.0.9), `anyhow` (1.0): Diese Bibliotheken erleichtern die Fehlerbehandlung in Rust, indem sie eine einfache und einheitliche Schnittstelle für Fehlermanagement bieten.
- `log` (0.4.21), `simple-logging` (2.0.2), `env_logger` (0.11.6): Logging-Frameworks, die zur Fehlerdiagnose und Analyse der Laufzeitperformance des Programms verwendet werden.
- `rayon` (1.10.0): Ermöglicht parallele Datenverarbeitung mittels Multithreading und verbessert die Performance rechenintensiver Prozesse.
- `parking_lot` (0.12.3): Eine leistungsfähige Alternative zu den Standard-Locking-Mechanismen in Rust für effizientere Synchronisation im Multithreading.
- `stl_io` (0.8.3): Diese Bibliothek erlaubt das Einlesen und Schreiben von STL-Dateien.
- `uni-path` (1.51.1): Erleichtert die Arbeit mit Dateipfaden, um plattformübergreifende Dateioperationen durchzuführen.
- `phf` (0.11): Eine Bibliothek zur Erstellung statischer Hash-Maps, die für schnelle und speichereffiziente Datenabfragen genutzt wird.
- `native-dialog` (0.7.0): Wird für native Dialogfenster und Dateiauswahl-Interfaces genutzt, um die Benutzerfreundlichkeit der Anwendung zu verbessern.
- `puffin` (0.19.0), `puffin_http` (0.16.0), `puffin_egui`: Diese Bibliotheken dienen dem Profiling und der Echtzeit-Analyse der Softwareperformance und können direkt in der Benutzeroberfläche visualisiert werden.
- `futures-channel` (0.3.31): Bietet Mechanismen für asynchrone Nachrichtenübermittlung zwischen Threads, was für Event-basierte Kommunikation nützlich ist.
- `egui_code_editor` (0.2.11): Integriert einen Code-Editor in die Benutzeroberfläche, was die Bearbeitung von Skripten und Einstellungen innerhalb des Programms ermöglicht.
- `ordered-float` (4.6.0): Ermöglicht das Sortieren und Vergleichen von Gleitkommazahlen, indem spezielle Mechanismen zur Vermeidung von Rundungsfehlern genutzt werden.
- `atomic_float` (1.1.0): Erlaubt atomare Operationen mit Gleitkommazahlen, was insbesondere für Multithreading-Szenarien hilfreich ist.
- `egui-toast`: Erweiterung für egui, die Benachrichtigungen und Toast-Meldungen innerhalb der Benutzeroberfläche ermöglicht.
- `egui_xml`: Ist ein selbst entwickeltes Crate, welches es ermöglicht dynamische Layouts mit XML-Format zu erstellen.

6.7 Softwarearchitektur

Die vorgestellte Software besitzt eine modulare Struktur, die sich in drei zentrale *Crates* unterteilt. Diese Aufteilung dient dazu, Verantwortlichkeiten klar zu trennen und die Wartung langfristig zu vereinfachen:

- Applikation Crate: Enthält die Hauptanwendung inklusive Rendering, Picking und Benutzeroberfläche.
- Slicer Crate: Implementiert die Slicing-Algorithmen und stellt die G-Code-Generierung bereit.
- Shared Crate: Definiert gemeinsame Datentypen, die eine reibungslose Kommunikation zwischen den Modulen ermöglichen.

6.7.1 Start der Applikation

Der Einstiegspunkt der Anwendung wird durch die `main`-Funktion gebildet, in der eine Konfigurationsdatei geladen, ein *Event-Loop* erstellt und wichtige Dienste initialisiert werden. Listing 11 zeigt exemplarisch den Ablauf des Programmstarts.

Listing 11: Start der Applikation

```

1  #[tokio::main]
2  async fn main() -> Result<(), EventLoopError> {
3      load_config();
4
5      // Start the Puffin server for profiling
6      ...
7
8      // Initialize the logger
9      ...
10
11     let event_loop: EventLoop<RootEvent> = EventLoop::with_user_event()
12         .build().unwrap();
13
14     event_loop.set_control_flow(winit::event_loop::ControlFlow::Wait);
15
16     let mut application = Application {
17         proxy: event_loop.create_proxy(),
18         state: None,
19     };
20
21     event_loop.run_app(&mut application)
22 }
```

6.7.2 Event-Loop und Steuerung der Applikation

Ein zentrales Element der Software stellt der *Event-Loop* dar, der in der **Application**-Struktur gekapselt ist. Diese Struktur verwaltet den globalen Zustand der Anwendung und reagiert auf sämtliche eingehenden Ereignisse wie Benutzerinteraktionen oder Aktualisierungsanfragen. Listing 12 illustriert den Aufbau der **Application**-Struktur mit den relevantesten Ereignis-Handlern.

Listing 12: Die **Application**-Struktur mit Event-Loop-Implementierung

```

1 impl ApplicationHandler<RootEvent> for Application {
2     fn resumed(
3         ...
4     ) {
5         // Wird automatisch ausgefuehrt, wenn eine Neuinitialisierung notwendig ist.
6     }
7
8     fn window_event(
9         ...
10    ) {
11        // Verarbeitung von Fensterereignissen
12    }
13
14     fn device_event(
15         ...
16    ) {
17        // Verarbeitung von Eingabegeraete-Ereignissen
18    }
19
20     ...
21 }
```

6.7.3 Initialisierung der Module

Die in Listing 12 gezeigte `resumed`-Funktion wird bei Bedarf von der Bibliothek `winit` automatisch aufgerufen (etwa nach einem Wechsel zwischen Fenstern oder bei Änderung des verwendeten Grafikgeräts). In dieser Methode werden sämtliche Module über das **Adapter**-Trait erzeugt und in die Anwendung eingebunden. Dieser Ansatz stellt sicher, dass eine einheitliche und erweiterbare Verwaltung der einzelnen Teilsysteme gewährleistet ist.

Listing 13: Die `resumed`-Funktion mit der Initialisierung der Module

```

1 fn resumed(&mut self, event_loop: &winit::event_loop::ActiveEventLoop) {
2     let window = Arc::new(window::create_window(event_loop)
3         .expect("Failed to create window"));
4
5     let wgpu_context = WgpuContext::new(window.clone()).unwrap();
6
7     let (_, _, render_adapter) = render::RenderAdapter::create(&wgpu_context);
8     let (_, camera_event_writer, mut camera_adapter) =
9         viewer::CameraAdapter::create(&wgpu_context);
10    let (picking_state, picking_event_writer, picking_adapter) =
11        input::InputAdapter::create(&wgpu_context);
12    let (ui_state, ui_event_writer, ui_adapter) =
13        ui::UiAdapter::create(&wgpu_context);
14
15    self.state = Some(ApplicationState {
16        window,
17        wgpu_context,
18        render_adapter,
19        camera_adapter,
20        picking_adapter,
21        ui_adapter,
22        start_time: Instant::now(),
23    });
24 }
```

6.7.4 Erstellung der Module mit dem Adapter-Trait

Das **Adapter-Trait** definiert den einheitlichen Mechanismus, mit dem jedes Subsystem (z. B. Rendering, UI, Picking) erzeugt und im *Event-Loop* registriert wird. Auf diese Weise lässt sich jedes Modul konsistent verwalten und bei Bedarf einfach erweitern oder austauschen. Konkret stellt das **Adapter-Trait** sicher, dass:

- Alle nötigen Initialisierungsschritte ausgeführt und korrekt in die Anwendung eingebunden werden.
- Ein periodisches Update jedes Moduls gewährleistet ist.
- Eine gemeinsame Schnittstelle für die Reaktion auf Eingaben und Ereignisse existiert.

Diese modulare Herangehensweise ermöglicht eine saubere Trennung der Zuständigkeiten zwischen den Bereichen Rendering, Slicing und gemeinsamer Datenhaltung. Dadurch bleibt der Code leichter verständlich und wartbar, während gleichzeitig eine Erweiterbarkeit gewährleistet ist.

6.8 Benutzeroberflächen

Der Benutzeroberflächen-Adapter (**UiAdapter**) ist verantwortlich für die Darstellung und Interaktion der Anwendung über `egui`. Er verarbeitet UI-Ereignisse, verwaltet den Zustand der Benutzeroberfläche und ermöglicht eine dynamische Aktualisierung des Interfaces basierend auf Nutzereingaben.

6.8.1 Funktionalität des **UiAdapter**

Der **UiAdapter** basiert auf dem *Adapter-Trait* und integriert sich nahtlos in den Event-Loop der Applikation. Seine Hauptaufgaben sind:

- Initialisierung und Verwaltung der `egui`-Plattform.
- Rendering der Benutzeroberfläche inklusive Layout und Komponenten.
- Verarbeitung von UI-Ereignissen wie Button-Klicks, Toast-Nachrichten und Benutzerinteraktionen.
- Verwaltung von UI-Themes (`Light` und `Dark`).

6.8.2 Struktur des **UiAdapter**

Der **UiAdapter** besitzt folgende Hauptkomponenten:

- **UiState** – Verwaltet den Zustand der UI, darunter Themes und Interaktionsflags.
- **Platform** – Die Schnittstelle zur `egui`-Plattform.
- **Screen** – Organisiert die Darstellung der UI-Komponenten.
- **EventReader<UiEvent>** – Handhabt und verarbeitet UI-bezogene Ereignisse.

6.8.3 Die Implementierung des **UiAdapter**

Die folgende Implementierung zeigt, wie der **UiAdapter** aufgebaut ist (siehe Listing 14):

Listing 14: Die Struktur des **UiAdapter**

```

1 pub struct UiAdapter {
2     state: UiState,
3     screen: Screen,
4     platform: Platform,
5     event_reader: EventReader<UiEvent>,
6 }
```

Die Erstellung des **UiAdapter** erfolgt über die Methode `create()`, die alle notwendigen Ressourcen und Events initialisiert (siehe Listing 15):

Listing 15: Erstellung des UiAdapter

```

1 fn create(context: &WgpuContext) -> AdapterCreation<UiState, UiEvent, Self> {
2     let platform = Platform::new(PlatformDescriptor {
3         ...
4     });
5
6     let state = UiState::default();
7     let screen = Screen::new();
8     let (reader, writer) = create_event_bundle::<UiEvent>();
9
10    (
11        state.clone(),
12        writer,
13        Self {
14            state,
15            screen,
16            platform,
17            event_reader: reader,
18        },
19    )
20 }
```

6.8.4 Event-Verarbeitung im UiAdapter

Der **UiAdapter** verarbeitet UI-bezogene Ereignisse wie Informationsmeldungen, Erfolgsmeldungen und Fehlernachrichten. Diese Ereignisse werden im **UiEvent**-Enum definiert (siehe Listing 16) und anschließend verarbeitet (siehe Listing 17):

Listing 16: Definition der UI-Ereignisse

```

1 #[derive(Debug, Clone)]
2 pub enum UiEvent {
3     ShowInfo(String),
4     ShowSuccess(String),
5     ShowError(String),
6     ShowProgressBar(u32, String),
7     GCodeReaderLookAt(usize),
8 }
```

Listing 17: Verarbeitung der UI-Ereignisse

```

1 fn handle_event(
2     &mut self,
3     wgpu_context: &WgpuContext,
4     _global_state: &GlobalState<RootEvent>,
5     event: UiEvent,
6 ) {
7     match event {
8         UiEvent::ShowInfo(message) => {
9             // Show info message
10            wgpu_context.window.request_redraw();
11        }
12        ...
13    }
14 }
```

6.8.5 Rendering der UI

Der **UiAdapter** ist außerdem für das Rendering der UI verantwortlich, indem er das **handle_frame()**-Trait implementiert (siehe Listing 18). Die UI-Komponenten werden hier in einem separaten Rendering-Pass aktualisiert:

Listing 18: Rendering der Benutzeroberfläche

```

1 fn handle_frame(
2     ...
3 ) -> Result<(UiUpdateOutput, Viewport), Error> {
4     self.platform.begin_pass();
5
6     self.screen.show(
7         &self.platform.context(),
8         &(self.state.clone(), global_state),
9     );
10
11    let full_output = self.platform.end_pass(Some(&wgpu_context.window));
12
13    let viewport = self.screen.construct_viewport(wgpu_context);
14
15    let paint_jobs = ...;
16
17    let tdelta: egui::TexturesDelta = full_output.textures_delta;
18
19    let screen_descriptor = ScreenDescriptor {
20        ...
21    };
22
23    if self.platform.context().has_requested_repaint() {
24        wgpu_context.window.request_redraw();
25    }
26
27    Ok((
28        UiUpdateOutput {
29            paint_jobs,
30            tdelta,
31            screen_descriptor,
32        },
33        viewport,
34    ))
35 }
```

6.9 Benutzeroberflächen-Komponenten

Die Benutzeroberfläche der Anwendung besteht aus mehreren modularen Komponenten, die mit der **Screen**-Struktur (Listing 19) organisiert sind. Diese Komponenten umfassen Menüleisten, Werkzeugleisten, Seitenleisten und weitere Interaktionselemente, die eine benutzerfreundliche Steuerung ermöglichen.

Listing 19: Screen Definition

```

1 pub struct Screen {
2     toasts: Toasts,
3     toasts_progress_bar: Toasts,
4     tools: tools::Tools,
5     addons_state: addons::AddonsState,
6     settings_state: sidebar::SidebarState,
7     menubar_state: MenubarState,
8     taskbar_state: TaskbarState,
9     modebar_state: ModebarState,
10    toolbar_state: ToolBarState,
11    topbar_state: TopBarState,
12 }
```

6.9.1 Anzeige der Benutzeroberfläche

Die `show()`-Methode eines **Screen**-Objekts (siehe Listing 20) wird aufgerufen, um die Benutzeroberfläche zu rendern und die einzelnen UI-Komponenten darzustellen. Dabei wird ein Layout erstellt, das die verschiedenen Steuerleisten und Werkzeuge organisiert:

Listing 20: Anzeige der UI-Komponenten

```

1 pub fn show(&mut self, ctx: &egui::Context, shared_state: &(UiState, GlobalState)) {
2     menubar::Menubar::with_state(&mut self.menubar_state)
3         .with_component_states(&mut [
4             ...
5         ])
6         .show(ctx, shared_state);
7
8     topbar::Topbar::with_state(&mut self.topbar_state).show(ctx, shared_state);
9     taskbar::Taskbar::with_state(&mut self.taskbar_state).show(ctx, shared_state);
10    sidebar::Settingsbar::with_state(&mut self.settings_state).show(ctx, shared_state);
11    modebar::Modebar::with_state(&mut self.modebar_state).show(ctx, shared_state);
12    toolbar::Toolbar::with_state(&mut self.toolbar_state).show(ctx, shared_state);
13
14    egui::CentralPanel::default().frame(frame).show(ctx, |ui| {
15        self.toasts.show_inside(ui);
16        self.toasts_progress_bar.show_inside(ui);
17        addons::Addons::with_state(&mut self.addons_state).show(ui, shared_state);
18        self.tools.show(ctx, shared_state);
19    });
20
21 }
```

6.9.2 Verarbeitung von Toast-Nachrichten

Das Screen-Objekt verarbeitet Toast-Benachrichtigungen, um Benutzer über verschiedene Ereignisse zu informieren. Diese Toasts werden in separaten Bereichen der UI angezeigt (siehe Listing 21):

Listing 21: Hinzufügen von Toast-Benachrichtigungen

```

1 pub fn add_toast(&mut self, toast: egui_toast::Toast) {
2     self.toasts.add(toast);
3 }
4
5 pub fn add_process_as_toast(&mut self, toast: egui_toast::Toast) {
6     self.toasts_progress_bar.add(toast);
7 }
```

6.9.3 Viewport-Berechnung für das Rendering

Das `construct_viewport()`-Verfahren berechnet die Position und Größe des Viewports basierend auf den UI-Elementen und sorgt dafür, dass der verfügbare Bildschirmplatz optimal genutzt wird (Listing 22):

Listing 22: Viewport-Kalkulation

```

1 pub fn construct_viewport(&self, wgpu_context: &WgpuContext) -> (f32, f32, f32, f32) {
2     let height = wgpu_context.surface_config.height as f32
3         - self.taskbar_state.get_boundary().get_height()
4         - self.modebar_state.get_boundary().get_height()
5         - self.menubar_state.get_boundary().get_height();
6
7     (
8         self.settings_state.get_boundary().get_width(),
9         self.taskbar_state.get_boundary().get_height()
10            + self.modebar_state.get_boundary().get_height(),
11         wgpu_context.surface_config.width as f32
12             - self.toolbar_state.get_boundary().get_width()
13             - self.settings_state.get_boundary().get_width(),
14         height,
15     )
16 }
```

6.9.4 Interaktion mit Werkzeugen und Addons

Die Benutzeroberfläche der Anwendung ist in verschiedene Kategorien unterteilt, um eine einfache Erweiterbarkeit zu gewährleisten:

- **Tools:** Werkzeuge, die über die Toolbar auf der rechten Seite ein- und ausgeschaltet werden können. Diese erscheinen als eigene Fenster innerhalb der Anwendung.
- **Komponenten:** UI-Elemente, die fest in das Layout integriert sind, wie z. B. die Menüleiste oder die Seitenleiste.
- **Addons:** Funktionen, die direkt in der 3D-Ansicht sichtbar sind, beispielsweise Transformationen wie Verschieben, Skalieren oder Rotieren.
- **Widgets:** Wiederverwendbare UI-Bausteine, die innerhalb mehrerer Komponenten verwendet werden.

6.9.5 Tools (Steuerbare UI-Fenster)

Die Tools sind interaktive Fenster, die innerhalb der Anwendung gerendert werden und spezifische Funktionen bereitstellen. Sie sind über die Toolbar aktivierbar und beinhalten beispielsweise:

- **Debug-Tool:** Ermöglicht das Analysieren interner Prozesse und das Debuggen von Anwendungskomponenten.
- **Explorer:** Bietet eine Übersicht über geladene Objekte.
- **GCode-Anzeige:** Stellt G-Code-Dateien effizient dar und ermöglicht eine flüssige Navigation durch große Dateien.
- **Sichtbarkeits-Tool:** Ermöglicht das Ein- und Ausblenden von Werkzeugpfaden.

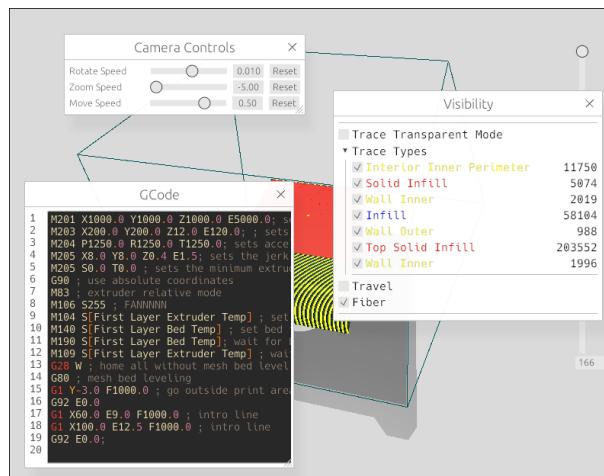


Abb. 114: Werkzeuge

Abbildung 114 zeigt das Werkzeug-Fenster, in dem die verschiedenen Tools angezeigt und ein- bzw. ausgeschaltet werden können.

6.9.6 Komponenten (UI-Strukturelemente)

Komponenten sind fest in die Benutzeroberfläche integriert und stellen verschiedene Steuer- und Verwaltungsbereiche dar:

- **Menüleiste:** Enthält übergeordnete Steuerfunktionen der Anwendung.
- **Seitenleiste:** Bietet Konfigurations- und Einstellungsoptionen.
- **Modusleiste:** Zeigt den aktuellen Betriebsmodus der Anwendung an.
- **Taskleiste:** Listet laufende Prozesse und Hintergrundaufgaben auf.
- **Werkzeugeleiste:** Stellt die wichtigsten Werkzeuge für den Arbeitsprozess bereit.
- **Oberste Steuerleiste:** Eine zusätzliche Steuerleiste zur Verwaltung von Funktionen.

6.9.7 Addons (3D-Interaktionselemente)

Addons sind Funktionen, die direkt in der 3D-Ansicht dargestellt und genutzt werden. Sie beeinflussen die Manipulation von Objekten und sind eng mit der 3D-Engine verbunden:

- Transformations-Werkzeuge: Ermöglichen das Verschieben, Skalieren und Rotieren von Objekten.
- CAD-Tools: Spezialisierte Werkzeuge zur Konstruktion innerhalb der 3D-Ansicht.



Abb. 115: Addons

Abbildung 115 illustriert Addons, die für die 3D-Ansicht zur Verfügung stehen.

6.10 Transformationen und Manipulationen für 3D-Objekte

Die 3D-Transformationen und Manipulationen ermöglichen die Bearbeitung von Objekten in der Anwendung. Dazu gehören das Verschieben, Rotieren und Skalieren, die über interaktive Werkzeuge realisiert werden. Die Implementierung erfolgt über ein **Gizmo-Tool**, das in der 3D-Ansicht sichtbar ist und mit den Objekten direkt interagiert.

Listing 23: Definition der Gizmo-Transformationen

```

1 #[derive(Debug, Clone, Copy, PartialEq, Eq, EnumIter, EnumCount)]
2 pub enum GizmoTool {
3     Translate,
4     Rotate,
5     Scale,
6     Flatten,
7 }
```

Diese Werkzeuge werden als Addons dargestellt und können vom Nutzer ausgewählt werden. Die Auswahl wird durch die **GizmoTools**-Struktur verwaltet (siehe Listing 24):

Listing 24: Verwaltung der Werkzeuge

```

1 #[derive(Debug, Default)]
2 pub struct GizmoTools {
3     selected: Option<GizmoTool>,
4 }
```

6.10.1 Darstellung der Transformationswerkzeuge

Die Werkzeuge werden über Icons in einer vertikalen Leiste dargestellt (Listing 25). Die Icons sind klickbar und zeigen beim Überfahren eine Tooltip-Beschreibung des jeweiligen Werkzeugs.

Listing 25: Darstellung der Werkzeuge mit Icons

```

1 pub fn show_icons(&mut self, ui: &mut egui::Ui, _shared_state: &(UiState, GlobalState)) {
2     ui.vertical(|ui| {
3         let mut builder = GridBuilder::new();
4         for _ in 0..GizmoTool::COUNT {
5             builder = builder.new_row(Size::remainder());
6         }
7
8         builder.show(ui, |mut grid| {
9             for (tool, (name, _)) in GizmoTool::iter().zip(GIZMO_TOOL_LABELS.iter()) {
10                 grid.cell(|ui| {
11                     let image_button = ImageButton::new(get_gizmo_tool_icon(tool))
12                         .rounding(5.0)
13                         .selected(self.selected == Some(tool))
14                         .frame(true);
15
16                     let response = ui.add(image_button);
17                     if response.clicked() {
18                         self.selected = Some(tool);
19                     } else if response.hovered() {
20                         ...
21                     }
22                 });
23             });
24         });
25     });
26 }
```

Abbildung 116 zeigt die Symbolleiste mit den Transformationswerkzeugen.



Abb. 116: Transformations Werkzeuge

6.10.2 Transformation von 3D-Objekten

Die gewählte Transformation wird innerhalb eines eigenen Fensters durchgeführt, das sich öffnet, sobald ein Werkzeug aktiviert wird (siehe Listing 26). Je nach Auswahl werden unterschiedliche Parameter angepasst:

- Verschieben (Translate): Veränderung der Position entlang der X-, Y- und Z-Achse.
- Rotieren (Rotate): Drehung um die drei Achsen mit Anpassung über Drag-Elemente.
- Skalieren (Scale): Veränderung der Größe entlang der drei Achsen mit einer Mindest- und Maximalbegrenzung.
- Abflachen (Flatten): Eine spezifische Funktion zur Anpassung von Objektgeometrien.

Listing 26: Transformationen je nach Werkzeug

```

1  match tool {
2      GizmoTool::Translate => {
3          let mut changed = false;
4          ui.horizontal(|ui| {
5              changed |= ui.add(DragValue::new(&mut translation.x).max_decimals(3)).changed();
6              changed |= ui.add(DragValue::new(&mut translation.z).max_decimals(3)).changed();
7              changed |= ui.add(DragValue::new(&mut translation.y).max_decimals(3)).changed();
8          });
9
10         *transform = Mat4::from_scale_rotation_translation(scale, rotation, translation);
11     }
12     GizmoTool::Rotate => {
13         ...
14     }
15     GizmoTool::Scale => {
16         ...
17     }
18 }
```

Abbildung 117 demonstriert das Translationswerkzeug im aktiven Zustand.

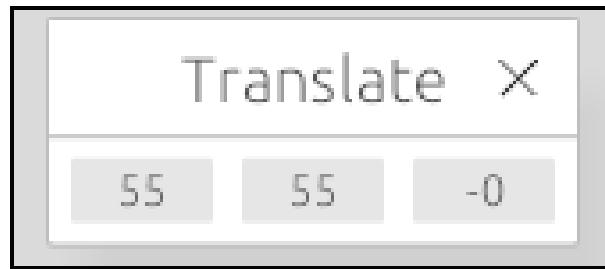


Abb. 117: Translations Werkzeug

6.11 Effiziente Anzeige von G-Code

Der **EfficientReader** ist eine speziell entwickelte Komponente zur effizienten Anzeige von langen G-Code-Dateien innerhalb der Benutzeroberfläche. Da G-Code-Dateien oft mehrere tausend Zeilen umfassen, liegt der Fokus dieser Implementierung darauf, eine ressourcenschonende und performante Darstellung zu ermöglichen, ohne den gesamten Code auf einmal zu laden.

- Liniertes Rendering: Nur die aktuell sichtbaren Zeilen des G-Codes werden gerendert, um Speicher- und Leistungsprobleme zu vermeiden.
- Syntax-Highlighting: Der Code wird farblich hervorgehoben, um wichtige Befehle und Parameter visuell hervorzuheben.
- Zeilenummerierung: Eine optionale Zeilenummerierung hilft bei der Orientierung innerhalb großer Dateien.

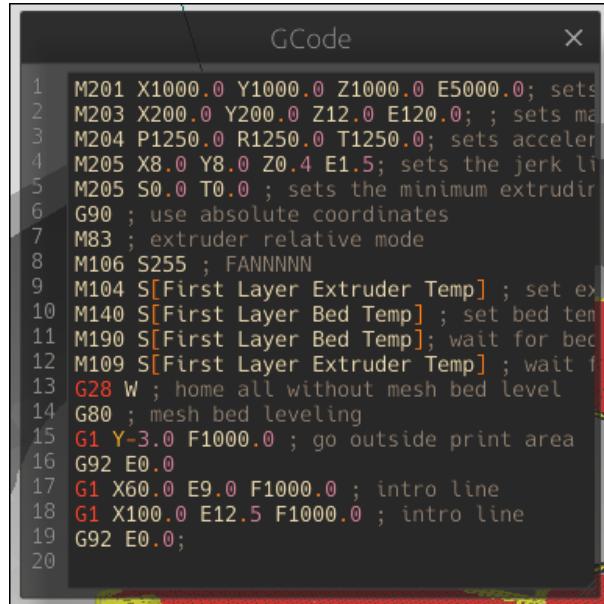


Abb. 118: GCode Leser

Abbildung 118 zeigt den integrierten G-Code-Reader mit Syntax-Hervorhebung und Zeilenummerierung.

6.11.1 Implementierung des EfficientReader

Die folgende Struktur definiert den `EfficientReader`, der auf `egui` und `egui_code_editor` basiert (siehe Listing 27). Die Implementierung nutzt einen eigenen Speicherbereich (`ReadSection`), um nur einen definierten Ausschnitt des Codes anzusehen (Listing 28).

Listing 27: Definition des EfficientReader

```

1 #[derive(Debug, PartialEq)]
2 pub struct EfficientReader<'a> {
3     id: String,
4     theme: ColorTheme,
5     syntax: Syntax,
6     numlines: bool,
7     fontsize: f32,
8     shrink: bool,
9     editable: bool,
10    view: &'a mut ReadSection,
11 }
```

Listing 28: Definition von ReadSection

```

1 #[derive(Debug, Clone, Default, Copy, PartialEq)]
2 pub struct ReadSection {
3     offset: usize,
4     size: usize,
5 }
```

6.11.2 Anzeige von G-Code mit dem EfficientReader

Die Anzeige von G-Code erfolgt innerhalb eines separaten Fensters, das über die `show()`-Methode des `GCodeTool` gesteuert wird. Der `EfficientReader` verarbeitet dabei nur die im aktuellen Viewport sichtbaren Zeilen, um eine performante Darstellung zu gewährleisten. Listing 29 zeigt die Integration in das `GCodeTool`:

Listing 29: Integration des EfficientReader im GCodeTool

```

1 impl Tool for GCodeTool<'__> {
2     fn show(
3         ...
4     ) -> bool {
5         if self.state.enabled {
6             global_state.viewer.sliced_gcode(|sliced_gcode| {
7                 egui::Window::new("GCode")
8                     .open(&mut self.state.enabled)
9                     .show(ctx, |ui| {
10                         EfficientReader::new(&mut self.state.view)
11                             .with_fontsize(14.0).with_theme(ColorTheme::GRUVBOX)
12                             .with_syntax(Syntax::gcode()).with_numlines(true)
13                             .show(ui, &sliced_gcode.gcode, &sliced_gcode.line_breaks);
14                     });
15                 });
16             }
17         self.state.enabled
18     }
19 }
```

6.12 Kamera

Der Kamera-Adapter (`CameraAdapter`) ist verantwortlich für die Steuerung der Kameraperspektive innerhalb der 3D-Ansicht. Er ermöglicht es dem Benutzer, das Modell aus verschiedenen Winkeln zu betrachten, zu zoomen und zu navigieren. Dabei integriert sich der Adapter in das bestehende Event-System und wird über das `Adapter-Trait` verwaltet.

6.12.1 Funktionalität des `CameraAdapter`

- Verwaltung der Kamera-Position und -Orientierung.
- Verarbeitung von Benutzerinteraktionen wie zoomen, schwenken und drehen.
- Integration in das Event-System der Anwendung, um Kamerabewegungen effizient zu aktualisieren.
- Berechnung der View- und Projection-Matrizen für das Rendering.

6.12.2 Die Implementierung des `CameraAdapter`

Die folgende Struktur (Listing 30) zeigt den grundlegenden Aufbau des Kamera-Adapters:

Listing 30: Struktur des `CameraAdapter`

```
1 pub struct CameraAdapter {
2     state: SharedMut<CameraState>,
3     event_reader: EventReader<CameraEvent>,
4 }
```

Die Initialisierung des `CameraAdapter` erfolgt in der `create()`-Funktion (siehe Listing 31), in der die Kamera-Parameter gesetzt und der Event-Reader initialisiert wird:

Listing 31: Erstellung des `CameraAdapter`

```
1 fn create(context: &WgpuContext) -> AdapterCreation<CameraState, CameraEvent, Self> {
2     let state = SharedMut::from_inner(CameraState::default());
3     let (reader, writer) = create_event_bundle::<CameraEvent>();
4
5     (
6         state.clone(),
7         writer,
8         Self {
9             state,
10            event_reader: reader,
11        },
12    )
13 }
```

6.12.3 Verarbeitung der Kamera-Ereignisse

Der `CameraAdapter` verarbeitet verschiedene Kamera-Ereignisse (vgl. Listing 32), darunter das Ändern der Kameraausrichtung und das Anpassen der Zoom-Stufe. Diese Ereignisse sind im `CameraEvent`-Enum definiert:

Listing 32: Definition der Kamera-Ereignisse

```
1 #[derive(Debug, Clone)]
2 pub enum CameraEvent {
3     CameraOrientationChanged(Orientation),
4     UpdatePreferredDistance(BoundingBox),
5 }
```

Die Ereignisverarbeitung erfolgt über die Methode `handle_event()` (Listing 33):

Listing 33: Verarbeitung der Kamera-Ereignisse

```

1 fn handle_event(
2     ...
3 ) {
4     match event {
5         CameraEvent::CameraOrientationChanged(orientation) => {
6             self.state.write().set_orientation(orientation);
7         }
8         CameraEvent::UpdatePreferredDistance(bounding_box) => {
9             self.state.write().update_preferred_distance(bounding_box);
10        }
11    }
12 }
```

6.13 Orbit-Kamera

Die Orbit-Kamera hat eine sphärische Bewegung um ein Zielobjekt. Sie wird häufig in 3D-Anwendungen verwendet, um eine intuitive Navigation zu ermöglichen. Der Benutzer kann die Kamera um das Ziel drehen, zoomen und neigen, während das Ziel fixiert bleibt.

6.13.1 Berechnung der Kameraposition

Die Position der Kamera wird in *sphärischen Koordinaten* beschrieben, die anschließend in kartesische Koordinaten umgerechnet werden müssen. Die folgenden Gleichungen (11, 12 und 13) verdeutlichen dies:

$$x = r \cdot \cos(\theta) \cdot \sin(\phi) \quad (11)$$

$$y = r \cdot \sin(\theta) \quad (12)$$

$$z = r \cdot \cos(\theta) \cdot \cos(\phi) \quad (13)$$

Variablenbeschreibung

- r ist die Entfernung der Kamera vom Ziel,
- θ ist der Pitch-Winkel (Neigung nach oben/unten),
- ϕ ist der Yaw-Winkel (Drehung um das Ziel).

Diese Umrechnung ist in der Funktion `calculate_cartesian_eye_position()` (Listing 34) implementiert:

Listing 34: Berechnung der Kameraposition in kartesischen Koordinaten

```

1 fn calculate_cartesian_eye_position(
2     pitch: f32,
3     yaw: f32,
4     distance: f32
5 ) -> Vec3 {
6     Vec3::new(
7         distance * yaw.sin() * pitch.cos(),
8         distance * pitch.sin(),
9         distance * yaw.cos() * pitch.cos(),
10    )
11 }
```

6.13.2 Berechnung der View- und Projection-Matrizen

In einer 3D-Rendering-Umgebung wird die Position der Kamera durch eine View-Matrix und die Perspektivdarstellung durch eine Projection-Matrix definiert. Die View-Matrix beschreibt die Transformation der Weltkoordinaten in Kamerakoordinaten und wird als **Look-At**-Matrix berechnet. In Gleichung (14) ist das allgemeine Schema einer Look-At-Matrix dargestellt, gefolgt von Gleichung (15), welche die konkrete Berechnung angibt:

$$\mathbf{V} = \begin{bmatrix} \mathbf{R}_x & \mathbf{R}_y & \mathbf{R}_z & -\mathbf{C} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (14)$$

$$\mathbf{V} = \text{look_at}(\mathbf{eye}, \mathbf{target}, \mathbf{up}) \quad (15)$$

Variablenbeschreibung

- \mathbf{C} ist die Position der Kamera,
- $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$ sind die Basisvektoren des Kamerakoordinatensystems.

Die Projection-Matrix wird für eine perspektivische Projektion definiert (Gleichung 16 und 17):

$$\mathbf{P} = \begin{bmatrix} \frac{1}{\tan(\frac{\text{fov}}{2}) \cdot \text{aspect}} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\text{fov}}{2})} & 0 & 0 \\ 0 & 0 & \frac{\text{far} + \text{near}}{\text{near} - \text{far}} & \frac{2 \cdot \text{far} \cdot \text{near}}{\text{near} - \text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (16)$$

$$\mathbf{P} = \text{perspective}(\text{fovy}, \text{aspect}, \text{near}, \text{far}) \quad (17)$$

Variablenbeschreibung

- fov ist das Sichtfeld (Field of View),
- aspect ist das Seitenverhältnis der Ansicht,
- near und far sind die Nah- und Fern-Clipping-Ebenen.

Diese Berechnungen sind in der Methode `build_view_proj_matrix()` (Listing 35) implementiert:

Listing 35: View- und Projektion-Matrizen der Orbit-Kamera

```

1 impl Camera for OrbitCamera {
2     fn build_view_proj_matrix(&self) -> (Mat4, Mat4) {
3         let view = Mat4::look_at_lh(self.eye, self.target, self.up);
4         let proj = Mat4::perspective_lh(
5             self.fovy,
6             self.aspect,
7             self.znear,
8             self.zfar
9         );
10
11         (view, proj)
12     }
13 }
```

6.13.3 Steuerung der Kamera

Die Kamera kann durch Anpassung der Winkel θ und ϕ rotiert werden. Die Methoden `add_yaw()` und `add_pitch()` (Listing 36) erlauben eine schrittweise Änderung:

Listing 36: Rotation der Kamera durch Anpassung der Winkel

```

1 pub fn add_yaw(&mut self, delta: f32) {
2     self.set_yaw(self.yaw + delta);
3 }
4
5 pub fn add_pitch(&mut self, delta: f32) {
6     self.set_pitch(self.pitch + delta);
7 }
```

6.13.4 Zoom-Funktion

Die Entfernung der Kamera zum Ziel kann mit der Funktion `add_distance()` (Listing 37) verändert werden:

Listing 37: Zoom-Steuerung der Kamera

```

1 pub fn add_distance(&mut self, delta: f32) {
2     self.set_distance(self.distance + delta);
3 }
```

Diese Funktion stellt sicher, dass die Kamera innerhalb eines definierten Minimal- und Maximalbereichs bleibt.

6.13.5 Einschränkungen der Kamerabewegung

Um zu verhindern, dass die Kamera sich außerhalb eines sinnvollen Bereichs bewegt, werden Grenzwerte für die Rotation und den Abstand zur Zielposition definiert (Listing 38):

Listing 38: Einschränkungen der Kamerabewegung

```

1 #[derive(Debug, Clone, Copy)]
2 pub struct OrbitCameraBounds {
3     pub min_distance: Option<f32>,
4     pub max_distance: Option<f32>,
5     pub min_pitch: f32,
6     pub max_pitch: f32,
7     pub min_yaw: Option<f32>,
8     pub max_yaw: Option<f32>,
9 }
```

6.13.6 Automatische Anpassung der Entfernung

Um eine optimale Ansicht zu gewährleisten, kann die Kamera ihre Distanz basierend auf der Größe eines Bounding-Volumes automatisch anpassen. In Listing 39 ist diese Funktion dargestellt:

Listing 39: Automatische Anpassung der Kameraentfernung

```

1 pub fn set_preferred_distance(&mut self, bounding_box: &BoundingBox) {
2     self.distance = bounding_box.diagonal().length();
3     self.update();
4 }
```

6.14 Der Kamera-Kontroller

Der Kamera-Kontroller (`CameraController`) ist verantwortlich für die Verarbeitung von Benutzereingaben zur Steuerung der Kamera innerhalb der 3D-Ansicht. Er interpretiert Mausbewegungen und Scroll-Events, um Rotationen, Bewegungen und Zoom-Operationen der Kamera umzusetzen. Dabei interagiert er direkt mit der `OrbitCamera`-Struktur und hat folgende Aufgaben:

- Erfassung von Maus- und Tastatureingaben zur Kamerasteuerung.
- Umsetzung von Drehbewegungen durch Drag-Events der Maus.
- Steuerung der Kamerabewegung innerhalb der Szene.
- Anpassung der Zoom-Stufe basierend auf Scroll-Events.

6.14.1 Implementierung des CameraController

Der `CameraController` besitzt mehrere Parameter zur Feinanpassung der Kamerasteuerung. In Listing 40 ist die Struktur definiert:

Listing 40: Definition der CameraController-Struktur

```

1 #[derive(Debug)]
2 pub struct CameraController {
3     pub rotate_speed: f32,
4     pub zoom_speed: f32,
5     pub move_speed: f32,
6
7     is_drag_rotate: bool,
8     is_drag_move: bool,
9 }
```

Die öffentlichen Felder definieren die Geschwindigkeiten für Drehungen, Zoom und Bewegung. Die privaten Felder `is_drag_rotate` und `is_drag_move` speichern den Status der Maussteuerung.

6.14.2 Verarbeitung von Window-Events

Die Methode `handle_window_events()` (Listing 41) verarbeitet Maus- und Scroll-Events:

Listing 41: Verarbeitung von Fenster-Events

```

1 pub fn handle_window_events(
2     ...
3 ) {
4     if !pointer_in_use {
5         match event {
6             WindowEvent::MouseWheel { delta, .. } => {
7                 let scroll_amount = -match delta {
8                     ...
9                 };
10                camera.add_distance(scroll_amount * self.zoom_speed);
11                window.request_redraw();
12            }
13            WindowEvent::MouseInput { button, state, .. } => match button {
14                ...
15            },
16            _ => (),
17        }
18    }
19 }
```

Das Mausrad beeinflusst die Zoom-Stufe der Kamera. Linke und mittlere Maustasten steuern Rotations- und Bewegungsoperationen.

6.14.3 Verarbeitung von Geräte-Events

Die Methode `handle_device_events()` (Listing 42) verarbeitet direkte Mausbewegungen und setzt sie in Kamerabewegungen um:

Listing 42: Verarbeitung von Geräte-Events

```

1 pub fn handle_device_events(
2     ...
3 ) {
4     if !pointer_in_use {
5         if self.is_drag_rotate {
6             if let DeviceEvent::MouseMove { delta } = event {
7                 camera.add_yaw(delta.0 as f32 * self.rotate_speed);
8                 camera.add_pitch(delta.1 as f32 * self.rotate_speed);
9                 window.request_redraw();
10            }
11        } else if self.is_drag_move {
12            if let DeviceEvent::MouseMove { delta } = event {
13                let direction = (camera.target - camera.eye).normalize();
14                let right = direction.cross(camera.up).normalize();
15                let up = right.cross(direction).normalize();
16
17                let move_amount = right * delta.0 as f32 + up * delta.1 as f32;
18                camera.eye += move_amount * self.move_speed;
19                camera.target += move_amount * self.move_speed;
20
21                window.request_redraw();
22            }
23        }
24    }
25 }
```

Bewegungen der Maus werden in Rotationen und Translationen der Kamera umgesetzt, während sich die Position der Kamera relativ dazu verändert.

6.15 Visualisierung

Der Render-Adapter (`RenderAdapter`) ist für die 3D-Grafikpipeline und die Integration der UI-Rendering-Ergebnisse verantwortlich. Er nutzt `wgpu` für Low-Level-Grafikoperationen und koordiniert die Darstellung von 3D-Modellen, Beleuchtung und UI. Die Struktur des Adapters, einschließlich der Hauptkomponenten der Rendering-Pipeline, ist in Listing 43 dargestellt.

6.15.1 Funktionalität des `RenderAdapter`

Der Adapter erfüllt folgende Kernaufgaben:

- Verwaltung der GPU-Ressourcen (Buffers, Textures, Bind Groups)
- Konfiguration der Rendering-Pipelines für 3D-Modelle
- Integration des UI-Renderings von `egui`
- Handhabung von Kamera- und Beleuchtungstransformationen
- Reaktion auf Fensterereignisse (Resize, Skalierung)

6.15.2 Implementierung

Die Struktur des Render-Adapters (siehe Listing 43) umfasst die Hauptkomponenten der Rendering-Pipeline und speichert den aktuellen Renderzustand.

Listing 43: Grundstruktur des `RenderAdapter`

```

1 pub struct RenderAdapter {
2     multisampled_framebuffer: wgpu::TextureView,
3     egui_rpass: egui_wgpu_backend::RenderPass,
4     pipelines: DefaultPipelines,
5     render_state: RenderState,
6     event_reader: EventReader<RenderEvent>,
7 }
8
9 struct RenderState {
10     depth_texture: Texture,
11     camera_uniform: CameraUniform,
12     camera_buffer: wgpu::Buffer,
13     camera_bind_group: wgpu::BindGroup,
14     light_uniform: LightUniform,
15     light_buffer: wgpu::Buffer,
16     light_bind_group: wgpu::BindGroup,
17 }
```

Initialisierung

Die Erstellung des Adapters erfolgt über die `create()`-Methode (siehe Listing 44):

Listing 44: Adapter-Initialisierung

```

1 fn create(context: &WgpuContext) -> AdapterCreation<(), RenderEvent, Self> {
2     ...
3
4     (
5         (),
6         writer,
7         RenderAdapter {
8             multisampled_framebuffer,
9             egui_rpass,
10            pipelines,
11            render_state,
12            event_reader: reader,
13        },
14    )
15 }
```

6.15.3 Rendering-Prozess

Die Hauptrendering-Logik ist in zwei Funktionen unterteilt: `handle_frame()` und `render()`. Erstere koordiniert die gesamte Rendering-Pipeline, während letztere die eigentliche 3D-Modell-Darstellung übernimmt. In Abbildung 119 ist der Ablauf des gesamten Rendering-Prozesses dargestellt.

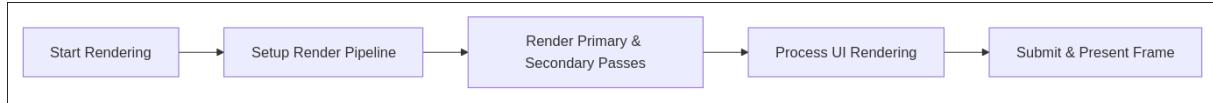


Abb. 119: Flussdiagramm-Rendering

Im ersten Schritt wird die `handle_frame()`-Methode aufgerufen, die den gesamten Rendering-Prozess steuert. Diese Methode ist für die Aktualisierung der Kamera, das Rendern der 3D-Modelle und die Integration der UI verantwortlich. Im Listing 45 ist der Rendering-Prozess dargestellt:

Listing 45: Renderintegration

```

1 fn handle_frame(
2     ...
3 ) -> Result<(), Error> {
4     // Camera updates
5     self.render_state.update(wgpu_context, proj * view, eye);
6
7     // Main rendering
8     self.render(&mut encoder, &view, &viewport, &state);
9     self.render_secondary(&mut encoder, &view, &viewport, &state);
10
11    // UI rendering integration
12    ...
13
14    // Finalization
15    ...
16 }
```

6.15.4 Fensterevent-Handling

Der Adapter reagiert auf Größenänderungen des Fensters. In Listing 46 wird gezeigt, wie auf ein Resize-Ereignis reagiert wird:

Listing 46: Resize-Handling

```

1 fn handle_window_event(
2     &mut self,
3     event: &winit::event::WindowEvent,
4     wgpu_context: &WgpuContext,
5     /* ... */
6 ) {
7     match event {
8         winit::event::WindowEvent::Resized(size) => {
9             self.render_state.depth_texture = Texture::create_depth_texture(
10                 /* ... */
11             );
12             self.multisampled_framebuffer =
13                 Texture::create_multisampled_framebuffer(
14                     /* ... */
15                 );
16         }
17     }
18 }
```

In der Abbildung 120 ist der Ablauf des Event-Handling-Prozesses dargestellt. Der Event-Loop verarbeitet alle eingehenden Ereignisse und leitet sie an die entsprechenden Module weiter. Dies geschieht in einer Schleife, die kontinuierlich auf neue Ereignisse wartet und diese verarbeitet.

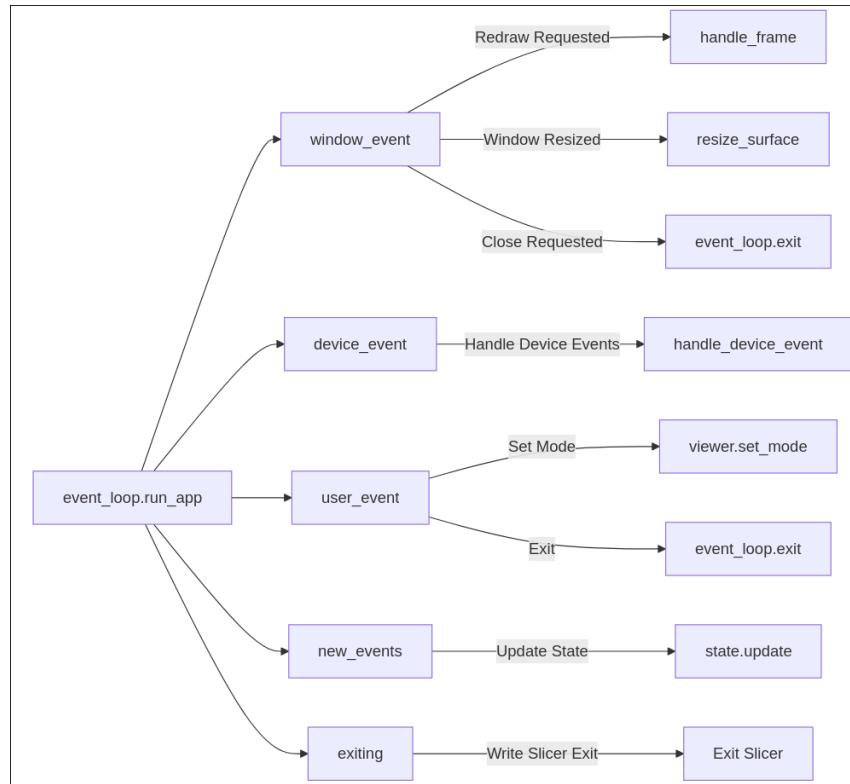


Abb. 120: Flussdiagramm-Events

6.15.5 Vertex- und Texturverarbeitung

Die korrekte Darstellung von 3D-Modellen erfordert eine effiziente Handhabung von Vertex- und Texturdaten. Dazu werden speziell definierte Strukturen zur Speicherung und Verarbeitung dieser Daten genutzt.

Vertex-Datenstruktur

Die Struktur `Vertex` repräsentiert einen einzelnen Punkt in einem 3D-Raum und enthält folgende Attribute:

- Position: Ein 3D-Vektor, der die Koordinaten des Punktes im Raum beschreibt.
- Normalenvektor: Definiert die Richtung der Normalen für Beleuchtungsberechnungen.
- Farbe: Ein RGBA-Farbwert zur Darstellung individueller Farbzueweisungen.

In Listing 47 ist die Definition der `Vertex`-Struktur abgebildet:

Listing 47: Definition der Vertex-Struktur

```

1  #[repr(C)]
2  #[derive(Copy, Clone, Debug, bytemuck::Pod, bytemuck::Zeroable)]
3  pub struct Vertex {
4      pub position: [f32; 3],
5      pub normal: [f32; 3],
6      pub color: [f32; 4],
7  }
  
```

Zur Beschreibung des Speicherlayouts innerhalb eines Vertex-Buffers stellt die Methode `desc()` die notwendigen Attribute bereit (siehe Listing 48):

Listing 48: Vertex-Buffer-Layout

```

1 impl Vertex {
2     pub fn desc<'a>() -> wgpu::VertexBufferLayout<'a> {
3         wgpu::VertexBufferLayout {
4             array_stride: std::mem::size_of::<Vertex>() as wgpu::BufferAddress,
5             step_mode: wgpu::VertexStepMode::Vertex,
6             attributes: &[
7                 wgpu::VertexAttribute {
8                     offset: 0,
9                     shader_location: 0,
10                    format: wgpu::VertexFormat::Float32x3,
11                },
12                wgpu::VertexAttribute {
13                    offset: std::mem::size_of::<[f32; 3]>() as wgpu::BufferAddress,
14                    shader_location: 1,
15                    format: wgpu::VertexFormat::Float32x3,
16                },
17                wgpu::VertexAttribute {
18                    offset: std::mem::size_of::<[f32; 6]>() as wgpu::BufferAddress,
19                    shader_location: 2,
20                    format: wgpu::VertexFormat::Float32x4,
21                },
22            ],
23        }
24    }
25 }
```

Texturierte Vertex-Daten

Für Objekte, die mit Texturen belegt werden, existiert die Struktur `TextureVertex`. Sie erweitert `Vertex` um eine zusätzliche Texturkoordinate (siehe Listing 49):

Listing 49: Definition der TextureVertex-Struktur

```

1 #[repr(C)]
2 #[derive(Copy, Clone, Debug, bytemuck::Pod, bytemuck::Zeroable)]
3 pub struct TextureVertex {
4     pub position: [f32; 3],
5     pub normal: [f32; 3],
6     pub tex_coords: [f32; 2],
7 }
```

Analog zu `Vertex` wird die Methode `desc()` definiert, um das Speicherlayout für texturierte Vertices zu beschreiben.

6.15.6 Texturen und Framebuffer

Die `Texture`-Struktur ist für die Handhabung von Texturen im Rendering-Prozess des Slicers zuständig. Da der Slicer keine besonders aufwendigen Texturen benötigt, dient diese Implementierung hauptsächlich der optischen Verbesserung.

Funktionalität der Texture-Struktur

- Erstellung von Tiefentexturen zur Tiefenberechnung in der Grafikpipeline
- Generierung von Multisampling-Framebuffers zur Verbesserung der Kantenglättung (MSAA)
- Laden und Verarbeiten von Bildtexturen aus Speicher oder Datei
- Bereitstellung von GPU-Speicher für Texturen mit passenden Filtern und Sampler-Einstellungen

6.15.7 Erstellung von Tiefentexturen

Tiefentexturen werden zur Berechnung der Tiefenwerte von Pixeln benötigt und sind essenziell für das Rendering. Die Methode `create_depth_texture` (siehe Listing 50) erstellt eine neue Tiefentextur für eine gegebene Gerät konfiguration:

Listing 50: Erstellung einer Tiefentextur

```

1 pub fn create_depth_texture(
2     ...
3 ) -> Self {
4     let desc = wgpu::TextureDescriptor {
5         ...
6     };
7     let texture = device.create_texture(&desc);
8     let sampler = device.create_sampler(&wgpu::SamplerDescriptor {
9         ...
10 });
11
12     let view = texture.create_view(&wgpu::TextureViewDescriptor::default());
13
14     Self {
15         texture,
16         view,
17         sampler,
18     }
19 }
```

6.15.8 Laden und Verarbeiten von Bildtexturen

Um Texturen aus einer Datei oder einem Speicher puffer zu laden, stehen die Methoden `from_bytes` und `from_image` zur Verfügung (siehe Listing 51 bzw. 52). Sie ermöglichen das Dekodieren und Hochladen von Bilddateien als GPU-Texturen:

Listing 51: Laden einer Bildtextur

```

1 pub fn from_bytes(
2     device: &wgpu::Device,
3     queue: &wgpu::Queue,
4     bytes: &[u8],
5     label: &str,
6 ) -> Result<Self> {
7     let img = image::load_from_memory(bytes)?;
8     Self::from_image(device, queue, &img, Some(label))
9 }
```

Listing 52: Verarbeitung einer Bildtextur

```

1 pub fn from_image(
2     ...
3 ) -> Result<Self> {
4     let texture = device.create_texture(&wgpu::TextureDescriptor {
5         label,
6         size,
7         ...
8     });
9
10    queue.write_texture(
11        wgpu::ImageCopyTexture {
12            ...
13        },
14        &rgba,
15        wgpu::ImageDataLayout {
16            ...
17        },
18        size,
19    );
20
21    let view = texture.create_view(&wgpu::TextureViewDescriptor::default());
22    let sampler = device.create_sampler(&wgpu::SamplerDescriptor {
23        ...
24    });
25
26    Ok(Self {
27        texture,
28        view,
29        sampler,
30    })
31 }

```

6.15.9 Pipeline Erstellung und Management

Das Pipeline-Management ist ein zentraler Bestandteil des Rendering-Systems. Es ermöglicht die flexible Konfiguration und Erstellung von Render-Pipelines. Die Hauptkomponente dieses Systems ist die `PipelineBuilder`-Struktur, die den Prozess der Pipeline-Erstellung kapselt und verschiedene Parameter wie Primitive- und Tiefenstencil-Einstellungen unterstützt.

Funktionalität des `PipelineBuilder`

Die `PipelineBuilder`-Struktur (siehe Listing 53) bietet eine bequeme Schnittstelle zur Erstellung von Render-Pipelines und erlaubt die Konfiguration folgender Aspekte:

- Definition des primitiven Zeichnungsmodus (z. B. `TriangleList`, `LineList`)
- Festlegung der Tiefen- und Stencil-Operationen zur Steuerung der Tiefentestlogik
- Integration von Shader-Modulen zur Vertex- und Fragmentbearbeitung
- Verwaltung von Bind-Group-Layouts zur Interaktion mit GPU-Ressourcen

Listing 53: Definition der PipelineBuilder-Struktur

```

1 pub struct PipelineBuilder {
2     device: Arc<Device>,
3     primitive: Option<wgpu::PrimitiveState>,
4     depth_stencil: Option<Option<wgpu::DepthStencilState>>,
5 }

```

Pipeline-Erstellung

Die Erstellung einer Render-Pipeline erfolgt über die Methode `build()` (siehe Listing 54), die verschiedene Parameter akzeptiert:

Listing 54: Erstellung einer Render-Pipeline

```

1 pub fn build(
2     ...
3 ) -> wgpu::RenderPipeline {
4     let shader = self.device.create_shader_module(
5         ...
6     );
7
8     let layout = self.device.create_pipeline_layout(
9         ...
10    );
11
12     self.device.create_render_pipeline(
13         &wgpu::RenderPipelineDescriptor {
14             label: Some(label),
15             layout: Some(&layout),
16             vertex: wgpu::VertexState {
17                 ...
18             },
19             fragment: Some(wgpu::FragmentState {
20                 module: &shader,
21                 entry_point: "fs_main",
22                 ...
23             }),
24             primitive: self.primitive.unwrap_or_default(),
25             depth_stencil: self.depth_stencil.unwrap_or_default(),
26             ...
27         })
28 }
```

6.15.10 Render Descriptor

Der `RenderDescriptor` ist eine Struktur zur Verwaltung von Rendering-Pass-Informationen und wird zur Initialisierung und Konfiguration der Rendering-Prozesse innerhalb der Pipeline verwendet (siehe Listing 55). Er definiert unter anderem die zu verwendenden Pipelines, Bind Groups und den Viewport.

Listing 55: Definition des RenderDescriptor

```

1 pub struct RenderDescriptor<'a> {
2     pub(super) pipelines: &'a render::DefaultPipelines,
3     pub(super) bind_groups: &'a [&'a wgpu::BindGroup],
4     pub(super) encoder: &'a mut wgpu::CommandEncoder,
5     pub(super) viewport: &'a render::Viewport,
6     pub(super) pass_descriptor: wgpu::RenderPassDescriptor<'a>,
7 }
```

Rendering-Prozess mit RenderDescriptor

Die Methode `pass()` (siehe Listing 56) ermöglicht das Initialisieren eines Render-Passes, sofern der definierte Viewport eine gültige Größe aufweist.

Listing 56: Render-Pass Initialisierung

```

1 pub fn pass(&mut self) -> Option<(&DefaultPipelines, wgpu::RenderPass)> {
2     let (x, y, width, height) = *self.viewport;
3
4     if width > 0.0 && height > 0.0 {
5         let mut render_pass = self.encoder.begin_render_pass(&self.pass_descriptor);
6         render_pass.set_viewport(x, y, width, height, 0.0, 1.0);
7
8         for (index, bind_group) in self.bind_groups.iter().enumerate() {
9             render_pass.set_bind_group(index as u32, *bind_group, &[]);
10        }
11
12        Some((self.pipelines, render_pass))
13    } else {
14        None
15    }
16 }
```

6.16 Modell und Transformation

Die `Model<T>` Struktur stellt ein renderbares Objekt in der Slicing-Software dar und ist in der Lage, Transformationen wie Translation, Rotation und Skalierung durchzuführen. Dabei wird eine `TransformUniform`-Struktur verwendet, um die Transformationen effizient auf der GPU zu speichern und anzuwenden.

Die `Model<T>` Struktur beinhaltet mehrere Komponenten:

- State: Der aktuelle Zustand des Modells, entweder `Dormant` oder `Awake` mit einem GPU-Buffer.
- Transform: Eine 4x4-Matrix, die die Transformation des Objekts im Raum beschreibt.
- Transform-Buffer: Ein `wgpu::Buffer`, der die Transformationsmatrix für die GPU bereitstellt.
- BindGroup: Eine Bindungsgruppe, die die Transformation mit dem Shader verbindet.
- ColorBinding: Eine separate Struktur zur Verwaltung der Farbe des Modells.
- Zustandsflags: Ein Flag zur Aktivierung/Deaktivierung des Modells und ein Zerstörungsindikator.

Listing 57 zeigt die Definition dieser Struktur:

Listing 57: Definition der Model-Struktur

```

1 #[derive(Debug)]
2 pub struct Model<T> {
3     state: ModelState,
4
5     transform: Mat4,
6     transform_buffer: wgpu::Buffer,
7     transform_bind_group: wgpu::BindGroup,
8
9     color_group: ColorBinding,
10
11    enabled: bool,
12    destroyed: bool,
13    _phantom: std::marker::PhantomData<T>,
14 }
```

6.16.1 Transformationen

Die `Model<T>` Struktur ermöglicht verschiedene Arten von Transformationen über eine 4x4-Matrix M , die folgendermaßen aufgebaut ist:

$$M = T \cdot R \cdot S \quad (18)$$

Variablenbeschreibung

- T ist die Translationsmatrix,
- R ist die Rotationsmatrix,
- S ist die Skalierungsmatrix.

Diese Operationen (siehe Gleichung 18) werden durch die Methode `transform` angewendet, wie in Listing 58 ersichtlich:

Listing 58: Transformation eines Modells

```

1 impl<T> TransformMut for Model<T> {
2     fn transform(&mut self, transform: Mat4) {
3         let queue_read = QUEUE.read();
4         let queue = queue_read.as_ref().unwrap();
5
6         self.transform = transform;
7         let transform_uniform = TransformUniform {
8             transform: self.transform.to_cols_array_2d(),
9         };
10
11        queue.write_buffer(
12            &self.transform_buffer,
13            0,
14            bytemuck::cast_slice(&[transform_uniform]),
15        );
16    }
17 }
```

Translation

Die Translation verschiebt das Modell im Raum entlang der Achsen x , y und z :

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Die Methode `Mat4::from_translation(Vec3::new(tx, ty, tz))` erstellt diese Matrix automatisch in Rust.

Rotation

Die Rotation erfolgt durch Multiplikation mit einer Rotationsmatrix um eine der drei Hauptachsen. Zum Beispiel die Rotation um die z -Achse um den Winkel θ :

$$R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Die Rotation in Rust kann mit `Mat4::from_rotation_z(theta)` umgesetzt werden.

Skalierung

Die Skalierung verändert die Größe des Modells entlang der Achsen x , y und z :

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dies kann in Rust mit `Mat4::from_scale(Vec3::new(sx, sy, sz))` realisiert werden.

Hinweis

Diese Operationen werden mit dem Crate `glam` durchgeführt, das eine Vielzahl von Funktionen zur Manipulation von Vektoren und Matrizen bereitstellt.

6.16.2 Rendering der Modell-Struktur

Das Modell wird über das Interface `Renderable` dargestellt. Der eigentliche Render-Aufruf erfolgt in Listing 59. Dabei wird unter anderem die transformierte Matrix an den Shader übergeben:

Listing 59: Render-Funktion

```

1 impl<T> Renderable for Model<T> {
2     fn render<'a>(&'a self, render_pass: &mut wgpu::RenderPass<'a>) {
3         if self.destroyed || !self.enabled {
4             return;
5         }
6
7         let (buffer, size) = match &self.state {
8             ModelState::Dormant => return,
9             ModelState::Awake(buffer, size) => (buffer, size),
10        };
11
12        render_pass.set_bind_group(2, &self.transform_bind_group, &[]);
13        render_pass.set_vertex_buffer(0, buffer.slice(..));
14        render_pass.draw(0..*size, 0..1);
15    }
16 }
```

6.17 Werkzeugpfad-Visualisierung

Der **TraceTree** dient als hierarchische Datenstruktur zur Darstellung von Bewegungsabläufen. Er ermöglicht eine effiziente Verwaltung von Modellgeometrien, Reisebewegungen und Faserpfaden (8.1.5 *Software*, S.201). Die Struktur ist darauf ausgelegt, eine schnelle Abfrage von Kollisionen, Interaktionen und Rendering-Operationen zu unterstützen. Wie in Abbildung 121 dargestellt, lassen sich die Pfade damit übersichtlich in einem Baum aufbauen.

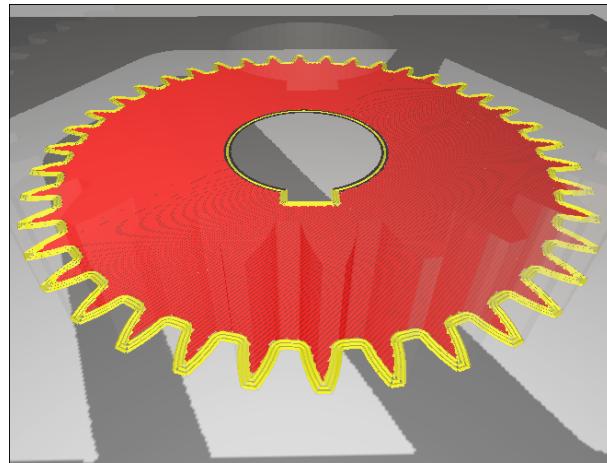


Abb. 121: Traces

6.17.1 Struktur und Hierarchie

Der **TraceTree** ist rekursiv aufgebaut und umfasst drei Haupttypen von Knoten:

- Root: Der Hauptknoten enthält Referenzen zu den untergeordneten Elementen und aggregiert die gesamte Bounding-Box des Modells.
- Trace: Diese Knoten repräsentieren spezifische Bewegungsabläufe eines Druckkopfes, wobei zwischen Bewegungen mit und ohne Faser unterschieden wird.
- Travel: Diese Knoten speichern Reisebewegungen zwischen einzelnen Pfaden.

Listing 60: Grundstruktur des TraceTree

```

1  #[derive(Debug)]
2  pub enum TraceTree {
3      Root {
4          model: LockModel<TraceVertex>,
5          children: Vec<Arc<Self>>,
6          size: BufferAddress,
7          ...
8      },
9      Travel {
10         offset: BufferAddress,
11         size: BufferAddress,
12         ...
13     },
14     Trace {
15         id: MoveId,
16         move_type: MoveType,
17         offset: BufferAddress,
18         size: BufferAddress,
19         r#box: RwLock<Box<TraceHitbox>>,
20     },
21 }

```

Listing 60 zeigt die grundlegende Struktur des **TraceTree**.

6.17.2 Erstellung eines TraceTree

Die Erstellung eines neuen Baumes erfolgt über die Methode `create_root()`, welche die Wurzelstruktur initialisiert (siehe Listing 61):

Listing 61: Initialisierung eines TraceTree

```

1 pub fn create_root() -> Self {
2     Self::Root {
3         model: LockModel::new(Model::create()),
4         fiber_model: LockModel::new(Model::create()),
5         travel_model: LockModel::new(Model::create()),
6
7         children: Vec::new(),
8         bounding_box: RwLock::new(BoundingBox::default()),
9         size: 0,
10        travel_size: 0,
11    }
12 }
```

Zusätzlich können spezifische Bewegungen oder Reisepfade über separate Methoden erzeugt werden, wie Listing 62 zeigt:

Listing 62: Erstellung von Bewegungspfaden

```

1 pub fn create_move(
2     path_box: TraceHitbox,
3     id: MoveId,
4     trace_type: TraceType,
5     offset: BufferAddress,
6     size: BufferAddress,
7 ) -> Self {
8     Self::Trace {
9         offset,
10        id,
11        move_type: MoveType::WithoutFiber(trace_type),
12        size,
13        r#box: RwLock::new(Box::new(path_box)),
14    }
15 }
```

6.17.3 Kollisionen und Auswahl

Das `HitboxNode`-Trait ermöglicht eine Kollisionserkennung zwischen dem 3D-Modell und Eingabeoperationen, beispielsweise für das Auswählen von Modellen innerhalb der Visualisierung (siehe Listing 63).

Listing 63: Definition des HitboxNode-Traits

```

1 pub trait HitboxNode {
2     fn check_hit(&self, ray: &crate::input::Ray) -> Option<f32>;
3     fn inner_nodes(&self) -> &[Arc<Self>];
4     fn get_min(&self) -> glam::Vec3;
5     fn get_max(&self) -> glam::Vec3;
6 }
```

Der `TraceTree` implementiert das `HitboxNode`-Trait, um Kollisionserkennung für alle Modelltypen zu unterstützen.

6.17.4 Interaktive Funktionen

Das `InteractiveModel`-Trait erweitert die Kollisionserkennung um interaktive Funktionen für Benutzerinteraktionen, beispielsweise das Auslösen von Mausereignissen oder das Abrufen von Transformationen (siehe Listing 64):

Listing 64: Definition des InteractiveModel-Traits

```

1 pub trait InteractiveModel {
2     fn aabb(&self) -> (Vec3, Vec3);
3     fn transformation(&self) -> glam::Mat4;
4     fn as_transformable(&self) -> Option<&dyn crate::render::model::Transform>;
5     fn mouse_right_click(&self);
6 }
```

6.17.5 Rendering

Der `TraceTree` implementiert das `Renderable`-Trait und kann somit innerhalb der 3D-Rendering-Pipeline dargestellt werden. In Listing 65 wird die zugehörige Rendering-Funktionalität gezeigt:

Listing 65: Rendering-Funktionalität des TraceTree

```

1 impl Renderable for TraceTree {
2     fn render<'a>(&'a self, render_pass: &mut wgpu::RenderPass<'a>) {
3         match self {
4             Self::Root { model, .. } => model.render(render_pass),
5             Self::Travel { .. } => panic!("Cannot render travel"),
6             Self::Trace { .. } => panic!("Cannot render path"),
7         }
8     }
9 }
```

6.17.6 Erzeugung und Verwaltung der Werkzeugpfad-Meshes

Der `TraceMesher` ist für die Erzeugung von Werkzeugpfaden in Form von 3D-Meshes zuständig. Diese Meshes visualisieren die Extrusionspfade und ermöglichen gleichzeitig Interaktionsmöglichkeiten durch Kollisionserkennung (siehe Listing 66):

Listing 66: Definition des TraceMesher

```

1 pub struct TraceMesher {
2     current_layer: usize,
3     current_type: Option<TraceType>,
4     color: Vec4,
5     last_cross_section: Option<TraceCrossSection>,
6     vertices: Vec<TraceVertex>,
7 }
```

- `current_layer`: Gibt an, in welcher Schicht sich der Mesher aktuell befindet.
- `current_type`: Speichert den aktuellen Bewegungstyp (`TraceType`).
- `color`: Bestimmt die Farbgebung des aktuellen Pfades.
- `last_cross_section`: Speichert das vorherige Querschnittsprofil, um Verbindungen zu ermöglichen.
- `vertices`: Enthält die generierten Eckpunkte für die Visualisierung.

6.17.7 Berechnung des Querschnitts (TraceCrossSection)

Ein Querschnitt der Extrusion wird durch vier Punkte A, B, C, D definiert, die den äußeren Rand des extrudierten Pfades beschreiben. Diese Punkte liegen auf zwei orthogonalen Achsen:

- Horizontale Achse: \mathbf{h}
- Vertikale Achse: \mathbf{v}

Die Positionen der Punkte im lokalen Koordinatensystem werden wie folgt berechnet:

$$\begin{aligned} A &= \mathbf{v} \cdot \frac{a}{2}, & C &= \mathbf{v} \cdot \left(-\frac{a}{2}\right) \\ B &= \mathbf{h} \cdot \frac{b}{2}, & D &= \mathbf{h} \cdot \left(-\frac{b}{2}\right) \end{aligned}$$

Variablenbeschreibung

- A, B, C, D sind die Punkte des Querschnitts.
- \mathbf{h} ist der Richtungsvektor entlang der horizontalen Achse.
- \mathbf{v} ist der Richtungsvektor entlang der vertikalen Achse.
- a ist die horizontale Dicke des Querschnitts.
- b ist die vertikale Dicke des Querschnitts.

6.17.8 Berechnung der Querschnittsverschiebung

Nachdem der Querschnitt für einen bestimmten Punkt berechnet wird, kann er entlang des Extrusionspfades verschoben werden. Die Transformation eines Querschnitts durch eine Translation \mathbf{t} ist gegeben durch:

$$A' = A + \mathbf{t}, \quad B' = B + \mathbf{t}, \quad C' = C + \mathbf{t}, \quad D' = D + \mathbf{t}$$

Variablenbeschreibung

- A, B, C, D sind die ursprünglichen Punkte des Querschnitts.
- A', B', C', D' sind die neuen Punkte nach der Verschiebung.
- \mathbf{t} ist der Verschiebungsvektor.

6.17.9 Berechnung der Mesh-Vernetzung

Für zwei aufeinanderfolgende Querschnitte (A_1, B_1, C_1, D_1) und (A_2, B_2, C_2, D_2) wird ein prismaisches Mesh generiert. Jede Seite wird durch zwei Dreiecke beschrieben:

$$\text{Seite 1: } (A_1, A_2, B_1), (B_1, A_2, B_2) \quad (19)$$

$$\text{Seite 2: } (B_1, B_2, D_1), (D_1, B_2, D_2) \quad (20)$$

$$\text{Seite 3: } (D_1, D_2, C_1), (C_1, D_2, C_2) \quad (21)$$

$$\text{Seite 4: } (C_1, C_2, A_1), (A_1, C_2, A_2) \quad (22)$$

Variablenbeschreibung

- A_1, B_1, C_1, D_1 sind die Punkte des ersten Querschnitts.
- A_2, B_2, C_2, D_2 sind die Punkte des zweiten Querschnitts.

6.17.10 Verbindungsnetz zwischen Abschnitten

Falls eine Verbindung zwischen zwei aufeinanderfolgenden Extrusionspfaden hergestellt werden muss, wird eine Brücke zwischen den Endpunkten generiert. Die Berechnung erfolgt durch:

$$\mathbf{m} = \frac{A_2 + B_2 + C_2 + D_2}{4}, \quad \mathbf{n} = \frac{A_1 + B_1 + C_1 + D_1}{4}$$

Die Verbindung wird durch ein Netz aus Dreiecken zwischen diesen beiden Mittelpunkten \mathbf{m} und \mathbf{n} gebildet.

6.17.11 Generierung der Werkzeugpfade

Der `TraceMesher` nutzt die Methode `next()`, um neue Abschnitte eines Werkzeugpfads zu generieren (siehe Listing 67):

Listing 67: Generierung von Werkzeugpfaden

```

1 pub fn next(
2     ...
3 ) -> (usize, TraceHitbox) {
4     let context_bits = match self.current_type {
5         Some(ty) => bit_representation(&ty),
6         None => bit_representation_setup(),
7     };
8
9     let start_profile =
10        TraceCrossSection::from_direction(end - start, horizontal, vertical).with_offset(start);
11
12    let end_profile =
13        TraceCrossSection::from_direction(end - start, horizontal, vertical).with_offset(end);
14
15    let mesh = TraceMesh::from_profiles(start_profile, end_profile).with_color(self.color);
16
17    if let Some(last_cross_section) = self.last_cross_section {
18        if connection {
19            ...
20        }
21    } else {
22        ...
23    }
24
25    self.last_cross_section = Some(end_profile);
26
27    let toolpath_vertices = mesh
28        .to_triangle_vertices()
29        .into_iter()
30        .map(|v| TraceVertex::from_vertex(v, context_bits, self.current_layer as u32));
31
32    let offset = self.vertices.len();
33
34    self.vertices.extend(toolpath_vertices);
35
36    (offset, TraceHitbox::from(mesh))
37 }
```

Diese Methode erzeugt neue Querschnitte für den Werkzeugpfad und speichert die Vertex-Daten. Falls ein vorheriger Querschnitt existiert und eine Verbindung erforderlich ist, wird zusätzlich eine `TraceConnectionMesh` erstellt.

6.17.12 Verwaltung der Pfade

Die Struktur `SlicedObject` stellt das Datenmodell für geslicte Geometrien dar. Sie übernimmt die Verwaltung der Werkzeugpfade, speichert deren Eigenschaften und generiert daraus ein hierarchisches Baumdiagramm (`TraceTree`), das für Visualisierungs- und Interaktionszwecke genutzt wird (siehe Listing 68).

Listing 68: Definition der SlicedObject-Struktur

```

1 #[derive(Debug)]
2 pub struct SlicedObject {
3     pub model: Arc<TraceTree>,
4     pub count_map: HashMap<TraceType, usize>,
5     pub max_layer: usize,
6     pub moves: Vec<Command>,
7     pub settings: slicer::Settings,
8 }
```

- `model`: Ein `TraceTree`-Objekt, das die hierarchische Struktur der Werkzeugpfade speichert.
- `count_map`: Eine `HashMap` zur Speicherung der Anzahl der einzelnen Bewegungstypen.
- `max_layer`: Die maximale Layer-Höhe des gesuchten Objekts.
- `moves`: Eine Liste der verarbeiteten Slicing-Befehle.
- `settings`: Die Einstellungen, die beim Slicing verwendet werden.

6.17.13 Erstellung eines SlicedObject

Ein `SlicedObject` kann aus einer Liste von Befehlen generiert werden. Dies geschieht mit der Methode `from_commands()` (siehe Listing 69):

Listing 69: Erstellung eines SlicedObject

```

1 pub fn from_commands(
2     commands: &[slicer::Command],
3     settings: &slicer::Settings,
4     _process: &Process,
5 ) -> Result<Self, ()>;
```

Die Methode verarbeitet die Liste der `Command`-Objekte und wandelt sie in ein `TraceTree` um.

Ablauf

1. Initialisierung:
 - Ein leerer `TraceTree` wird erzeugt.
 - Zwei Instanzen von `TraceMesher` werden erstellt (eine für normale Pfade, eine für Faserpfade).
2. Iterieren durch die Befehle:
 - Jeder Befehl wird interpretiert und entsprechend verarbeitet.
 - `MoveTo`-Befehle erzeugen Reisebewegungen.
 - `MoveAndExtrude`-Befehle generieren Extrusionspfade und speichern diese als `TraceTree`-Knoten.
3. Erstellung der Hitboxen:
 - Jeder Werkzeugpfad wird mit einer `TraceHitbox` versehen, um Kollisionserkennung zu ermöglichen.

Listing 70: Verarbeitung eines MoveAndExtrude-Befehls

```

1  slicer::Command::MoveAndExtrude {
2      ...
3  } => {
4      let start = Vec3::new(
5          ...
6      );
7      let end = Vec3::new(
8          ...
9      );
10
11     let (offset, hitbox) = mesher.next(start, end, *thickness, *width, true);
12
13     let tree_move = TraceTree::create_move(
14         ...
15     );
16
17     root.push(tree_move);
18 }
```

Wie in Listing 70 zu sehen, sorgt diese Implementierung dafür, dass jeder extrudierte Pfad als eigener Eintrag im `TraceTree` gespeichert wird.

6.17.14 Integration mit TraceMesher

`SlicedObject` nutzt den `TraceMesher`, um Werkzeugpfade zu erzeugen. Die Methode `next()` im `TraceMesher` erzeugt für jeden Abschnitt ein neues Mesh und berechnet die Hitbox (siehe Listing 71):

Listing 71: Mesh-Erstellung mit TraceMesher

```
1 let (offset, hitbox) = mesher.next(start, end, *thickness, *width, true);
```

6.17.15 Kollisionserkennung mit HitboxNode

Jeder Werkzeugpfad wird mit einer `TraceHitbox` ausgestattet, die die Implementierung des `HitboxNode`-Traits ermöglicht. Dadurch können Kollisionen mit den Pfaden erkannt werden.

6.18 Shader für Werkzeugpfade

Der Shader für die Darstellung von Werkzeugpfaden im `SlicedObjectServer` ist dafür verantwortlich, die Sichtbarkeit einzelner Pfade anhand der aktuellen Ansichtseinstellungen zu bestimmen und eine realistische Beleuchtung der Pfade zu ermöglichen. Shader bestehen grundsätzlich aus zwei Hauptkomponenten:

- Vertex Shader: Transformiert die Positionen der Werkzeugpfade und entscheidet, ob sie sichtbar sind.
- Fragment Shader: Berechnet die Beleuchtung und Farbe der Werkzeugpfade basierend auf Lichtquellen und Materialeigenschaften.

6.18.1 Vertex Shader

Der Vertex Shader transformiert die Positionen der Werkzeugpfade und prüft, ob sie sichtbar sind. Falls ein Pfad sichtbar ist, wird er an den Fragment Shader weitergegeben. Listing 72 zeigt eine Beispielimplementierung:

Listing 72: Vertex Shader zur Sichtbarkeitsprüfung

```

1  @vertex
2  fn vs_main(in: VertexInput) -> VertexOutput {
3      var out: VertexOutput;
4
5      if (in.trace_type & context.visibility) > 0 &&
6          in.layer >= context.min_layer &&
7          in.layer <= context.max_layer {
8
9          out.world_normal = in.normal;
10         var pos = transform.matrix * vec4<f32>(in.position, 1.0);
11
12         out.world_position = pos.xyz;
13         out.clip_position = camera.view_proj * pos;
14         out.camera_view_pos = camera.view_pos;
15         out.color = in.color;
16     }
17
18     return out;
19 }
```

Ablauf

- Sichtbarkeitsprüfung: Falls der Pfad sichtbar ist, wird er transformiert und an den Fragment Shader weitergegeben.
- Transformation: Die Positionen werden mit der Transformationsmatrix multipliziert.
- Kameraprojektion: Die Positionen werden mit der Kameraprojektionsmatrix verarbeitet.

6.18.2 Sichtbarkeitsprüfung

Die Sichtbarkeit eines Werkzeugpfads wird mithilfe einer `if`-Abfrage geprüft. Dabei werden zwei Kriterien berücksichtigt:

- Der Pfad muss sich in einer aktuell sichtbaren Schicht (Layer) befinden:

Listing 73: Prüfung der Sichtbarkeit nach Layer

```
1 in.layer >= context.min_layer && in.layer <= context.max_layer
2
```

- Der `trace_type` des Pfads muss mit dem Sichtbarkeits-Flag (`context.visibility`) übereinstimmen. Die Sichtbarkeit wird durch eine Bitmaske verwaltet:

Listing 74: Prüfung der Sichtbarkeit nach Typ

```
1 if (in.trace_type & context.visibility) > 0
2
```

`visibility` ist eine 32-Bit-Ganzzahl, wobei jedes Bit einem bestimmten `TraceType` zugewiesen ist. Dadurch können bis zu 32 verschiedene Pfadtypen gleichzeitig sichtbar oder unsichtbar geschaltet werden.

6.18.3 Datenstrukturen

Der Shader nutzt mehrere Uniform-Strukturen zur Verwaltung von Kamera-, Licht- und Transformationsinformationen:

- `Camera`: Speichert die Kamera-Position und Projektionsmatrix.
- `Light`: Definiert die Lichtposition und Lichtfarbe.
- `Transform`: Enthält die Transformationsmatrix für die Modelltransformation.
- `Color`: Speichert die globale Farbskalierung.
- `ToolpathContext`: Enthält die Sichtbarkeitseinstellungen für Werkzeugpfade.

6.18.4 Fragment Shader

Der Fragment Shader berechnet die Beleuchtung und die endgültige Farbe des Pfads anhand des Blinn-Phong-Modells. Die Beispielimplementierung in Listing 75 illustriert dies:

Listing 75: Fragment Shader mit Blinn-Phong-Beleuchtung

```
1 @fragment
2 fn fs_main(in: VertexOutput) -> @location(0) vec4<f32> {
3     let ambient_color = light.color.xyz * light.color.a;
4
5     let light_dir = normalize(light.position.xyz - in.world_position);
6
7     let diffuse_strength = max(dot(in.world_normal, light_dir), 0.0);
8     let diffuse_color = light.color.xyz * diffuse_strength;
9
10    let view_dir = normalize(in.camera_view_pos.xyz - in.world_position);
11    let half_dir = normalize(view_dir + light_dir);
12    let specular_strength = pow(max(dot(in.world_normal, half_dir), 0.0), 32.0);
13
14    let specular_color = light.color.xyz * specular_strength;
15
16    let result = (ambient_color + diffuse_color + specular_color) * in.color.xyz;
17    return vec4<f32>(
18        result.r * color.color.r,
19        result.g * color.color.g,
20        result.b * color.color.b,
21        in.color.a * color.color.a
22    );
23 }
```

- Umgebungslicht (Ambient Lighting): Grundbeleuchtung, unabhängig von Lichtquellen.
- Diffuse Beleuchtung: Abhängig vom Winkel des Lichts zur Normalen.
- Spiegelnde Reflexion (Specular Lighting): Hervorhebung von Glanzeffekten mit Blinn-Phong.

6.19 Verwaltung und Rendering geslicerter Objekte

Der `SlicedObjectServer` ist für die Verwaltung, Verarbeitung und das Rendering der Werkzeugpfade verantwortlich. Er verwaltet sowohl die Visualisierung der geslicten Pfade als auch die Speicherung und den Export der generierten G-Code-Dateien.

6.19.1 Rendering der Werkzeugpfade

Die Methode `render()`, siehe Listing 76, stellt die Werkzeugpfade dar und prüft, ob Faserpfade sichtbar sind:

Listing 76: Rendering der Werkzeugpfade

```

1 fn render(&'a self, render_pass: &mut wgpu::RenderPass<'a>) {
2     if let Some(toolpath) = self.sliced_object.as_ref() {
3         render_pass.set_bind_group(4, &self.toolpath_context_bind_group, &[]);
4         render_pass.set_pipeline(&self.pipeline);
5
6         if self.fiber_visible {
7             toolpath.model.render_fiber(render_pass);
8         }
9
10        render_pass.set_bind_group(3, self.trace_color_group.binding(), &[]);
11        toolpath.model.render_without_color(render_pass);
12    }
13 }
```

6.19.2 Laden und Verarbeitung geslicerter Daten

Die Methode `load_from_slice_result()`, die in Listing 77 dargestellt ist, verarbeitet die von der Slicing-Engine gelieferten Daten und startet eine asynchrone Aufgabe zum Laden des G-Codes:

Listing 77: Laden eines geslicerter Objekts

```

1 pub fn load_from_slice_result(&mut self, slice_result: SliceResult, process: Arc<Process>) {
2     let (tx, rx) = tokio::sync::oneshot::channel();
3
4     let handle = tokio::spawn(async move {
5         process.set_task("Loading toolpath".to_string());
6         process.set_progress(0.8);
7
8         let obj = SlicedObject::from_commands(&slice_result.moves, &slice_result.settings, &process)
9             .expect("Failed to load toolpath");
10
11        process.set_task("Build GCode".to_string());
12        process.set_progress(0.9);
13
14        let mut writer = GCodeMemoryWriter::new();
15        let navigator = write_gcode(&slice_result.moves, &slice_result.settings, &mut writer).unwrap();
16        let sliced_gcode = writer.finish(navigator);
17
18        process.set_progress(1.0);
19        tx.send((obj, sliced_gcode, process)).unwrap();
20    });
21
22    self.queued = Some((rx, handle));
23 }
```

6.19.3 Export der Werkzeugpfade als G-Code

Die Methode `export()`, siehe Listing 78, ermöglicht es dem Nutzer, den generierten G-Code als Datei zu speichern:

Listing 78: Export von G-Code

```

1 pub fn export(&self) {
2     if let Some(toolpath) = self.sliced_object.as_ref() {
3         let path = FileDialog::new()
4             .set_location("~/")
5             .set_filename("model.gcode")
6             .set_title("Export GCode")
7             .add_filter("GCode", &["gcode"])
8             .show_save_single_file()
9             .unwrap();
10
11     if let Some(path) = path {
12         let file = match File::create_new(path) {
13             Ok(file) => file,
14             Err(e) => {
15                 println!("Failed to create file: {:?}", e);
16                 return;
17             }
18         };
19
20         let mut writer = BufWriter::new(file);
21         let mut writer = GCode.FileWriter::new(&mut writer);
22
23         match write_gcode(&toolpath.moves, &toolpath.settings, &mut writer) {
24             Ok(_) => println!("Gcode saved"),
25             Err(e) => println!("Failed to save gcode: {:?}", e),
26         }
27     }
28 }
29 }
```

6.19.4 Steuerung der Sichtbarkeit von Pfaden

Die Sichtbarkeit einzelner Werkzeugpfade kann über Bitmasken gesteuert werden. Die Methode `update_visibility()`, dargestellt in Listing 79, aktualisiert die Sichtbarkeit und schickt die Daten an die GPU:

Listing 79: Aktualisierung der Werkzeugpfad-Sichtbarkeit

```

1 pub fn update_visibility(&mut self, value: u32) {
2     self.toolpath_context.visibility = value;
3
4     let queue_read = QUEUE.read();
5     let queue = queue_read.as_ref().unwrap();
6
7     queue.write_buffer(
8         &self.toolpath_context_buffer,
9         0,
10        bytemuck::cast_slice(&[self.toolpath_context]),
11    );
12 }
```

6.20 Verwaltung und Rendering von CAD-Modellen

Der `ObjectServer` verwaltet CAD-Modelle innerhalb der 3D-Visualisierung. Er ermöglicht das Laden, Speichern, Rendern und Interagieren mit den Objekten. Neben dem Standard-`ObjectServer` gibt es noch spezialisierte Server:

- EnvironmentServer: Verwaltung von Umgebungsobjekten.
- MaskServer: Handhabt CAD-Modelle als Masken.

Da alle Server ähnlich aufgebaut sind, wird hier ausschließlich der `ObjectServer` beschrieben, der folgende Aufgaben übernimmt:

- Laden von CAD-Modellen aus Dateien oder Byte-Daten.
- Verwaltung und Speicherung von Modellen in einer internen Struktur.
- Rendern der Objekte mit Unterstützung für Transparenz und Interaktion.
- Kollisionsprüfung und Objektselektion.

6.20.1 Rendering der CAD-Modelle

Die Methode `render()`, siehe Listing 80, stellt alle geladenen Modelle dar:

Listing 80: Rendering der Modelle

```

1 fn render<'a>(&'a self, render_pass: &mut wgpu::RenderPass<'a>) {
2     render_pass.set_bind_group(3, &self.color_bind_group, &[]);
3
4     self.models.values().for_each(|model| {
5         model.model.render_without_color(render_pass);
6     });
7 }
```

6.20.2 Laden von CAD-Dateien

Das Laden eines Modells kann entweder aus einer Datei oder direkt aus Byte-Daten erfolgen. Die Methode `load_from_file()`, in Listing 81 dargestellt, verarbeitet die Datei:

Listing 81: Laden eines CAD-Modells aus einer Datei

```

1 pub fn load_from_file<P>(&mut self, path: P)
2 where
3     P: AsRef<Path>,
4 {
5     let file_name = path.as_ref().file_name().unwrap().to_string_lossy().to_string();
6     let path = path.as_ref().to_str().unwrap_or("").to_string();
7
8     // Starte asynchrone Aufgabe zum Laden des Modells
9     let (tx, rx) = tokio::sync::oneshot::channel();
10
11    // Lade das Modell im Hintergrund
12    let handle = tokio::spawn(async move {
13        let mesh = match shared::loader::STLLoader {}.load(&path) {
14            Ok(model) => model,
15            Err(e) => {
16                tx.send(Err(Error::LoadError(e))).unwrap();
17                return;
18            }
19        };
20
21        let result = Self::load(file_name, mesh);
22        tx.send(result.unwrap());
23    });
24
25    self.queue.push((rx, handle));
26 }
```

6.20.3 Verarbeitung des CAD-Modells

Nach dem Laden wird das Modell analysiert, gefiltert und mit Farben versehen. Listing 82 zeigt die Funktion `load()`, in der die einzelnen Schritte durchgeführt werden:

Listing 82: Verarbeitung der CAD-Daten

```

1 fn load(name: String, mesh: ObjectMesh) -> CADObjectResult {
2     let (min, max) = mesh.min_max();
3
4     let vertices: Vec<Vec3> = mesh.vertices().iter().map(|v| v.xzy()).collect();
5
6     // Normalisiere die Dreiecke und berechne die Normalen
7     let mut triangles: Vec<(shared::IndexedTriangle, Vec3)> = ...;
8
9     let models = clusterize_models(&triangles);
10
11    // Extrahiere die einzelnen Vertexpunkte
12    let vertices = ...;
13
14    // Gruppiere die Polygonflächen, um Hitboxen zu erstellen und Mausinteraktion zu ermöglichen
15    let polygon_faces: Vec<PolygonFace> = clusterize_faces(&triangles, &vertices)...;
16
17    // Erstelle das CAD-Objekt mit den FlächenHitboxen
18    let mut root = ...;
19
20    root.awaken(&vec3s_into_vertices(vertices, Color::BLACK));
21
22    Ok(LoadResult {
23        ...
24    })
25 }
```

6.21 Auswahl von Objekten

Die Picking-Implementierung ermöglicht es, Objekte in einer 3D-Szene durch einen Mausklick auszuwählen. Dies geschieht durch die Umrechnung von Bildschirmkoordinaten in einen 3D-Strahl (**Ray**) und die Kollisionserkennung mit Bounding-Boxen (**HitboxRoot**). Dabei wird in den folgenden Abschnitten Schritt für Schritt beschrieben, wie aus Mauskoordinaten ein Strahl im Weltkoordinatensystem entsteht und wie letztlich Kollisionen mit Objekten erkannt werden können.

6.21.1 Umwandlung der Mauskoordinaten in Normalized Device Coordinates (NDC)

Um herauszufinden, welches Objekt sich unter der Maus befindet, wird ein 3D-Strahl aus der Sicht der Kamera in die Szene projiziert. Dies geschieht in mehreren Schritten. Zunächst befindet sich der Mauszeiger in Fensterkoordinaten ($x_{\text{mouse}}, y_{\text{mouse}}$), wobei die obere linke Ecke des Fensters den Ursprung $(0,0)$ und die untere rechte Ecke (w, h) bildet. Diese Werte werden in Normalized Device Coordinates (NDC) umgewandelt, die sich im Bereich $[-1, 1]$ erstrecken. Die Umrechnung erfolgt gemäß Gleichungen (23) und (24):

$$\text{ndc}_x = 1 - \frac{2 \cdot (x_{\text{mouse}} - x_{\text{viewport}})}{w_{\text{viewport}}} \quad (23)$$

$$\text{ndc}_y = \frac{2 \cdot (y_{\text{mouse}} - y_{\text{viewport}})}{h_{\text{viewport}}} - 1 \quad (24)$$

Variablenbeschreibung

- $x_{\text{viewport}}, y_{\text{viewport}}$ geben die Position des Viewports im Fenster an.
- $w_{\text{viewport}}, h_{\text{viewport}}$ sind die Breite und Höhe des Viewports.

Der entsprechende Rust-Code zur Umrechnung in NDC ist in Listing 83 zu sehen.

Listing 83: Umrechnung in NDC

```
1 let ndc_x = 1.0 - (2.0 * (position.0 - x)) / width;
2 let ndc_y = (2.0 * (position.1 - y)) / height - 1.0; // Y-Achse invertieren
```

6.21.2 Umwandlung in Clip-Space-Koordinaten

Da WGPU im Clip-Space ein Koordinatensystem verwendet, das sich im Bereich $[-1, 1]$ erstreckt, werden die NDC-Koordinaten direkt in den Clip-Space übernommen. Dies geschieht gemäß Gleichung (25):

$$\mathbf{p}_{\text{clip}} = \begin{bmatrix} \text{ndc}_x \\ \text{ndc}_y \\ -1 \\ 1 \end{bmatrix} \quad (25)$$

Listing 84 zeigt die Berechnung in Rust:

Listing 84: Berechnung der Clip-Koordinaten

```
1 let clip_coords = glam::Vec4::new(ndc_x, ndc_y, -1.0, 1.0);
```

6.21.3 Rücktransformation in Eye-Space

Um den Strahl aus der Kamera heraus zu berechnen, wird die inverse Projektionsmatrix P^{-1} benötigt. Zunächst wird

$$\mathbf{p}_{\text{eye}} = P^{-1} \cdot \mathbf{p}_{\text{clip}} \quad (26)$$

berechnet. Anschließend erfolgt eine perspektivische Division, wobei der w -Wert auf 0 gesetzt wird, da ein Strahl keine Position, sondern eine Richtung darstellt:

$$\mathbf{p}_{\text{eye}} = \begin{bmatrix} x_{\text{eye}} \\ y_{\text{eye}} \\ -1 \\ 0 \end{bmatrix} \quad (27)$$

Listing 85 zeigt diesen Schritt:

Listing 85: Transformation in den Eye-Space

```
1 let inv_proj = proj.inverse();
2 let eye_coords = inv_proj * clip_coords;
3 let eye_coords = glam::Vec4::new(eye_coords.x, eye_coords.y, -1.0, 0.0);
```

6.21.4 Transformation in Weltkoordinaten

Um aus dem Eye-Space-Strahl einen Strahl im Weltkoordinatensystem zu erhalten, wird die inverse View-Matrix V^{-1} multipliziert, siehe Gleichung (28):

$$\mathbf{d}_{\text{world}} = V^{-1} \cdot \mathbf{p}_{\text{eye}} \quad (28)$$

Variablenbeschreibung

- V^{-1} ist die inverse View-Matrix (Kamera-Transformation).
- $\mathbf{d}_{\text{world}}$ ist die Richtung des Strahls im Weltkoordinatensystem.

Listing 86 zeigt den Code in Rust:

Listing 86: Transformation in Weltkoordinaten

```
1 let inv_view = view.inverse();
2 let world_coords = inv_view * eye_coords;
3 let direction = Vec3::new(world_coords.x, world_coords.y, world_coords.z).normalize();
```

6.21.5 Berechnung der Schnittpunkte mit Ebenen

Ein wichtiger Teil des Picking ist die Berechnung von Schnittpunkten mit Objekten oder Referenzebenen. Um den Schnittpunkt eines Strahls mit einer Ebene zu bestimmen, wird die allgemeine Ebenengleichung aus (29) und (30) herangezogen:

$$t = \frac{(\mathbf{p}_{\text{plane}} - \mathbf{o}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}} \quad (29)$$

$$\mathbf{p}_{\text{intersection}} = \mathbf{o} + t \cdot \mathbf{d} \quad (30)$$

Variablenbeschreibung

- $\mathbf{p}_{\text{plane}}$ ist ein Punkt auf der Ebene.
- \mathbf{n} ist der Normalenvektor der Ebene.
- \mathbf{o} ist der Ursprung des Strahls.
- \mathbf{d} ist die Richtung des Strahls.
- t ist der Parameter, der den Schnittpunkt beschreibt.
- $\mathbf{p}_{\text{intersection}}$ ist der Schnittpunkt des Strahls mit der Ebene.

Listing 87 zeigt die entsprechende Implementierung:

Listing 87: Berechnung des Schnittpunkts mit einer Ebene

```

1 pub fn intersection_plane(&self, plane: Vec3, point: Vec3) -> Vec3 {
2     let d = plane.dot(self.direction);
3     if d.abs() > f32::EPSILON {
4         let t = (point - self.origin).dot(plane) / d;
5         self.origin + self.direction * t
6     } else {
7         Vec3::new(0.0, 0.0, 0.0)
8     }
9 }
```

6.21.6 Hierarchische Kollisionserkennung mit Hitboxen

Damit die Objektauswahl effizient funktioniert, wird eine hierarchische Bounding-Box-Struktur verwendet. Die `HitboxRoot`-Struktur speichert alle Objekte, die auf Kollision geprüft werden müssen. Die Verwaltung dieser Hitboxen ist in Listing 88 dargestellt.

Listing 88: Hierarchische Hitbox-Verwaltung

```

1 impl<M: HitboxNode + Renderable> HitboxRoot<M> {
2     pub fn check_hit(&self, ray: &Ray, level: usize, reverse: bool) -> Option<Arc<M>> {
3         let mut queue = HitboxQueue::<M>::new(); // Prioritaetswarteschlange fuer Kollisionen
4
5         // Iteriere ueber alle Hitboxen und pruefe Kollision
6         for hitbox in self.inner_hitboxes.iter() {
7             ...
8         }
9
10        while let Some(HitBoxQueueEntry {
11            ...
12            }) = queue.pop()
13        {
14            if hitbox.inner_nodes().is_empty() || level == entry_level {
15                return Some(hitbox);
16            } else {
17                // Iteriere ueber alle inneren Hitboxen
18                for inner_hitbox in hitbox.inner_nodes() {
19                    ...
20                }
21            }
22        }
23
24        None
25    }
26 }
```

Um den Algorithmus besser zu verstehen, wird in Abbildung 122 ein Flussdiagramm dargestellt, das den Auswahlprozess veranschaulicht. Zunächst wird die Hitbox des Objekts überprüft. Wenn eine Kollision festgestellt wird, wird die Hitbox in die Warteschlange eingefügt. Anschließend wird die Warteschlange nach der Entfernung sortiert, um das nächstgelegene Objekt zu finden.

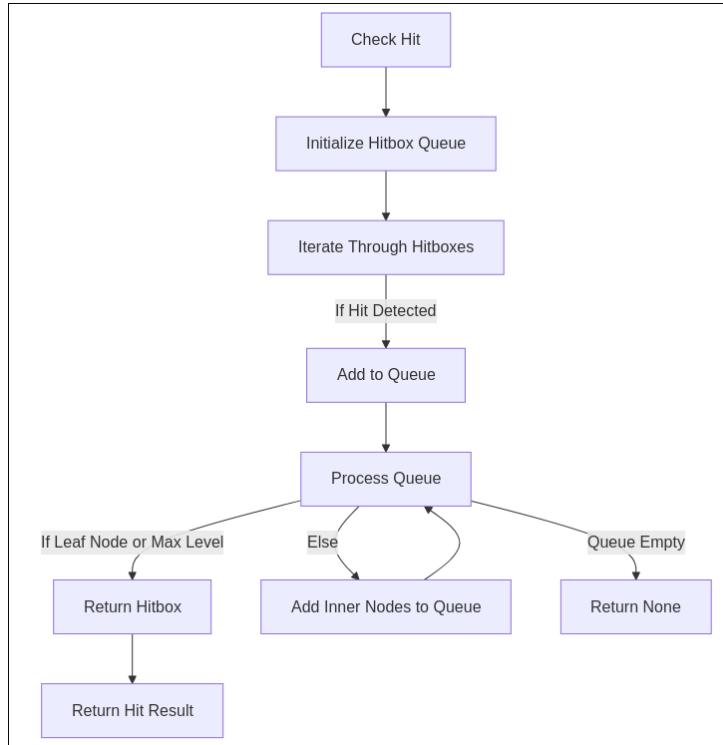


Abb. 122: Flussdiagramm-Auswahl

6.21.7 Verwendung einer Prioritätswarteschlange für Kollisionen

Da mehrere Objekte getroffen werden können, wird eine Prioritätswarteschlange (`BinaryHeap`) genutzt, um das nächstgelegene Objekt zu finden. Die Hitboxen werden nach Entfernung sortiert, sodass das nächstgelegene Objekt zuerst ausgewählt wird.

Es wird eine Prioritätswarteschlange für Hitboxen erstellt, die nach Entfernung sortiert ist. Die `HitBoxQueueEntry`-Struktur speichert die Hitbox, die Entfernung und die Ebene, auf der sich die Hitbox befindet. Listing 89 zeigt den entsprechenden Code:

Listing 89: Prioritätswarteschlange für Hitboxen

```

1 pub type HitboxQueue<M> = BinaryHeap<HitBoxQueueEntry<M>>;
2
3 #[derive(Debug)]
4 pub struct HitBoxQueueEntry<M> {
5     pub hitbox: Arc<M>,
6     pub distance: f32,
7     pub level: usize,
8 }
  
```

Die `BinaryHeap` speichert die Objekte somit nach steigender Entfernung, sodass das nächste Objekt zuerst ausgewählt werden kann.

6.22 Selektierer für Werkzeugpfade und Modelle

Die Struktur `Selector` dient der Auswahl und Transformation von 3D-Objekten in einer Szene. Sie speichert eine Liste von ausgewählten Objekten und stellt Methoden zur Manipulation und Anzeige der Selektion bereit (siehe Listing 90):

Listing 90: Selektierer für 3D-Objekte

```

1 pub struct Selector {
2     selected: Vec<Arc<dyn InteractiveModel>>,
3     grouped_transform: Option<Mat4>,
4
5     select_box: Model<Vertex>,
6     select_box_lines: Model<Vertex>,
7 }
```

6.22.1 Berechnung der Achsenausrichtenden Begrenzungsbox (AABB)

Jedes Objekt besitzt eine AABB (Axis-Aligned Bounding Box), die durch zwei Vektoren

$$\min = (x_{\min}, y_{\min}, z_{\min}), \quad \max = (x_{\max}, y_{\max}, z_{\max})$$

definiert wird und das jeweilige Objekt vollständig umschließt. Wenn mehrere Objekte ausgewählt sind, wird eine gemeinsame AABB berechnet. Die Gesamt-min- und max-Werte ergeben sich durch das Minimum bzw. Maximum aller Einzel-AABBS.

Diese Berechnung erfolgt iterativ in Listing 91:

Listing 91: AABB

```

1 self.selected.iter().fold(
2     (Vec3::splat(INFINITY), Vec3::splat(-INFINITY)),
3     |(min, max), model| {
4         let (model_min, model_max) = model.aabb();
5         (min.min(model_min), max.max(model_max))
6     }
7 )
```

6.22.2 Transformation von Objekten

Die Methode `transform` erlaubt die Modifikation von 3D-Objekten durch Skalierung, Rotation oder Translation mithilfe einer 4x4-Transformationsmatrix \mathbf{T} . Listing 92 zeigt den relevanten Ausschnitt:

Listing 92: Transformation von Objekten

```

1 let (scale, rotate, translate) =
2     model.transformation().to_scale_rotation_translation();
3
4 let transform = Mat4::from_scale_rotation_translation(
5     scale * grouped_scale,
6     rotate * grouped_rotate,
7     translate + grouped_translate,
8 );
```

6.22.3 Erstellung der Auswahlbegrenzung

Nach der Auswahl eines oder mehrerer Objekte wird eine Bounding-Box um die Selektion gezeichnet. Die Größe wird um einen konstanten Wert von 0.4 vergrößert:

$$\min_{\text{box}} = \min_{\text{gesamt}} - (0.4, 0.4, 0.4), \quad \max_{\text{box}} = \max_{\text{gesamt}} + (0.4, 0.4, 0.4)$$

Diese Werte werden genutzt, um eine Box zu erstellen, wie in Listing 93 gezeigt:

Listing 93: Erstellung der Auswahlbegrenzung

```
1 let select_box = SelectBox::from(BoundingBox::new(
2     min - Vec3::splat(0.4),
3     max + Vec3::splat(0.4),
4 ));
```

Abbildung 123 zeigt eine Visualisierung dieser Auswahlbegrenzungsbox:

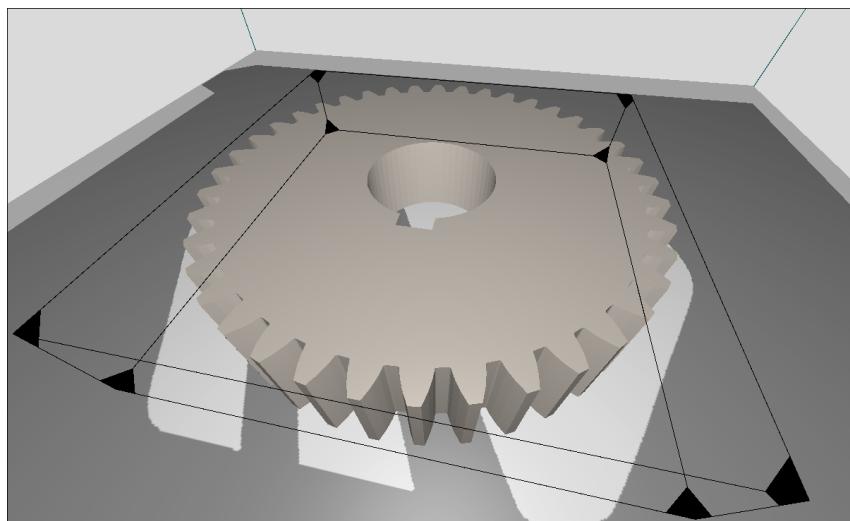


Abb. 123: Auswahl Begrenzungsbox Visualisierung

6.22.4 Löschen und Aktualisieren der Auswahl

Ein Objekt kann aus der Selektion entfernt werden, indem es mittels `deselect` aussortiert wird. Dabei wird geprüft, ob das Modell noch existiert, siehe Listing 94:

Listing 94: Löschen einer Selektion von Objekten

```
1 self.selected,retain(|model| !model.is_destroyed());
```

Nach dem Löschen wird die Selektion aktualisiert und die gemeinsame Bounding-Box neu berechnet, damit die Darstellung der Auswahlbegrenzung stets konsistent bleibt.

6.23 Slicer-Algorithmus

Das `slice`-Modul ist der Kern des Slicing-Prozesses. Wie in Listing 95 dargestellt, nimmt es ein 3D-Objekt, maskiert es optional, unterteilt es in Schichten und generiert anschließend eine Abfolge von Bewegungsbefehlen für den Druckprozess.

Listing 95: Hauptfunktion zum Slicen eines Objekts (Kernmodul) (Ince, 2021)

```

1 pub fn slice(
2     input: SliceInput<Mask>,
3     settings: &Settings,
4     process: &Process,
5 ) -> Result<SliceResult, SlicerErrors> {
6     ...
7
8     Ok(SliceResult {
9         moves,
10        calculated_values,
11        settings: settings.clone(),
12    })
13 }
```

Um den Slicing-Prozess zu veranschaulichen, wird ein Flussdiagramm verwendet, das die einzelnen Schritte des Algorithmus darstellt. Abbildung 124 zeigt den Ablauf des Slicing-Prozesses.

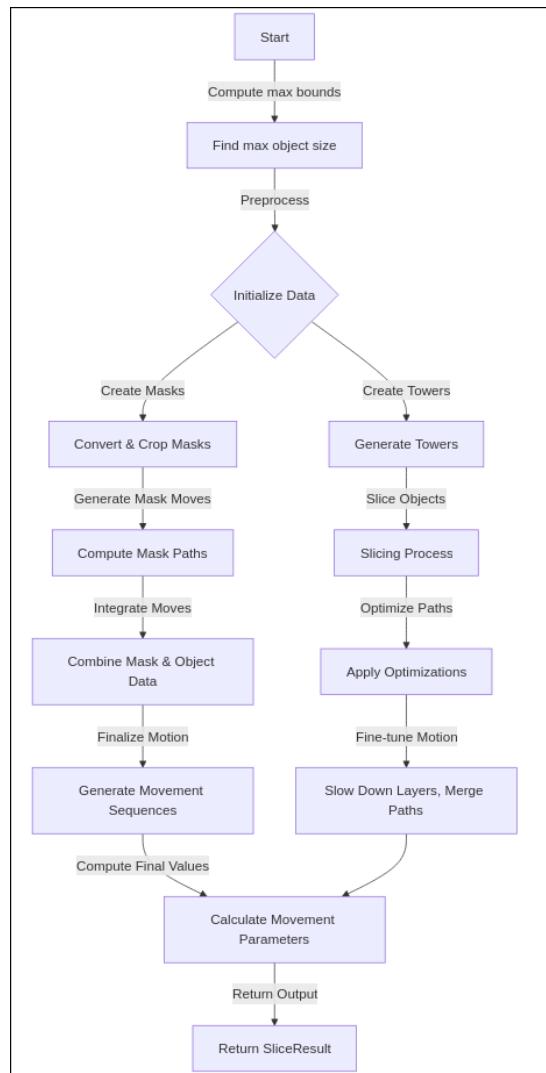


Abb. 124: Flussdiagramm-Slicing

6.23.1 Slicing: Zerlegung in horizontale Schichten

Der Slicing-Algorithmus teilt ein 3D-Objekt in horizontale Schichten auf und generiert daraus druckbare Pfade. Der Prozess basiert auf der Durchquerung eines *Triangle Towers*, wobei jede Schicht als eine Menge von Schnittpunkten mit den Dreiecken der Objektoberfläche berechnet wird.

1. Das 3D-Modell wird als eine Menge von Dreiecken gespeichert.
2. Ein *Triangle Tower* wird erstellt, in dem die Dreiecke nach Höhe sortiert sind.
3. Der Algorithmus iteriert durch die Höhen und berechnet die Schnittpunkte der Schicht mit den Dreiecken.
4. Diese Punkte werden zu Polygonen verknüpft, die die Schichtgeometrie definieren.
5. Die berechneten Schichten werden in eine Liste von `Slice`-Objekten gespeichert.

Wie in Listing 96 gezeigt, wird in dieser Variante ebenfalls eine `slice`-Funktion verwendet, allerdings liegt hier der Fokus auf dem Umgang mit dem `Triangle Tower` und den einzelnen Schichtobjekten.

Listing 96: Hauptfunktion zum Slicen eines Objekts (Schichtzerlegung) (Ince, 2021)

```

1 pub fn slice(
2     towers: &[TriangleTower],
3     max_height: f32,
4     settings: &Settings,
5 ) -> Result<Vec<Object>, SlicerErrors> {
6     towers
7         .iter()
8         .map(|tower| {
9             let mut tower_iter = TriangleTowerIterator::new(tower);
10            let mut layer = 0.0;
11
12            let res_points: Result<Vec<(f32, f32, Vec<Vec<ObjectVertex>>)>, SlicerErrors> = ...;
13            let points = res_points?;
14
15            let slices: Result<Vec<Slice>, SlicerErrors> = points
16                .par_iter()
17                .enumerate()
18                .map(|(count, (bot, top, layer_loops))| {
19                    ...
20                })
21                .collect();
22
23            Ok(Object { layers: slices? })
24        })
25        .collect()
26    }

```

6.23.2 Berechnung der Schnittpunkte einer Schicht mit Dreiecken

Ein wichtiger Schritt im Slicing-Prozess ist die Berechnung der Schnittpunkte einer Schicht mit den Dreiecken des Objekts. Jede Linie, die durch zwei Punkte v_{start} und v_{end} definiert ist, wird mit der Ebene z geschnitten, indem der Interpolationsfaktor f berechnet wird:

$$f = \frac{z - \mathbf{v}_{\text{start},z}}{\mathbf{v}_{\text{end},z} - \mathbf{v}_{\text{start},z}} \quad (31)$$

Dieser Faktor wird dann verwendet, um die x- und y-Koordinaten des Schnittpunkts zu berechnen:

$$x = \mathbf{v}_{\text{start},x} + f \cdot (\mathbf{v}_{\text{end},x} - \mathbf{v}_{\text{start},x}) \quad (32)$$

$$y = \mathbf{v}_{\text{start},y} + f \cdot (\mathbf{v}_{\text{end},y} - \mathbf{v}_{\text{start},y}) \quad (33)$$

Die zugehörige Implementierung ist in Listing 97 aufgeführt.

Listing 97: Berechnung des Schnittpunkts (Ince, 2021)

```

1 #[inline]
2 fn line_z_intersection(z: f32, v_start: ObjectVertex, v_end: ObjectVertex) -> ObjectVertex {
3     let z_normal = (z - v_start.z) / (v_end.z - v_start.z);
4     let y = lerp(v_start.y, v_end.y, z_normal);
5     let x = lerp(v_start.x, v_end.x, z_normal);
6     ObjectVertex::new(x, y, z)
7 }
8
9 #[inline]
10 fn lerp(a: f32, b: f32, f: f32) -> f32 {
11     a + f * (b - a)
12 }
```

6.23.3 Erstellung der Schnittpolygone

Sobald die Schnittpunkte berechnet werden, werden sie in geschlossene Polygone geordnet. Dabei wird ein `TriangleTowerIterator` verwendet, der sich schichtweise nach oben bewegt und die Ringe von Schnittpunkten verknüpft. Die Methode `advance_to_height` dazu ist in Listing 98 veranschaulicht.

Listing 98: Iterieren durch den Triangle Tower (Ince, 2021)

```

1 pub fn advance_to_height(&mut self, z: f32) -> Result<(), SlicerErrors> {
2     while self.tower.get_height_of_vertex(self.tower_vert_index) < z
3         && self.tower.tower_vertices.len() + 1 != self.tower_vert_index
4     {
5         let pop_tower_vert = self.tower.tower_vertices[self.tower_vert_index].clone();
6
7         self.active_rings = self
8             .active_rings
9             .drain(..)
10            .flat_map(|tower_ring| {
11                tower_ring
12                    .split_on_edge(pop_tower_vert.start_index)
13                    .into_iter()
14            })
15            .collect();
16
17         self.active_rings.extend(pop_tower_vert.next_ring_fragments);
18         join_fragments(&mut self.active_rings);
19         self.tower_vert_index += 1;
20
21         for ring in &self.active_rings {
22             if !ring.is_complete_ring() {
23                 return Err(SlicerErrors::TowerGeneration);
24             }
25         }
26     }
27
28     self.z_height = z;
29     Ok(())
30 }
```

6.23.4 Slice-Struktur und Slicing-Repräsentation

Die Slice-Struktur bildet das Fundament für die interne Repräsentation einer einzelnen Schicht (Layer) beim Slicing-Prozess. Wie in Listing 99 zu sehen ist, enthält sie sowohl geometrische Daten (z. B. die zu druckenden Polygone) als auch die dazugehörigen Bewegungs- und Einstellungselemente (MoveChains, Layer-Parameter).

Listing 99: Slice-Struktur zur Repräsentation einer Druckschicht (Ince, 2021)

```

1 #[derive(Debug)]
2 pub struct Slice {
3     pub main_polygon: MultiPolygon<f32>,
4     pub remaining_area: MultiPolygon<f32>,
5
6     pub support_interface: Option<MultiPolygon<f32>>,
7     pub support_tower: Option<MultiPolygon<f32>>,
8
9     pub fixed_chains: Vec<MoveChain>,
10    pub chains: Vec<MoveChain>,
11
12    pub bottom_height: f32,
13    pub top_height: f32,
14
15    pub layer_settings: LayerSettings,
16    pub layer: usize,
17 }
```

- **main_polygon:** Gesamtgeometrie der Schicht in Form eines MultiPolygon. Dieser Datensatz repräsentiert die Außen- und Innenkonturen (Lochränder) der Schicht.
- **remaining_area:** Restfläche der Schicht, die noch nicht bearbeitet wurde. Während verschiedener Verarbeitungsschritte (z. B. Perimeter-, Infill- oder Supporterstellung) wird aus **remaining_area** Teil für Teil subtrahiert, bis keine Fläche mehr zu füllen ist.
- **support_interface, support_tower:** Support-Bereiche, die speziell gekennzeichnet sind, um vom Slicer gesondert behandelt zu werden (z. B. Support-Interface für einfache Ablösbarkeit).
- **fixed_chains:** Feste Befehlsketten, die in genau der Reihenfolge abgearbeitet werden sollen (z. B. zwingend notwendige Bewegungen am Anfang jeder Schicht).
- **chains:** Allgemeine Befehlsketten, die im Zuge von Optimierungsschritten noch umgeordnet werden können (z. B. Sortierung von Infill-Pfaden für kürzere Fahrwege).
- **bottom_height, top_height:** Höhenangaben der Schicht. Die Schicht verläuft somit vertikal von **bottom_height** bis **top_height**.
- **layer_settings:** Einstellungen wie Druckgeschwindigkeit, Extrusionsbreite oder Materialangaben, die speziell für diese Schicht gelten.
- **layer:** Index, der die Position der Schicht (0 = unterste Schicht) kennzeichnet.

6.23.5 Erzeugung eines Slice

Um eine Schicht aus einer Punktfolge (Polygon) zu erstellen, stehen Methoden wie `from_single_point_loop` (Abschnitt 6.23.5) und `from_multiple_point_loop` (Abschnitt 6.23.5) zur Verfügung.

Implementierung von `from_single_point_loop`

Erzeugt einen `Slice` aus einer einzigen geschlossenen Kontur. Hierzu werden die Punkte ((`f32`, `f32`)) in ein `LineString` überführt und als `Polygon` gespeichert (Listing 100):

Listing 100: `Slice::from_single_point_loop` (Ince, 2021)

```

1 pub fn from_single_point_loop<I>(
2     ...
3 ) -> Self
4 where
5     I: Iterator<Item = (f32, f32)>,
6 {
7     let polygon = Polygon::new(LineString::from_iter(line), vec![]);
8     let layer_settings = settings.get_layer_settings(layer, (bottom_height + top_height) / 2.0);
9
10    Slice {
11        ...
12    }
13 }
```

1. Erzeugung eines `Polygon` aus dem `LineString`.
2. Aufruf von `settings.get_layer_settings`, um die passenden Parameter für die Schichthöhe zu ermitteln.
3. Setzen des `main_polygon` und `remaining_area`.

`from_multiple_point_loop`

Diese Funktion arbeitet ähnlich, verarbeitet jedoch ein `MultiLineString` und kann somit mehrere *Ringe* (Außenkontur und eventuell mehrere Innenkonturen) erstellen. Dabei unterscheidet der Code je nach Vorzeichen der Flächenberechnung, ob es sich um **Außenpolygon** (+) oder **Innenpolygon** (-) handelt und fügt Letzteres als *Loch* hinzu (siehe Listing 101):

Listing 101: `Slice::from_multiple_point_loop` (Ince, 2021)

```

1 pub fn from_multiple_point_loop(
2     ...
3 ) -> Result<Self, SlicerErrors> {
4     let mut lines_and_area: Vec<(LineString<f32>, f32)> = lines
5         .into_iter()
6         .map(|line| { ... })
7         .filter(|(_area)| area.abs() > 0.0001)
8         .collect();
9
10    let mut polygons = vec![];
11
12    for (line, area) in lines_and_area {
13        ...
14    }
15
16    let multi_polygon: MultiPolygon<f32> = MultiPolygon(polygons);
17
18    let layer_settings = settings.get_layer_settings(layer, (bottom_height + top_height) / 2.0);
19
20    Ok(Slice {
21        ...
22    })
23 }
```

Das resultierende `MultiPolygon` bildet die vollständige Fläche der Schicht ab, inklusive möglicher Innenkonturen.

6.23.6 Berechnung der Druckzeit

Die Gesamtzeit für den Druck wird nach Gleichung 34 durch die Summe der Bewegungszeit über alle Segmente berechnet:

$$T = \sum_{i=1}^N \frac{d_i}{v_i} \quad (34)$$

Die konkrete Implementierung zur Ausgabe der berechneten Druckzeit befindet sich in Listing 102.

Variablenbeschreibung

- T ist die Gesamtzeit.
- d_i ist die Länge des i -ten Bewegungssegments.
- v_i ist die Geschwindigkeit während dieses Segments.

Listing 102: Berechnung der Druckzeit (Ince, 2021)

```

1 impl CalculatedValues {
2     pub fn get_hours_minutes_seconds_fract_time(&self) -> (usize, usize, usize, f32) {
3         let total_time = self.total_time.floor() as usize;
4         let fract = self.total_time - total_time as f32;
5         (
6             total_time / 3600,
7             (total_time % 3600) / 60,
8             total_time % 60,
9             fract,
10        )
11    }
12 }
```

6.24 Slicer-Einstellungen

Die Slicing-Software verwendet eine Vielzahl an Einstellungen, um den Druckprozess optimal zu steuern. Diese Parameter beeinflussen verschiedene Aspekte des Drucks, darunter die Qualität, die Geschwindigkeit und die Materialverwendung. Die Einstellungen sind modular aufgebaut, sodass sie für spezifische Layer oder Druckbereiche angepasst werden können.

6.24.1 Grundlegende Einstellungen

Die grundlegenden Einstellungen umfassen Parameter wie die Schichthöhe (`layer_height`), die Extrusionsbreite (`extrusion_width`), die Druckgeschwindigkeit (`speed`) sowie die Beschleunigung (`acceleration`). Diese Werte bestimmen, wie der Slicer das Modell in Schichten zerlegt und die Bewegung des Druckkopfes steuert.

Beispiel einer Einstellung

Jede Einstellung wird in einer Struktur gespeichert und kann direkt im Code modifiziert werden. Ein Beispiel für eine wichtige Einstellung ist die Schichthöhe, wie in Listing 103 definiert:

Listing 103: Definition der Schichthöhe

```

1 pub struct Settings {
2     /// Die Hoehe jeder gedruckten Schicht in Millimetern
3     pub layer_height: f32,
4     ...
5 }
```

Die Schichthöhe beeinflusst maßgeblich die Qualität des Drucks. Eine kleinere Schichthöhe führt zu einer höheren Detailgenauigkeit, erhöht jedoch auch die Druckzeit. Eine größere Schichthöhe reduziert die Druckzeit, kann jedoch die Oberflächenqualität beeinträchtigen.

6.24.2 Erweiterte Einstellungen und Anpassung

Neben den grundlegenden Parametern bietet der Slicer die Möglichkeit, Einstellungen gezielt für bestimmte Druckbereiche oder Layer zu modifizieren. Dazu existiert ein Mechanismus zur Kombination von Einstellungen. Listing 104 zeigt, wie vordefinierte und benutzerspezifische Einstellungen zusammengeführt werden können.

Listing 104: Kombinieren von Einstellungen (Ince, 2021)

```

1 impl Settings {
2     pub fn combine_settings(self, mut settings: Settings) -> Settings {
3         if let Some(layer_height) = self.settings.layer_height {
4             settings.layer_height = layer_height;
5         }
6
7         ...
8
9         settings
10    }
11 }
```

Mit dieser Methode können beispielsweise für die ersten Layer eine langsamere Druckgeschwindigkeit und eine höhere Schichthöhe definiert werden, um eine bessere Haftung auf dem Druckbett zu erreichen.

6.24.3 Validierung von Einstellungen

Damit keine ungültigen Werte verwendet werden, existiert eine Validierungsfunktion. Diese überprüft, ob die gewählten Parameter innerhalb sinnvoller Grenzen liegen. Falls ein Wert ungültig ist, wird entweder eine Warnung oder ein Fehler ausgegeben. Ein Auszug ist in Listing 105 zu sehen:

Listing 105: Validierung von Einstellungen (Ince, 2021)

```

1 macro_rules! setting_less_than_or_equal_to_zero {
2     ($settings:ident, $setting:ident) => {{
3         if $settings.$setting as f32 <= 0.0 {
4             return SettingsValidationResult::Error(
5                 SlicerErrors::SettingLessThanOrEqualZero {
6                     setting: stringify!($setting).to_string(),
7                     value: $settings.$setting as f32,
8                 });
9     }
10    }};
11 }
```

Durch diese Validierung kann sichergestellt werden, dass zum Beispiel die Schichthöhe nicht negativ ist oder dass die Druckgeschwindigkeit innerhalb sinnvoller Grenzen liegt. Dies verhindert potenzielle Fehler während des Druckprozesses.

6.25 Plotter und Bewegungserzeugung

Der Plotter ist das zentrale Element für die Erzeugung der Druckpfade und Bewegungen des Druckkopfes. Er verarbeitet die berechneten Schichten (**Slices**) und wandelt sie in optimierte Bewegungssequenzen um. Im Folgenden werden die wichtigsten Funktionen im Detail beschrieben.

6.25.1 Bewegungsplanung und Druckpfade

Die folgenden Strukturen repräsentieren die grundlegenden Bewegungsabläufe des Druckkopfes. Eine **Move**-Struktur steht für eine einzelne Bewegung mit Extrusionseigenschaften (Breite, Typ), während **MoveChain** mehrere dieser **Move**-Kommandos in einer festgelegten Reihenfolge enthält und zusätzliche Informationen (Startpunkt, Schleifen-Flag) speichert. Listing 106 und 107 verdeutlichen dies.

Listing 106: Move-Struktur mit grundlegenden Bewegungsinformationen (Ince, 2021)

```

1 #[derive(Clone, Copy, Debug, PartialEq)]
2 pub struct Move {
3     pub end: Coord<f32>,
4     pub width: f32,
5     pub move_type: MoveType,
6 }
```

- **end**: Zielkoordinate der Bewegung. Der Startpunkt wird implizit durch das Ende der vorangegangenen Bewegung bestimmt.
- **width**: Definiert die Breite des auszubringenden Materials (z. B. zur Festlegung der Extrusionsspur im Druckprozess).
- **move_type**: Spezifiziert, ob es sich um eine „normale“ Bewegung (ohne Faser), eine Faserbewegung oder einen Reiseweg ohne Extrusion handelt.

Listing 107: MoveChain als Gruppe zusammenhängender Bewegungen (Ince, 2021)

```

1 #[derive(Debug)]
2 pub struct MoveChain {
3     pub start_point: Coord<f32>,
4     pub moves: Vec<Move>,
5
6     pub is_loop: bool,
7 }
```

- **start_point**: Anfangskoordinate der Kette; da eine einzelne **Move**-Struktur nur das Ziel enthält, wird der tatsächliche Startpunkt der gesamten Kette hier definiert.
- **moves**: Liste von **Move**-Strukturen, die nacheinander abgearbeitet werden.
- **is_loop**: Legt fest, ob die Kette geschlossen ist. In diesem Fall könnte der Start für die Ausführung beliebig verschoben werden (etwa zum „Naht“-Optimieren).

6.25.2 Erzeugung von Perimetern

Die äußeren und inneren Umrandungen eines Objekts werden als **Perimeter** bezeichnet. Diese werden durch die Funktion `slice_perimeter_into_chains` erstellt (siehe Listing 108). Dabei werden die Schichten so bearbeitet, dass sie die gewünschte Anzahl an Perimetern erhalten.

Listing 108: Erzeugung der Perimeter (Ince, 2021)

```

1 fn slice_perimeter_into_chains(
2     ...
3 ) {
4     let mut new_chains = self
5         .remaining_area
6         .iter()
7         .map(|poly| MultiPolygon(vec![poly.clone()]))
8         .filter_map(|multi| {
9             inset_polygon_recursive(
10                 &multi,
11                 &self.layer_settings,
12                 true,
13                 number_of_perimeters,
14                 number_of_perimeters - 1,
15                 layer,
16                 perimter_ranges,
17             )
18         })
19         .collect::<Vec<_>>();
20
21     self.fixed_chains.append(&mut new_chains);
22 }
```

Wie in Listing 108 ersichtlich, verkleinert diese Methode die ursprüngliche Polygonform rekursiv um die vorgegebene Anzahl an Perimetern, wodurch die äußeren und inneren Begrenzungslinien des Objekts entstehen.

6.25.3 Bestimmung des Bewegungstyps in der Perimetergenerierung

Jede Bewegung innerhalb der Perimeter kann entweder mit oder ohne Faser erfolgen (8.1.5 *Software*, S.202). Die Funktion `determine_move_type` (siehe Listing 109) bestimmt anhand der aktuellen Schicht-einstellungen, ob eine Bewegung mit oder ohne Faser ausgeführt wird.

Listing 109: Bestimmung des Bewegungstyps

```

1 pub fn determine_move_type(
2     ...
3 ) -> MoveType {
4     if settings.fiber.perimter_pattern.is_enabled() {
5         let perimter = if !perimter_ranges.is_empty() {
6             ...
7         } else {
8             Some(number_of_perimters + 1 - perimter)
9         };
10
11     if let Some(perimeter) = perimter {
12         ...
13     } else {
14         MoveType::WithoutFiber(trace_type)
15     }
16 } else {
17     MoveType::WithoutFiber(trace_type)
18 }
19 }
```

Diese Funktion (Listing 109) analysiert das aktuelle Layer-Setup und entscheidet anhand des definierten Perimeter-Musters, ob eine Bewegung mit oder ohne Faser erfolgen soll.

6.25.4 Optimale Positionierung des Nahtpunkts (Seam)

Die Funktion `seam` berechnet die optimale Startposition für das Zeichnen einer Wandlinie. Dies ist wichtig, um sichtbare Nahtstellen (*Seams*) an günstigen Positionen zu platzieren. Listing 110 zeigt die Implementierung dazu.

Listing 110: Bestimmung der Nahtstelle

```

1 pub fn seam<'a>(points: &'a [Coord<f32>]) -> Vec<&'a Coord<f32>> {
2     ...
3
4     for (i, end) in points.iter().enumerate() {
5         let start = vec2(start.x, start.y);
6         let end = vec2(end.x, end.y);
7         let direction = end - start;
8
9         if let Some(last_direction) = last_direction {
10             ...
11         } else {
12             index = Some(i);
13         }
14
15         last_direction = Some(direction);
16     }
17
18     let wanted_start_index = index.unwrap_or(0);
19
20     points
21         .iter()
22         .cycle()
23         .skip(wanted_start_index)
24         .take(points.len())
25         .collect()
26 }
```

Wie in Listing 110 ersichtlich, wird hier sichergestellt, dass die Nahtstellen des Drucks an den am wenigsten sichtbaren Stellen platziert werden.

6.25.5 Partielle vs. vollständige Flächenfüllung

Diese beiden Funktionen (Listing 111 und Listing 112) gehören zum `Plotter`-Trait und sind entscheidend für die Implementierung der Faserverstärkung in Kombination mit separatem Faserinfill. Der Hauptunterschied zwischen den beiden Funktionen liegt in der Art und Weise, wie der verbleibende Infill-Bereich berechnet und subtrahiert wird:

- `fill_remaining_area_partially`: Berechnet nur den tatsächlich von den Infill-Traces genutzten Bereich und subtrahiert nur diesen.
- `fill_remaining_area`: Erzeugt ein Infill und subtrahiert die gesamte Fläche.

Implementierung von `fill_remaining_area_partially`

Diese Funktion (siehe Listing 111) füllt die verbliebene Fläche mit Faserinfill, wobei sie nur die tatsächlich belegten Bereiche von der ursprünglichen Fläche subtrahiert.

Listing 111: Funktion zur partiellen Flächenfüllung

```

1 fn fill_remaining_area_partially(
2     &mut self,
3     layer_count: usize,
4     fill_ratio: f32,
5     ctx: &PassContext,
6 ) {
7     let mut remaining_polygons = vec![];
8
9     for poly in self.remaining_area.iter() {
10
11         let trace_polygons: Vec<Polygon<f32>> = ...;
12
13         for chain in new_moves {
14             self.chains.push(chain);
15         }
16
17         // Subtrahiere nur die von den Traces belegten Flächen
18         ...
19     }
20
21     self.remaining_area = MultiPolygon(remaining_polygons);
22 }
```

Implementierung von `fill_remaining_area`

In Listing 112 wird die gesamte Fläche mit normalem oder solidem Infill gefüllt. Anschließend wird die gesamte Fläche subtrahiert, sodass kein weiterer Infill möglich ist.

Listing 112: Funktion zur vollständigen Flächenfüllung

```

1 fn fill_remaining_area(&mut self, solid: bool, layer_count: usize, ctx: &PassContext) {
2     for poly in &self.remaining_area {
3         if solid {
4             ...
5         } else {
6             let fill_ratio = if ctx.is_fiber() {
7                 self.layer_settings.fiber.infill.infill_percentage
8             } else {
9                 self.layer_settings.infill.infill_percentage
10            };
11
12            let new_moves = ...;
13
14            for chain in new_moves {
15                self.chains.push(chain);
16            }
17        }
18    }
19
20    // Subtrahiere die gesamte Fläche
21    self.remaining_area = MultiPolygon(vec![]);
22 }
```

Hinweis

Die Funktion `fill_remaining_area_partially` (Listing 111) wird in der Regel für die Faserverlegung verwendet, während `fill_remaining_area` (Listing 112) für normales Infill zuständig ist.

6.25.6 Ermittlung der Spurfläche

Die Methode `trace_area` (Listing 113) erzeugt eine Sammlung von Polygonen (`MultiPolygon<f32>`), die die exakte Spur der gedruckten Linie abbilden. Dabei wird für jeden `Move` ein Viereck um den Bewegungsvektor aufgespannt, dessen Breite sich aus dem `width`-Attribut ergibt.

Listing 113: Berechnung der Spurflächen mittels `trace_area`

```

1 pub fn trace_area(&self) -> MultiPolygon<f32> {
2     let mut polygons = vec![];
3     let mut current_loc = self.start_point;
4
5     for m in self.moves.iter() {
6         let end = m.end;
7         let end = vec2(end.x, end.y);
8
9         let start = vec2(current_loc.x, current_loc.y);
10
11        // normalisierter Richtungsvektor
12        let direction = (end - start).normalize();
13
14        // Eckpunkte um die Bewegungsbahn
15        let p1 = start + vec2(direction.x, -direction.y) * (m.width / 2.0);
16        ...
17
18        let line = line_string![
19            ...
20        ];
21
22        polygons.push(Polygon::new(line, vec![]));
23
24        // Der Endpunkt wird zum neuen Startpunkt fuer den naechsten Move
25        current_loc = m.end;
26    }
27
28    MultiPolygon(polygons)
29 }
```

Für jeden `Move` entsteht somit ein Rechteck (teilweise Trapez, falls die Breite verändert wird) mit folgender Logik:

1. Berechnung des Richtungsvektors:

$$\mathbf{d} = (\text{end.x} - \text{start.x}, \text{end.y} - \text{start.y}) \quad (35)$$

2. Normierung dieses Vektors zur Ermittlung der Bewegungsrichtung $\mathbf{dir} = \mathbf{d}/\|\mathbf{d}\|$.
3. Erzeugung der „Offset“-Vektoren in orthogonaler Richtung $\pm(-\mathbf{dir.y}, \mathbf{dir.x})$.
4. Aufspannen eines `line_string` mit fünf Punkten (inkl. Schließen des Polygons).

Wie in Gleichung (35) und Listing 113 beschrieben, liegt damit für jede Bewegung (`Move`) die tatsächlich bedruckte Fläche vor. Dies ermöglicht unter anderem die Berechnung von Überlappungen oder die Erstellung komplexer Support-Strukturen, da die exakten Spurgeometrien für weitere geometrische Analysen genutzt werden können.

6.26 Zusammenfassen von Faserbewegungen

Der `MergeFiberPass` durchsucht die komplette Liste von Druckbefehlen (`cmds`) nach Faserbewegungen (`MoveAndExtrudeFiber`) und entscheidet, ob diese Ketten

- zu kurz sind und in normale `MoveAndExtrude`-Befehle umgewandelt werden,
- ausreichend lang sind und deshalb an einer bestimmten Stelle geschnitten werden (`MoveAndExtrudeFiberAndCut`),
- oder nur aus einem einzigen Befehl bestehen.

Die Hauptfunktion `pass` (siehe Listing 114) iteriert über alle Befehle und versucht, mit `FiberChain::find_next` eine Faserkette aus aufeinanderfolgenden `MoveAndExtrudeFiber`-Kommandos zu bilden. Anschließend übernimmt eine von drei Hilfsfunktionen die konkrete Weiterverarbeitung der ermittelten Kette:

Listing 114: Struktur des `MergeFiberPass`

```

1 fn pass(cmds: &mut Vec<Command>, settings: &crate::Settings) {
2     let mut current_index = 0;
3
4     while current_index < cmds.len() {
5         if let Some(chain) = FiberChain::find_next(cmds, current_index, settings) {
6             if chain.start_index == chain.end_index {
7                 handle_single_move(cmds, &chain, settings);
8             } else if chain.length >= settings.fiber.min_length {
9                 chain.find_cut_and_set(cmds, settings.fiber.cut_before);
10            } else {
11                convert_chain_to_normal(cmds, &chain);
12            }
13
14            current_index = chain.end_index + 1;
15        } else {
16            current_index += 1;
17        }
18    }
19 }
```

6.26.1 Einzelne Faserbewegungen verarbeiten

Wenn die Faserkette aus genau einem Befehl besteht, wird `handle_single_move` aufgerufen, um zu entscheiden, ob die Kette zu kurz ist und in normale Bewegungen umgewandelt werden muss (siehe Listing 115).

Listing 115: `handle_single_move`-Funktion

```

1 fn handle_single_move(cmds: &mut Vec<Command>, chain: &FiberChain, settings: &crate::Settings) {
2     let (start, end, thickness, width) = match cmdms[chain.start_index] {
3         Command::MoveAndExtrudeFiber {
4             ...
5         } => (start, end, thickness, width),
6         _ => unreachable!(),
7     };
8
9     // Laenge pruefen
10    if chain.length >= settings.fiber.min_length {
11        cmdms[chain.start_index] = Command::MoveAndExtrudeFiberAndCut {
12            ...
13        };
14    } else {
15        cmdms[chain.start_index] = Command::MoveAndExtrude {
16            ...
17        };
18    }
19 }
```

Ist die Länge der Kette groß genug, wird in Listing 115 ein `MoveAndExtrudeFiberAndCut`-Befehl erzeugt. Andernfalls wird der Befehl zu `MoveAndExtrude` herabgestuft.

6.26.2 Faserbewegungen in normale Bewegungen umwandeln

Sofern die gesamte Kette zu kurz ist, werden alle `MoveAndExtrudeFiber`-Befehle in normale `MoveAndExtrude`-Befehle umgewandelt (siehe Listing 116).

Listing 116: `convert_chain_to_normal`-Funktion

```

1 fn convert_chain_to_normal(cmds: &mut Vec<Command>, chain: &FiberChain) {
2     for i in chain.start_index..=chain.end_index {
3         let (start, end, thickness, width) = match cmd[i] {
4             Command::MoveAndExtrudeFiber {
5                 ...
6                 } => (start, end, thickness, width), _ => unreachable!(),
7             };
8
9         cmd[i] = Command::MoveAndExtrude {
10            ...
11        };
12    }
13 }
```

6.26.3 FiberChain-Struktur und deren Methoden

Die `FiberChain`-Struktur repräsentiert die Indizes einer Kette von Faserbefehlen und speichert zusätzlich die aufaddierte Länge (siehe Listing 117). Die Methode `find_next` (Listing 118) durchsucht die Befehlsliste ab `current_index` und sammelt Bewegungen, solange sie den Winkelkriterien entsprechen (siehe `max_angle` in den Einstellungen).

Listing 117: `FiberChain`-Struktur

```

1 #[derive(Debug, Clone)]
2 struct FiberChain {
3     start_index: usize,
4     end_index: usize,
5     length: f32,
6 }
```

Es wird geprüft, ob der Winkel θ zwischen zwei aufeinanderfolgenden Richtungsvektoren \mathbf{d}_1 und \mathbf{d}_2 unter einer bestimmten Schwelle liegt (z. B. `max_angle`). Mathematisch lässt sich dieser Winkel über das Skalarprodukt zweier normalisierter Vektoren bestimmen (siehe Gleichung 36):

$$\theta = \arccos\left(\frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \|\mathbf{d}_2\|}\right). \quad (36)$$

Da beide Vektoren normalisiert werden (`.normalize()`), reduziert sich der Divisor $\|\mathbf{d}_1\| \|\mathbf{d}_2\|$ auf 1. Um den Winkel in Grad zu erhalten, wird er zusätzlich mit $\frac{180}{\pi}$ multipliziert (siehe Gleichung 37):

$$\theta_{\text{deg}} = \theta_{\text{rad}} \times \frac{180}{\pi}. \quad (37)$$

Im Anschluss wird überprüft, ob dieser Winkel θ_{deg} (in Grad) kleiner oder gleich `settings.fiber.max_angle` ist. Liegt θ_{deg} innerhalb dieses Grenzwertes, werden die betreffenden Bewegungen in der Faserkette zusammengefasst.

Listing 118: `FiberChain::find_next`

```

1 fn find_next(
2     ...
3 ) -> Option<FiberChain> {
4     let start_index = current_index;
5     let mut last_direction = None;
6     let mut length = 0.0;
7
8     while current_index < cmds.len() {
9         match cmds[current_index] {
10             Command::MoveAndExtrudeFiber { start, end, ... } => {
11                 let direction = vec2(end.x - start.x, end.y - start.y).normalize();
12                 ...
13                 // Pruefung des Winkels und Hinzufuegen zur Kette
14             }
15             _ => {
16                 // Kette beenden, wenn kein Faserbefehl
17                 if start_index == current_index {
18                     return None;
19                 } else {
20                     return Some(FiberChain {
21                         start_index,
22                         end_index: current_index - 1,
23                         length,
24                         });
25                 }
26             }
27         }
28     }
29 }
```

Im Listing 118 wird die `find_next`-Methode beschrieben, die eine Kette von Faserbewegungen findet und deren Länge berechnet. Diese Methode wird in der `pass`-Funktion des `MergeFiberPass` aufgerufen, um die Ketten zu identifizieren.

6.26.4 Rückwärtssuche des Schnittpunkts

Hat `MergeFiberPass` eine genügend lange Kette gefunden, wird `find_cut_and_set` aufgerufen (siehe Listing 119). Diese Methode durchläuft die Kette rückwärts und prüft, an welcher Faserbewegung die Schnittstelle erreicht oder überschritten wird. Dieser Befehl wird anschließend zu `MoveAndExtrudeFiberAndCut` abgeändert.

Listing 119: `FiberChain::find_cut_and_set`

```

1 fn find_cut_and_set(&self, cmds: &mut [Command], cut_before: f32) {
2     let mut distance_backtraced = 0.0;
3
4     for i in (self.start_index..=self.end_index).rev() {
5         match cmds[i] {
6             Command::MoveAndExtrudeFiber { ... } => {
7                 distance_backtraced += start.euclidean_distance(&end);
8
9                 if distance_backtraced >= cut_before {
10                     ...
11                 }
12             }
13             _ => {}
14         }
15     }
16 }
```

6.27 Maskierungsfunktionen für das Slicing

Die Maskierungsfunktionen dienen dazu, Bereiche innerhalb eines 3D-Modells gezielt zu beschneiden oder zu beeinflussen. Dies ermöglicht unter anderem das Ausschneiden von bestimmten Regionen für Faserverstärkung oder das Entfernen unerwünschter Bereiche. Abbildung 125 und Abbildung 126 zeigen eine beispielhafte Anwendung.

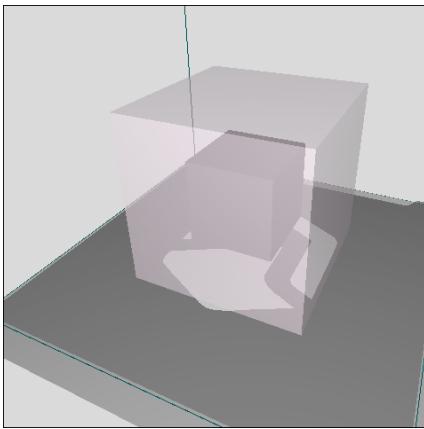


Abb. 125: Maskierungsfunktion im Prepare-Mode

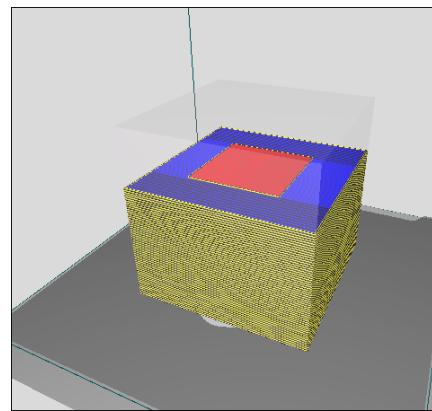


Abb. 126: Maskierungsfunktion im Preview-Mode

6.27.1 Mask-Erstellung und Transformation

Die Struktur `Mask` verwaltet ein Mesh-Objekt mit den zugehörigen Einstellungen und bietet Funktionen zur Transformation und Umwandlung in ein maskiertes Objekt (siehe Listing 120).

Listing 120: Erstellung einer neuen Maskierung

```

1 pub fn new(mesh: ObjectMesh, settings: MaskSettings) -> Self {
2     Self { mesh, settings }
3 }
```

Die Maske kann über eine Transformationsmatrix verändert werden, um das Mesh in eine neue Position oder Skalierung zu versetzen.

6.27.2 Beschneiden des maskierten Objekts

Die Methode `crop` (siehe Listing 121) ermöglicht das gezielte Entfernen von Bereichen innerhalb des maskierten Objekts. Dies geschieht durch die Differenzbildung zwischen den Polygonen der aktuellen Maske und den bestehenden Objekten.

Listing 121: Beschneiden der Maske anhand vorhandener Objekte

```

1 pub fn crop(&mut self, objects: &[Object], max: Vec3) {
2     self.layers
3         .iter_mut()
4         .enumerate()
5         .for_each(|(index, layer)| {
6             let mut remaining_polygon = layer.main_polygon.clone();
7             for object in objects.iter() {
8                 if let Some(layer) = object.layers.get(index) {
9                     if layer.main_polygon.is_empty() {
10                         info!("Layer is empty, skipping {}", index);
11                         continue;
12                     }
13                     remaining_polygon =
14                         remaining_polygon
15                             .difference_with(&layer.main_polygon.simplify(&0.2));
16                 }
17             }
18             layer.main_polygon =
19                 layer.main_polygon.difference_with(&remaining_polygon);
20             layer.remaining_area = layer.main_polygon.clone();
21         });
22     });
23
24     self.layers.retain(|layer| {
25         layer.main_polygon.unsigned_area() > f32::EPSILON || layer.top_height <= max.z
26     });
27 }
```

Die Funktion `crop` führt folgende Schritte aus:

- Iteration über alle Schichten der Maske.
- Differenzbildung zwischen der Maske und den Objektschichten.
- Entfernen von Bereichen, die innerhalb der maskierten Flächen liegen.
- Beibehaltung nur relevanter Schichten mit signifikanten Bereichen.

6.27.3 Zufällige Variation der Maskenbereiche

Zur Optimierung von Maskierungsübergängen kann eine zufällige Variation der Maskierung angewandt werden (siehe Listing 122). Diese Methode reduziert scharfe Kanten oder unnatürliche Schnittstellen zwischen maskierten und nicht-maskierten Bereichen.

Listing 122: Zufällige Variation der Maskierung

```

1 pub fn randomize_mask_underlaps(&mut self, epsilon: f32) {
2     self.layers.iter_mut().for_each(|layer| {
3         let inset: f32 = rand::random::f32() * epsilon;
4
5         layer.main_polygon = layer.main_polygon.offset_from(-inset);
6         layer.remaining_area = layer.main_polygon.clone();
7     });
8 }
```

6.28 Druckpfadtyp und Farbzordnung

Der `TraceType` bestimmt, welcher Bereich innerhalb eines gedruckten Objekts gerade erzeugt wird (z. B. Infill, Perimeter, Support). Er dient somit als Klassifizierung verschiedener Druckpfade (Slicing-Ergebnisse). Verschiedene Methoden (z. B. `into_color_vec4`) erlauben eine einfache Zuordnung zu Farbcodes, etwa zur visuellen Darstellung in einer Vorschau (siehe Listing 123).

Listing 123: `TraceType`-Enum zur Kategorisierung von Druckpfaden

```

1 #[derive(Serialize, Deserialize, Clone, Copy, Debug, PartialEq, Eq, Hash, EnumCount)]
2 pub enum TraceType {
3     TopSolidInfill,
4     SolidInfill,
5     Infill,
6     PerimeterOuter,
7     PerimeterInner,
8     InteriorPerimeterOuter,
9     InteriorPerimeterInner,
10    Bridging,
11    Support,
12 }
```

6.28.1 Farbauswahl (`into_color_vec4`)

Mit `into_color_vec4` lässt sich jedem `TraceType` eine spezifische RGBA-Farbe (als `Vec4`) zuordnen. Dies ist nützlich, um unterschiedliche Bereiche in einer Druckvorschau oder einem *Slicer*-UI deutlich erkennbar zu machen.

6.28.2 MoveType-Integration

In der `MoveType`-Struktur (bzw. dem `Enum`) wird unterschieden, ob ein Pfad „mit“ oder „ohne“ Faser gedruckt wird. Mithilfe von `TraceType` kann das Slicer-System entsprechend unterschiedliche Einstellungen (z. B. Geschwindigkeiten, Extrusionsraten) anwenden.

- `MoveType::WithFiber(_)`: Kennzeichnet Pfade, die Faserextrusion beinhalten.
- `MoveType::WithoutFiber(_)`: Kennzeichnet herkömmliche „Plastik-Only“-Pfade.

6.29 Command-Aufbau und Aufgaben

Die **Command**-Enumeration bildet das interne Protokoll für Bewegungs-, Zustands- und Konfigurationsbefehle im Slicer (siehe Listing 124). Jedes **Command** beschreibt, was der Drucker als Nächstes tun soll, etwa eine Bewegung, eine Schichtänderung oder das Ändern von Parametern wie Temperatur oder Geschwindigkeiten.

Listing 124: Command-Enumeration

```

1 #[derive(Serialize, Deserialize, Clone, Debug, PartialEq)]
2 pub enum Command {
3     MoveTo {
4         end: Coord<f32>,
5     },
6     MoveAndExtrude {
7         id: Option<MoveId>,
8         start: Coord<f32>,
9         end: Coord<f32>,
10        thickness: f32,
11        width: f32,
12    },
13    MoveAndExtrudeFiber {
14        ...
15    },
16    MoveAndExtrudeFiberAndCut {
17        cut_pos: f32,
18        ...
19    },
20    LayerChange {
21        z: f32,
22        index: usize,
23    },
24    SetState {
25        new_state: StateChange,
26    },
27    Delay {
28        msec: u64,
29    },
30    ChangeObject { ... },
31    ChangeType { ... },
32    NoAction,
33 }
```

6.29.1 Move-Varianten mit Extrusion

- **MoveAndExtrude**: Repräsentiert eine lineare Bewegung (`start` → `end`) mit konventioneller Extrusion (ohne Faser).
- **MoveAndExtrudeFiber**: Ähnlich wie **MoveAndExtrude**, allerdings für Faserextrusion vorgesehen.
- **MoveAndExtrudeFiberAndCut**: Kombination einer Faserbewegung mit anschließendem Schnitt an der Position `cut_pos`.

6.29.2 Weitere Bewegungsbefehle

- **MoveTo**: Bewegt den Druckkopf ohne Extrusion auf `end`.

6.29.3 Systembefehle

- **LayerChange**: Informiert den Slicer oder den Druckprozess, dass eine neue Schicht beginnt (z. B. z-Position + Schichthöhe).
- **SetState**: Ändert die Drucker-Parameter (**StateChange**), z. B. `extruder_temp` oder `fan_speed`.
- **Delay**: Legt eine `msec`-lange Wartezeit ein, etwa zum Aufheizen des Extruders.
- **ChangeObject**: Wechselt das zu druckende Objekt, wenn mehrere Modelle nacheinander erstellt werden.
- **ChangeType**: Ändert den aktuellen **TraceType** (z. B. von `Infill` zu `PerimeterOuter`).
- **NoAction**: Dient als Platzhalter für Optimierungen.

6.29.4 Filament- und Faser-Abfrage

Das `Command`-Enum bietet Hilfsfunktionen (siehe Listing 125), um zu ermitteln, ob ein Befehl Filament oder Faser benötigt:

Listing 125: Hilfsfunktionen zur Erkennung von Filament- und Faserbedarfen

```

1 impl Command {
2     pub fn needs_filament(&self) -> bool {
3         match self {
4             Command::MoveAndExtrude { .. } => true,
5             Command::MoveAndExtrudeFiber { .. } => true,
6             Command::MoveAndExtrudeFiberAndCut { .. } => true,
7             _ => false,
8         }
9     }
10
11    pub fn needs_fiber(&self) -> bool {
12        match self {
13            Command::MoveAndExtrudeFiber { .. } => true,
14            Command::MoveAndExtrudeFiberAndCut { .. } => true,
15            _ => false,
16        }
17    }
18 }
```

6.30 Generierung und Speicherung von G-Code

Die Generierung und Speicherung von G-Code ist ein wichtiger Bestandteil des Slicing-Prozesses. Mit „Generierung“ ist hier die Umwandlung der Druckpfade in eine lesbare G-Code-Datei gemeint. Die Speicherung erfolgt entweder in einem Speicherpuffer oder direkt in einer Datei (Listing 126).

6.30.1 Datenstruktur für G-Code

Die Struktur `SlicedGCode` speichert den generierten G-Code als Zeichenkette und enthält zusätzliche Metadaten, wie Zeilenumbrüche und Navigationsinformationen.

Listing 126: Struktur für den generierten G-Code

```

1 #[derive(Debug)]
2 pub struct SlicedGCode {
3     pub gcode: String,
4     pub line_breaks: Vec<usize>,
5     pub navigator: Navigator,
6 }
```

- Speichert den G-Code als `String`.
- Hält eine Liste von Zeilenumbrüchen zur schnellen Navigation.
- Enthält eine `Navigator`-Instanz zur Verwaltung der Code-Struktur.

Die Methode `new` analysiert den G-Code und speichert die Positionen der Zeilenumbrüche (siehe Listing 127):

Listing 127: Erstellung eines neuen G-Code Objekts

```

1 pub fn new(gcode: String, navigator: Navigator) -> Self {
2     let line_breaks = gcode
3         .char_indices()
4         .filter_map(|(i, c)| if c == '\n' { Some(i) } else { None })
5         .collect();
6
7     Self {
8         gcode,
9         line_breaks,
10        navigator,
11    }
12 }
```

6.30.2 Schreiben von G-Code

Die Trait-Implementierung `WriteGCode` erweitert die Funktionalität des Rust-`Write`-Traits um eine Methode zur Ermittlung der Zeilenanzahl (siehe Listing 128).

Listing 128: Erweiterung des Write-Traits für G-Code

```
1 pub trait WriteGCode: Write {
2     fn line_count(&self) -> usize;
3 }
```

Es gibt zwei Implementierungen für das Schreiben von G-Code:

1. `GCodeMemoryWriter`: Schreibt den G-Code in einen `String`-Puffer (siehe Listing 129).
2. `GCodeFileWriter`: Schreibt den G-Code direkt in eine Datei (siehe Listing 130).

Listing 129: Schreiben von G-Code in den Speicher

```
1 pub struct GCodeMemoryWriter {
2     buffer: String,
3     line_count: usize,
4 }
5
6 impl WriteGCode for GCodeMemoryWriter {
7     fn line_count(&self) -> usize {
8         self.line_count
9     }
10 }
```

Listing 130: Schreiben von G-Code in eine Datei

```
1 pub struct GCodeFileWriter<T: Write> {
2     writer: T,
3     line_count: usize,
4 }
5
6 impl<T: Write> WriteGCode for GCodeFileWriter<T> {
7     fn line_count(&self) -> usize {
8         self.line_count
9     }
10 }
```

6.30.3 Generierung von G-Code

Die Funktion `write_gcode` (siehe Listing 131) generiert den eigentlichen G-Code aus einer Liste von `Command`-Strukturen.

Listing 131: Generierung von G-Code

```

1 pub fn write_gcode(
2     cmd: &[Command],
3     settings: &Settings,
4     writer: &mut dyn WriteGCode,
5 ) -> Result<Navigator, Box<dyn std::error::Error>> {
6     ...
7 }
```

Ablauf

1. Initialisierung des `Navigator`-Objekts zur Verwaltung der G-Code-Struktur.
2. Schreiben der Startparameter (`M201`, `M203`, `M204`, etc.).
3. Verarbeitung aller `Command`-Befehle, die Bewegungen, Extrusionen und Statusänderungen repräsentieren.
4. Abschließen des G-Codes mit Endanweisungen.

6.30.4 Bewegungen ohne Extrusion

Der Programmausdruck Listing 132 zeigt den Code, der die Bewegung ohne Extrusion steuert. Die Funktion wird durch den Befehl `Command::MoveTo` ausgelöst und schreibt die Bewegung in den G-Code:

Listing 132: Bewegung ohne Extrusion

```

1 Command::MoveTo { end, .. } =>
2     writeln!(writer, "G1 X{:.5} Y{:.5}", end.x, end.y)?;
```

6.30.5 Bewegungen mit Extrusion

Das Listing 133 zeigt den Code, der die Bewegung und Extrusion der Faser steuert. Die Funktion wird durch den Befehl `Command::MoveAndExtrude` ausgelöst und berechnet die Extrusionsmenge anhand des Volumens der Extrusion. Diese wird dann in den G-Code geschrieben:

Listing 133: Bewegung mit Extrusion

```

1 Command::MoveAndExtrude {
2     start, end, width, thickness, ..
3 } => {
4     let length =
5         ((end.x - start.x).powi(2) + (end.y - start.y).powi(2)).sqrt();
6     let extrusion_volume = width * thickness * length;
7     let filament_area =
8         (std::f32::consts::PI * settings.filament.diameter.powi(2)) / 4.0;
9     let extrude = extrusion_volume / filament_area;
10
11    writeln!(writer, "G1 X{:.5} Y{:.5} E{:.5}", end.x, end.y, extrude)?;
12 }
```

6.30.6 Faserverstärkte Bewegungen mit Schnitt

Dieser Abschnitt beschreibt den Code, der die Bewegung und Extrusion der Faser steuert und den Schnittvorgang integriert. Die Funktion wird durch den Befehl `Command::MoveAndExtrudeFiberAndCut` ausgelöst und unterteilt den Extrusionsprozess in zwei Phasen: vor und nach dem Schnitt. Währenddessen wird ein Schnittbefehl M751 (siehe Listing 140) ausgeführt, um die Faser zu trennen.

Die 2D-Koordinaten `start` und `end` werden zunächst in Vektoren umgewandelt (vgl. Listing 134):

Listing 134: Umwandlung der Eingabekoordinaten in Vektoren

```
1 let (start, end) = (vec2(start.x, start.y), vec2(end.x, end.y));
```

Der Richtungsvektor ergibt sich aus der Differenz von Start- und Endpunkt (siehe Listing 135). Anschließend wird die Länge berechnet und der Vektor normalisiert:

Listing 135: Berechnung des Richtungsvektors und der Länge

```
1 let direction = start - end;
2 let length = direction.length();
3 let direction = direction.normalize();
```

Die Länge `length` entspricht der euklidischen Distanz zwischen `start` und `end`, was sich wie in Gleichung 38 ausdrücken lässt:

$$length = \sqrt{(end_x - start_x)^2 + (end_y - start_y)^2} \quad (38)$$

Die Position des Schnitts `cut_pos` wird relativ zum Endpunkt berechnet (vgl. Listing 136):

Listing 136: Bestimmung der Schnittposition

```
1 let cut_pos = end + direction * cut_pos;
```

Der Schnittpunkt verschiebt sich dabei entlang der Richtung des Bewegungsvektors.

Um den Prozess sauber zu unterteilen, wird die Gesamtbewegung in zwei Abschnitte zerlegt: vor und nach dem Schnitt (Listing 137):

Listing 137: Segmentierung der Bewegungslänge

```
1 let lenght_before_cut = (cut_pos - start).length();
2 let length_after_cut = length - lenght_before_cut;
```

Die Extrusionsmenge wird als Funktion der Strecke berechnet (siehe Listing 138). Dabei wird das Volumen des extrudierten Bereichs auf die Querschnittsfläche des Filaments umgerechnet, was in Gleichung 39 dargestellt ist.

$$extrusion_volume = \left(((width - thickness) \cdot thickness) + \pi \cdot \left(\frac{thickness}{2}\right)^2 \right) \cdot length \quad (39)$$

Listing 138: Berechnung des Extrusionsvolumens

```
1 let extrude_fn = |length: f32| {
2     let extrusion_volume = (((width - thickness) * thickness)
3         + (std::f32::consts::PI * (thickness / 2.0) * (thickness / 2.0)))
4         * length;
5     let filament_area = (std::f32::consts::PI
6         * settings.filament.diameter
7         * settings.filament.diameter) / 4.0;
8     extrusion_volume / filament_area
9 };
```

Zunächst wird die Bewegung und Extrusion bis zur Schnittposition ausgeführt (Listing 139):

Listing 139: Erste Extrusion bis zum Schnittpunkt

```
1 let extrude_before_cut = extrude_fn(lenght_before_cut);
2 writeln!(
3     writer,
4     "G1 X{:.5} Y{:.5} E{:.5} D{:.5}",
5     cut_pos.x, cut_pos.y, extrude_before_cut, extrude_before_cut
6 )?;
```

Anschließend wird der G-Code-Befehl zum Schnitt ausgeführt (Listing 140):

Listing 140: Schnitt der Faser

```
1 writeln!(writer, "M751; cut fiber")?;
```

Danach erfolgt die zweite Extrusion bis zum Endpunkt (Listing 141):

Listing 141: Zweite Extrusion nach dem Schnitt

```
1 let extrude_after_cut = extrude_fn(length_after_cut);
2 writeln!(
3     writer,
4     "G1 X{:.5} Y{:.5} E{:.5} D{:.5}",
5     end.x, end.y, extrude_after_cut, extrude_after_cut
6 )?;
```

7 Prototypenfertigung und Validierung

7.1 Einleitung zur Prototypenfertigung

Die Fertigung der Prototypen erfolgt primär mittels einer selbstkonstruierten und gebauten CNC-Fräse. Diese ermöglicht es, unabhängig von externen Fertigungsdienstleistern sowie den schuleigenen Werkstätten zeitnah und flexibel Bauteile herzustellen. Die CNC-Fräse erreicht eine Genauigkeit, die es erlaubt, Bauteile gemäß der Norm ISO 2768-fK zu fertigen.

Drehteile sowie Bauteile, die mittels Erodierverfahren bearbeitet werden müssen, werden in den Werkstätten der HTBLA Kaindorf gefertigt. Die Grobvalidierung der Maße findet im privaten Umfeld statt. Eine detaillierte und genaue Überprüfung wichtiger Maße erfolgt jedoch in den schulischen Werkstätten, da dort die notwendigen Messmittel zur Verfügung stehen.

Zur Realisierung der Fertigung im privaten Umfeld wird zusätzliches Werkzeug angeschafft. Diese Investition wird bewusst im Tausch für erhöhte Flexibilität und die Möglichkeit akzeptiert, auch außerhalb der Schulzeiten, insbesondere an Wochenenden, arbeiten zu können.

Die notwendigen G-Code-Dateien für die CNC-Bearbeitung werden mit dem CAM-Modul der Software Fusion360 generiert. Die Fertigung innerhalb der Schule beschränkt sich überwiegend auf Prozesse, die ohne computerunterstützte Verfahren durchgeführt werden.

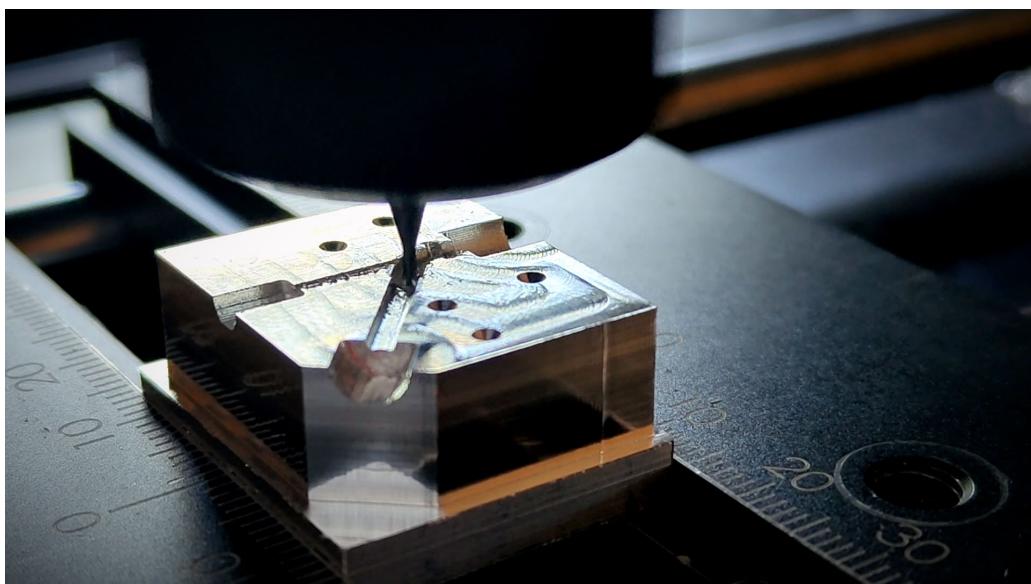
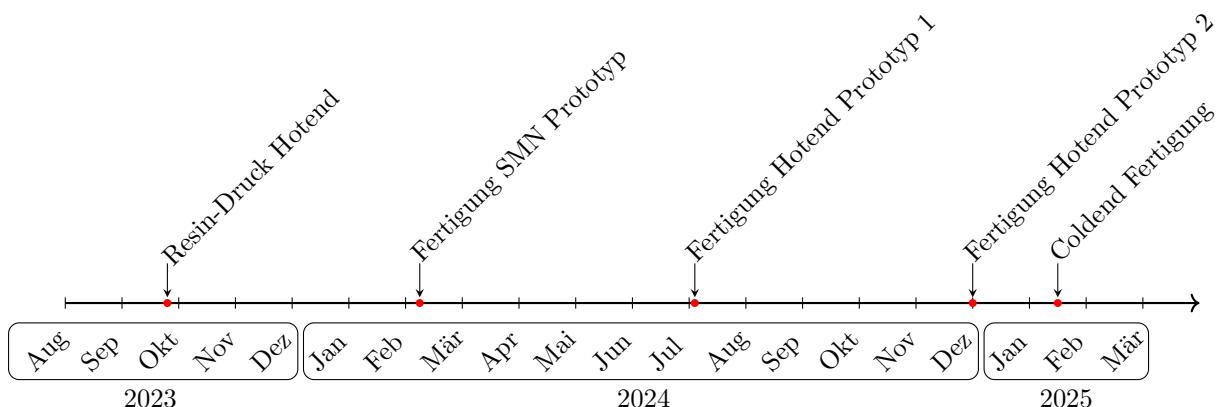


Abb. 127: Hotendfertigung des ersten Protoyps

7.1.1 Zeitlicher Ablauf der Prototypenfertigung

Hier ist der zeitliche Verlauf der Prototypenfertigung dargestellt.



7.2 Einleitung zur Validierung

Die Validierung besteht aus unterschiedlichen Verfahren, die sicherstellen sollen, dass die Konstruktionen so funktioniert wie vorgesehen. Es erfolgen Druckversuche sowie Schneidtests am Schneidmechanismus (SMN). Ergänzend dazu findet eine Validierung mittels einer selbstgebauten Kraftmessmaschine statt, die aus einem ausgeschlachteten 3D-Drucker, geeigneter Elektronik und entsprechender Software aufgebaut ist. Diese Maschine ermöglicht es, die auftretenden Schneidkräfte präzise zu erfassen, zu analysieren und somit den Schneidmechanismus gezielt zu optimieren. Ziel der Validierung ist es, Konstruktionsfehler frühzeitig zu erkennen und in nachfolgenden Versionen zu verbessern, um ein prozessstables Design und eine zuverlässige Funktionalität des gesamten Systems sicherzustellen.

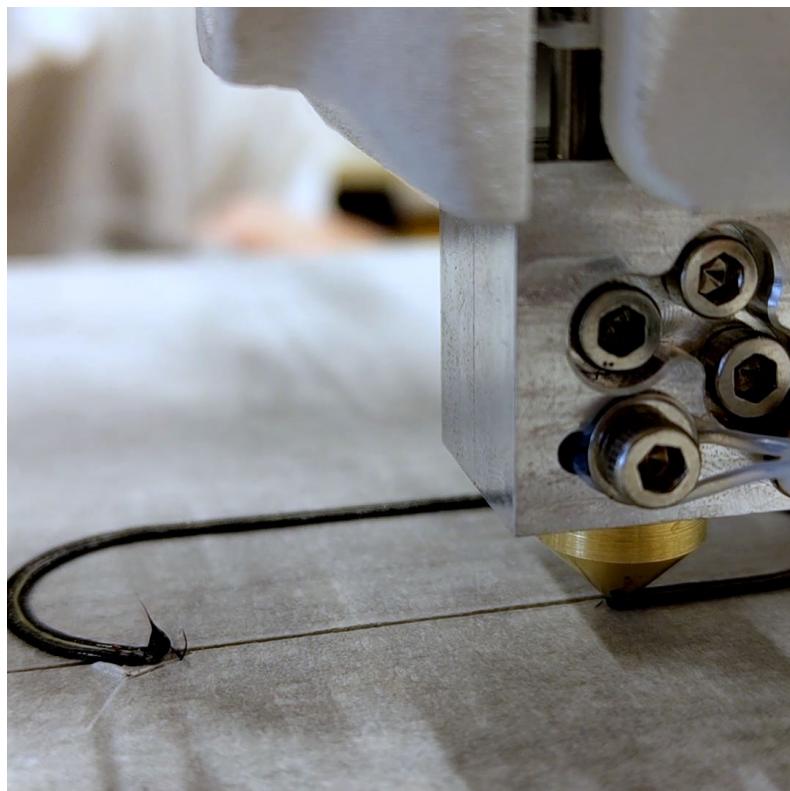
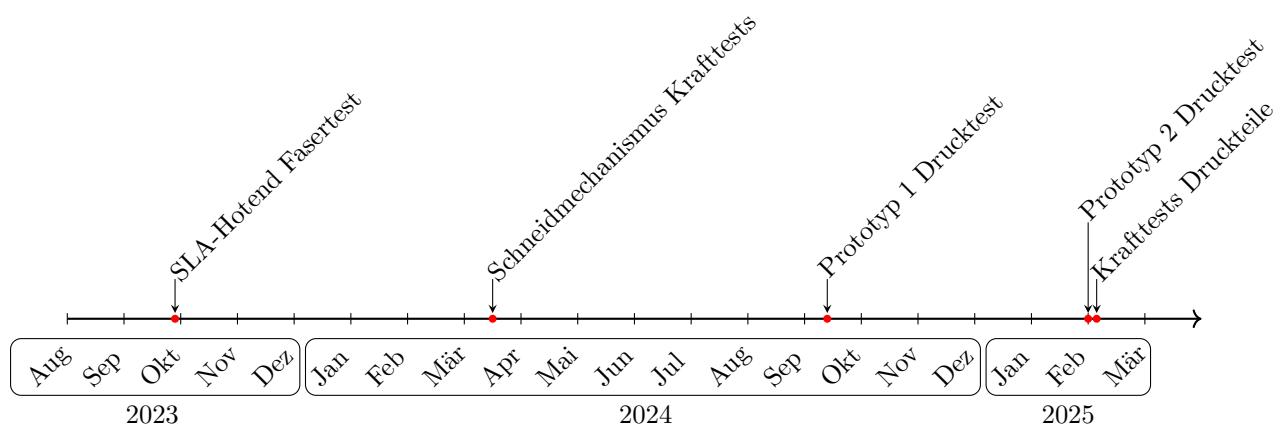


Abb. 128: Druckvalidierung des zweiten Protoyps

7.2.1 Zeitlicher Ablauf der Validierung

Hier ist der zeitliche Verlauf der Validierung dargestellt.



7.3 SLA-Druck Hotendprototyp

Zur Validierung des Designs erfolgt ein stellvertretender Versuch mit einem mittels SLA-Druck (*Stereolithografie*) hergestellten Prototyp des Split-Hotends (Version 4). Dabei wird eine flüssige Teigmischung, welche das reale Filament simuliert, mittels einer Spritze kontrolliert in den Filamentkanal injiziert. Ziel dieses Versuchs ist es, das Fließverhalten innerhalb des Filamentkanals zu untersuchen und sicherzustellen, dass die Mischung nicht unbeabsichtigt in den angrenzenden Faserkanal gelangt und diesen dadurch verstopft. Durch diesen Versuch werden frühzeitig mögliche Schwachstellen im Design identifiziert und Optimierungen ermöglicht.

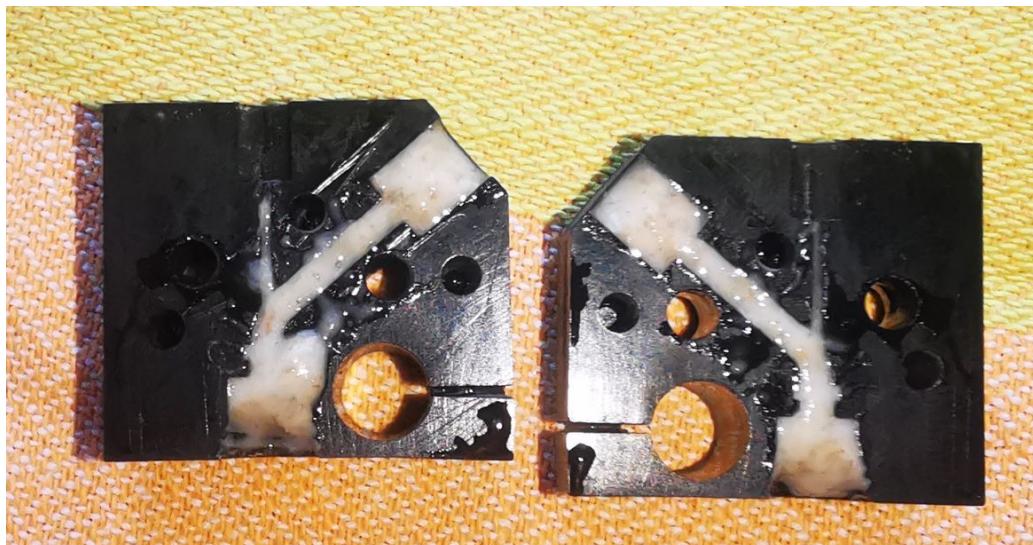


Abb. 129: Split-Hotend

Zur Sicherstellung der zuverlässigen Mitführung einer Faser im Faserkanal erfolgt ein zusätzlicher, einfacher Versuch. Dabei wird eine textile Faser exemplarisch in den dafür vorgesehenen Kanal des Hotend-Prototyps eingelegt. Anschließend erfolgt erneut die Injektion der flüssigen Teigmischung („Filament“), um zu testen, ob die eingelegte Faser durch den Materialfluss erfasst und kontinuierlich mitgezogen wird. Die Fasermitführung verläuft dabei erfolgreich, sodass keine Anpassungen am Design erforderlich sind. Ziel dieser Validierung ist es, die grundlegende Funktionalität des Designs hinsichtlich der automatischen Mitnahme der Faser durch den Filamentfluss zu bestätigen und gegebenenfalls Optimierungsbedarf zu identifizieren.



Abb. 130: Teigmischung



Abb. 131: Erfolgreiche Faserextrusion

7.4 Schneidmechanismus

Der Schneidmechanismus stellt ein zentrales Element zur zuverlässigen und präzisen Bearbeitung der Fasern dar. Die Fertigung sowie die Validierung dieses Mechanismus sind maßgeblich für seine spätere Funktionalität.

7.4.1 Fertigung

Die Fertigung des Schneidmechanismus umfasst die Prozesse Drehen und Erodieren, die nachfolgend genauer beschrieben werden.

Drehen

Die Buchse des Schneidmechanismus wird aus Stahl gefertigt. Diese Bearbeitung erfolgt auf einer Drehmaschine in der schulischen Werkstatt der HTBLA Kaindorf. Anschließend erfolgt eine Feinbearbeitung mittels einer Reibahle, um eine präzise Bohrung (\varnothing 4 mm, Toleranz H7) herzustellen. Als Schneidstift kommt ein 4 mm M6 Passstift zum Einsatz.

Erodieren

Das Erodieren der Buchse erfolgt mittels Senkerodierverfahren. Dabei nutzt man einen elektrischen Entladeprozess zwischen der leitfähigen Buchse und einer Graphitelektrode (in diesem Fall eine Graphitmine eines Minenbleistifts), um Material abzutragen. Aufgrund erster Erfahrungen mit diesem Verfahren dauert der Prozess mehrere Stunden und erfordert zwischendurch den Austausch sowie das erneute Antasten der Graphitmine. Durch das erneute Antasten verschiebt sich das Loch geringfügig, was zu einer Vergrößerung führt. Nach Abschluss der Bearbeitung zeigt die verwendete Graphitmine erhebliche Abnutzungsscheinungen.



Abb. 132: Graphitmine nach der Fertigung

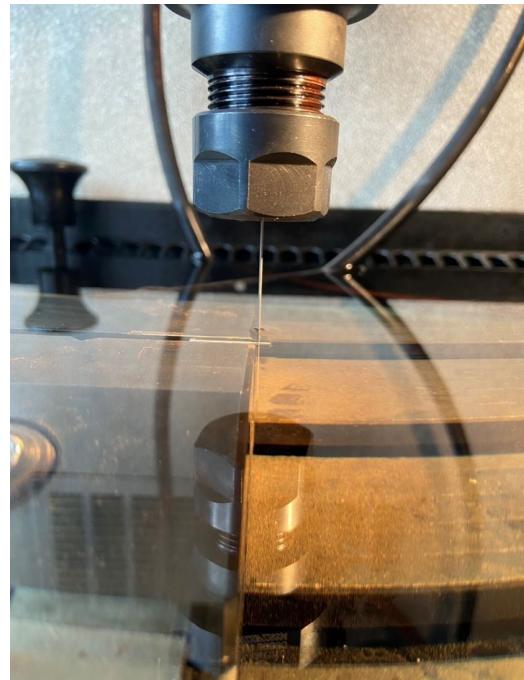


Abb. 133: Senkerodierprozess

7.4.2 Validierung

Die Validierung des Schneidmechanismus umfasst sowohl optische Messungen als auch Krafttests zur Untersuchung seiner Funktionalität.

Optische Vermessung

Die Vermessung der gefertigten Löcher erfolgt mithilfe einer optischen Messanlage in der Schule. Dabei ergeben sich unterschiedliche Ergebnisse aufgrund der bereits erwähnten Schwierigkeiten beim Erodieren: Das untere Loch weist einen Durchmesser von 0,588 mm auf, während das obere Loch, bedingt durch die Verschiebung beim erneuten Antasten, 0,997 mm misst. Diese Abweichungen sind vermutlich auf den Funkenspalt sowie unzureichende Parametereinstellungen zurückzuführen.

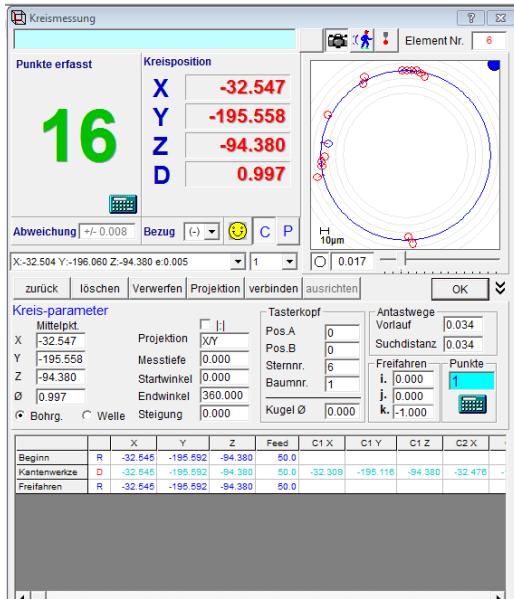


Abb. 134: Oberes Loch

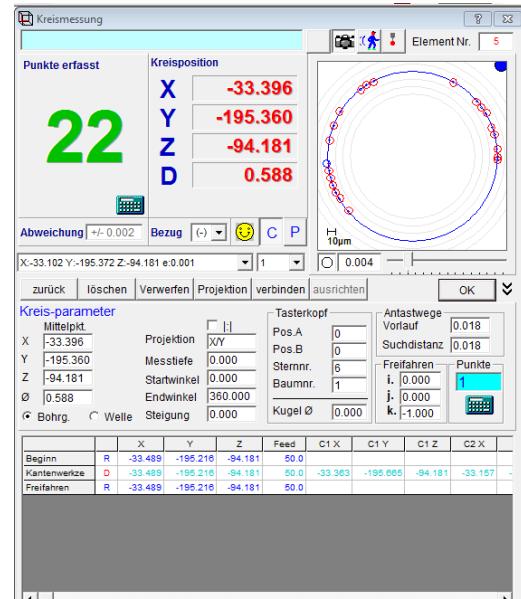


Abb. 135: Unteres Loch

Krafttestmaschine

Zur weiteren Validierung erfolgt ein Krafttest mithilfe einer eigens konstruierten Krafttestmaschine, basierend auf einem umgebauten 3D-Drucker. Aufgrund der mangelnden Steifigkeit des Systems wird softwareseitig die Elastizität herausgerechnet. Mithilfe dieser Maschine wird die notwendige Schneidkraft ermittelt, die erforderlich ist, um eine Kupferlitze aus einer flexiblen Kupferleitung zu schneiden. Die aufgenommenen Messwerte werden anschließend mittels eines Python-Skripts visualisiert. Es erfolgen jeweils zehn Messungen pro Bohrung, wobei sich zeigt, dass die kleinere Bohrung erwartungsgemäß geringere Schneidkräfte benötigt.

In Abbildung 137 ist der Aufbau des Krafttests dargestellt. Dabei wird der Kupferdraht durch beide Löcher des Schneidmechanismus geführt. In den Messungen gemäß Abbildung 138 erfolgt die Zuführung des Kupferdrahtes jedoch jeweils nur durch ein einzelnes Loch, da beim Schneiden der Faser im Hotend ausschließlich auf einer Seite geschnitten wird.

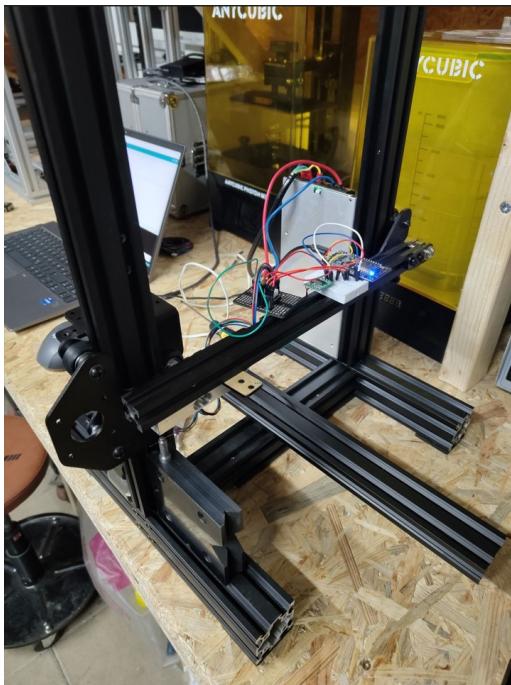


Abb. 136: Krafttestmaschine



Abb. 137: Schneidmechanismus Krafttest

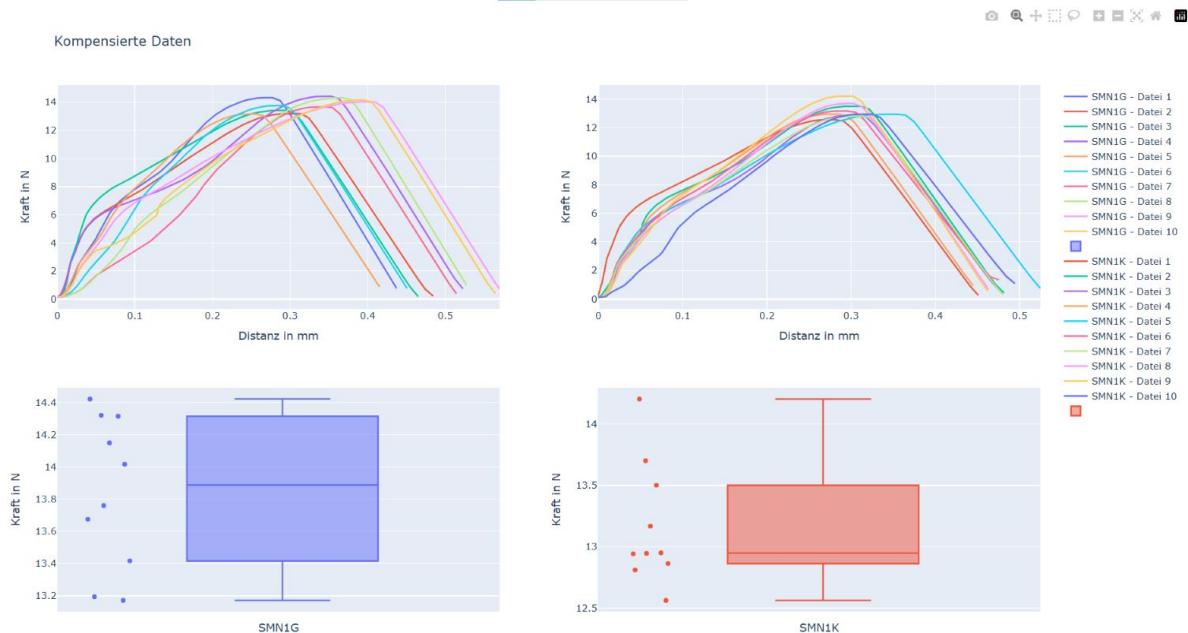


Abb. 138: Schneidmechanismus Krafttestdaten

Die Elastizität des gesamten Versuchsaufbaus wird durch eine Kompensationsmethode eliminiert, um möglichst präzise und reproduzierbare Messergebnisse zu gewährleisten. Hierzu erfolgt eine Referenzmessung mit einem Stahlblock, anhand derer die Nachgiebigkeit des Systems durch eine systematische Erfassung von Datenpunkten quantifiziert wird.

7.5 Prototyp 1

7.5.1 Fertigung

Der erste gefräste Prototyp basiert auf der Version 6 des Hotends. Aufgrund der noch nicht gelieferten Komponenten, wie dem vorgesehenen Heatbreak und Coldend, wird ein Coldend eines anderen Druckkopfs verwendet. Gefräst wird ausschließlich das Hotend der Version 6.

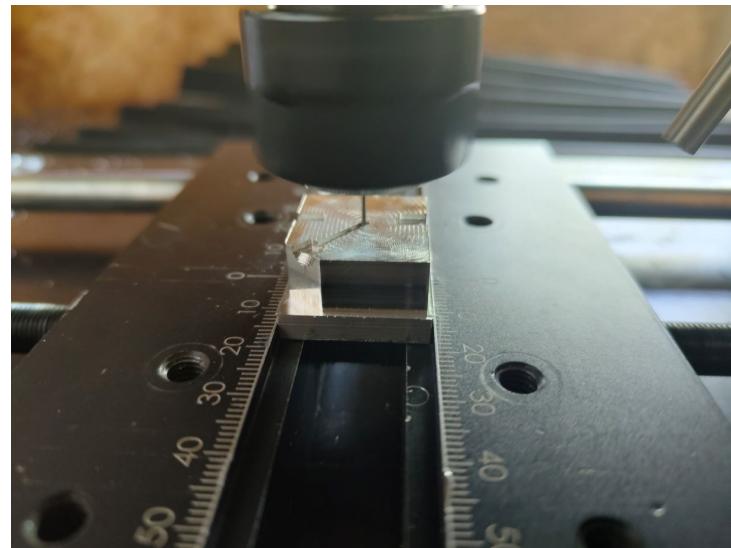


Abb. 139: Fertigung des Hotends Version 6

7.5.2 Validierung

Testaufbau

Für die Tests wird eine Standard-3D-Druckdüse mit 0,6 mm Durchmesser verwendet. Der Test erfolgt mit PLA-Filament und einem alten 3D-Drucker. Als Faser dient ein einfacher Faden, der in den Faserkanal eingelegt wird. Es erfolgt hauptsächlich eine gerade Extrusion, um zu untersuchen, ob die Faser zuverlässig mitgeführt wird. Da das geplante Insert zu diesem Zeitpunkt noch nicht gefertigt wurde, wird der Test ohne Insert durchgeführt.

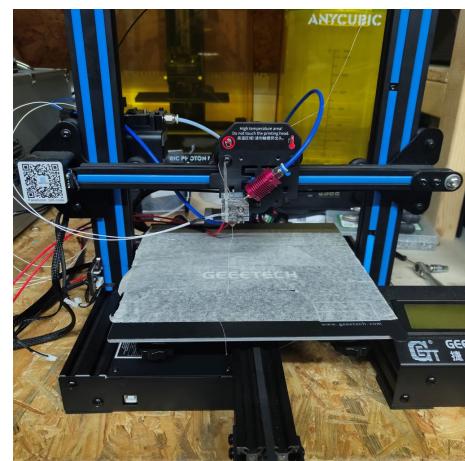
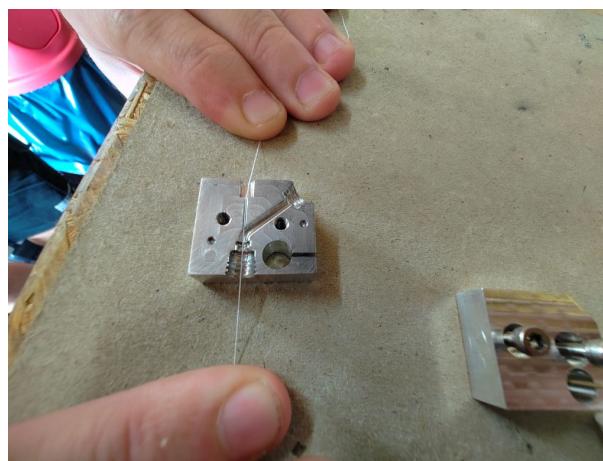


Abb. 140: Testaufbau des ersten gefrästen Hotend-Prototyps

Ergebnisse

Die Tests zeigen, dass die Faser nicht zuverlässig mitgeführt wird. Dies ist darauf zurückzuführen, dass das Filament mit einem Durchmesser von 1,75 mm im Inneren der Düse nicht gleichmäßig strömt. Die Düse verjüngt sich erst kurz vor der Austrittsstellung auf 0,6 mm, wodurch das Filament hinter der Düse langsamer fließt als erforderlich wäre, um die Faser mitzureißen.

Zusätzlich zeigen die Versuche, dass das Hotend anfangs eine gute Abdichtung aufweist. Allerdings tritt mit zunehmender Anzahl an Reinigungszyklen immer mehr Filament aus, was auf eine allmähliche Undichtigkeit durch mechanische Beanspruchung hindeutet.

Ein abschließender Test beinhaltet den Druck eines quadratischen Musters mit integrierter Faser. Dabei treten mehrere Probleme auf:

- Das Filament kühlt nicht schnell genug ab.
- Die Faser wird nicht zuverlässig mitgezogen.
- Die Ecken sind zu scharfkantig, was dazu führt, dass die Faser nicht der gewünschten Bahn folgt, sondern quer durch den Druck gezogen wird.

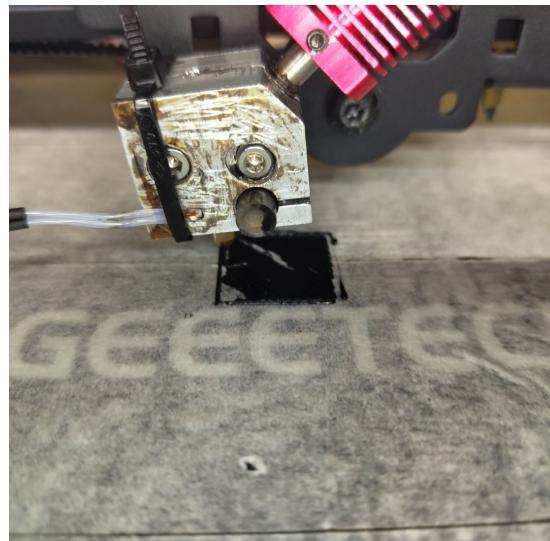


Abb. 141: Druckversuch eines quadratischen Musters mit Faserintegration



Abb. 142: Ergebnisse des ersten Hotend-Tests

Erkenntnisse

Trotz der aufgetretenen Probleme bestätigt der Test grundsätzlich die Funktionsfähigkeit des Hotends. Wichtige Erkenntnisse aus diesem Versuch sind:

- Die Düse muss im gesamten letzten Abschnitt einen gleichmäßigen Durchmesser von 0,6 mm aufweisen, um ein gleichmäßiges Fließverhalten des Filaments sicherzustellen.
- Die Faser sollte so kurz wie möglich im Bereich des Hotends vor der Verjüngung geführt werden, um eine bessere Mitnahme zu gewährleisten. Dies kann erreicht werden, indem das Insert bis kurz vor den 0,6-mm-Abschnitt der Düse reicht.

7.6 Prototyp 2

7.6.1 Fertigung

Der zweite gefräste Prototyp basiert auf der Version 7.0.0 des Hotends. Zusätzlich wird erstmals auch das zugehörige Coldend gefertigt. Die Düse wird in der schuleigenen Werkstatt auf einer Drehmaschine hergestellt. Der Schneidmechanismus ist zu diesem Zeitpunkt noch nicht gefertigt. Das G22-Kanülen-Insert muss vor der Montage mit einem Schleifblock auf die richtige Länge gebracht werden, da es beim Schneiden regelmäßig deformiert wird. Das Gehäuse des Hotends wird aus ABS-GF auf einem Bambulab-Drucker in der Schule gedruckt.



Abb. 143: Gefertigtes Hotend, Version 7.0.0

Während der CNC-Bearbeitung treten mehrere Probleme auf. Durch einen Wackelkontakt am Motorstecker der Z-Achse kommt es zu instabilen Bewegungen, was mehrfach zu Werkzeugbrüchen führt. Besonders anspruchsvoll gestaltet sich das Fräsen der Kühlrippen des Coldends. Obwohl die Hochgeschwindigkeitsspindel hohe Drehzahlen erreicht, fehlt es ihr an ausreichend Drehmoment für diese Bearbeitung. Zusätzlich wird irrtümlich eine Scheibenfräser-Schneidplatte mit zu vielen Zähnen verwendet, was zu deren Bruch führt. Da keine Ersatzscheibe zur Verfügung steht, muss auf die Kühlrippen verzichtet werden.

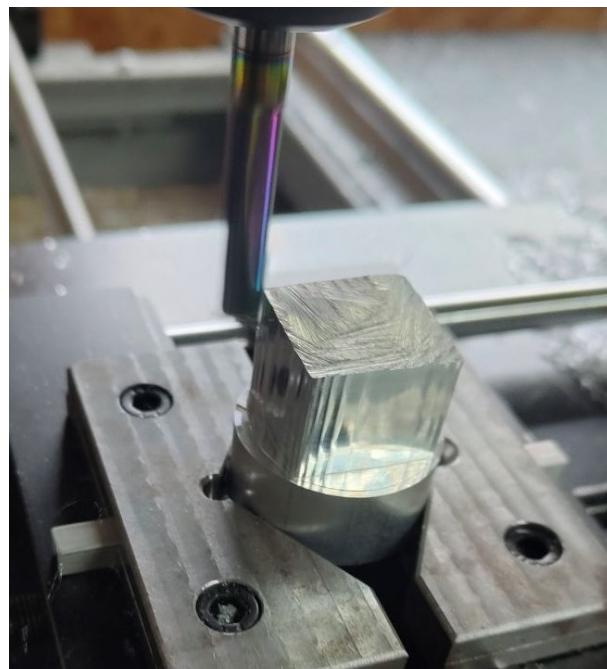


Abb. 144: Coldendfertigung, Version 7.0.0

7.6.2 Validierung

Der zweite Prototyp wird mehrfach getestet. Die Vorbereitungszeit beträgt dabei etwa 1,5 bis 2 Stunden und umfasst den Zusammenbau des Druckkopfes, das Einlegen der Faser, den Anschluss der provisorischen Elektronik sowie die Vorbereitung der Druckdatei.

Testaufbau

Beim ersten Test fehlt das Hochtemperatursilikon zur Fixierung des Inserts. Stattdessen wird Superkleber verwendet, welcher sich jedoch bereits bei 80°C auflöst. Dadurch rutscht das Insert in das Hotend und verstopft die Düse vollständig. Bei den weiteren Tests wird das vorgesehenen Hochtemperatursilikon verwendet, wodurch das Insert zuverlässig hält.

Während eines der Tests wird die Faser bei aufgeheiztem Hotend zurückgezogen. Dies führt dazu, dass die bereits mit Kunststoff benetzte Faser zurück in das Insert gezogen wird und dieses verstopft. Dieser Effekt zeigt, dass besondere Vorsicht geboten ist, um eine ungewollte Blockade des Inserts zu vermeiden.

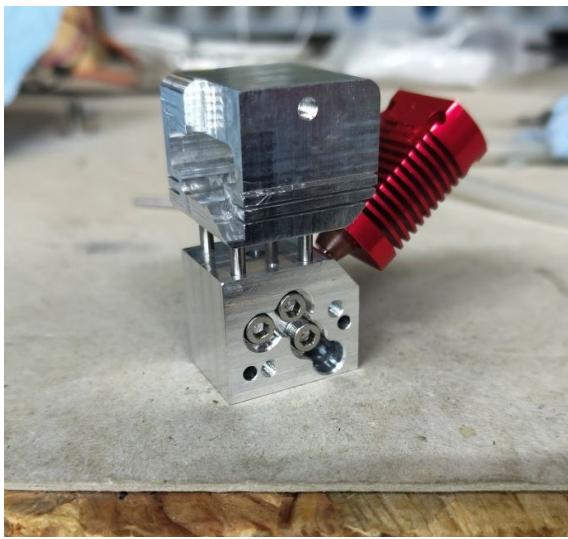


Abb. 145: Zusammengebautes Cold- und Hotend



Abb. 146: Montierter Druckkopf

Es werden insgesamt zwei erfolgreiche Testdrucke mit Faser durchgeführt. Zusätzlich wird dasselbe Modell ohne Faser gefertigt, um es auf der Krafttestmaschine zu untersuchen.

Kraftmessung

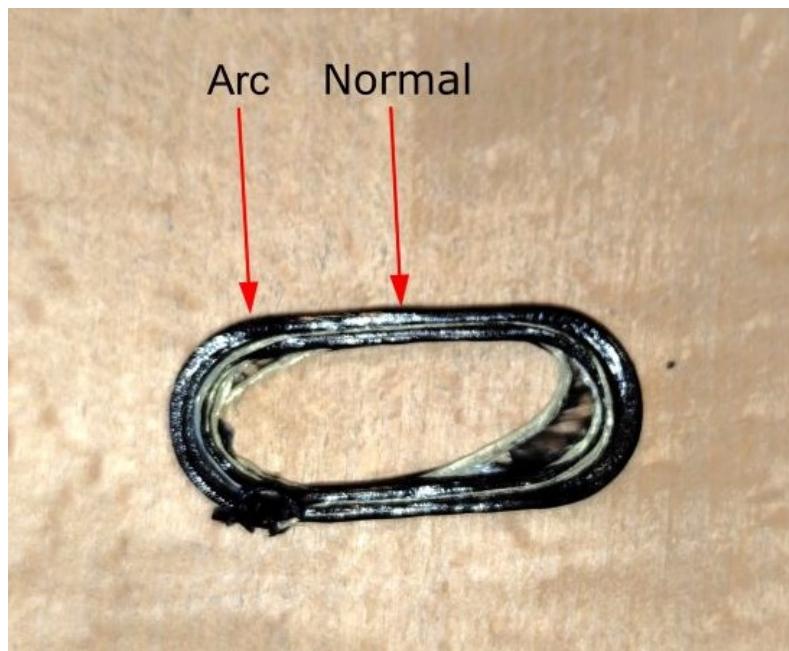


Abb. 147: Fasertestobjekt

In Abbildung 147 ist die Position der durchgeföhrten Messungen am faserverstärkten Testobjekt ersichtlich. Das Testobjekt ohne Faser weist eine identische Geometrie auf und wurde unter denselben Bedingungen getestet.

Die gemessenen Kraft-Weg-Daten sind in Abbildung 148 dargestellt. Hierbei repräsentiert „Datei 1“ das mit Faser verstärkte Bauteil, während „Datei 2“ die Messergebnisse des unverstärkten Bauteils zeigt. Da die Krafttestmaschine mit einer 5-kg-Kraftzelle ausgestattet ist, erfolgt eine automatische Beendigung des Tests, sobald diese maximale Belastungsgrenze erreicht wird.

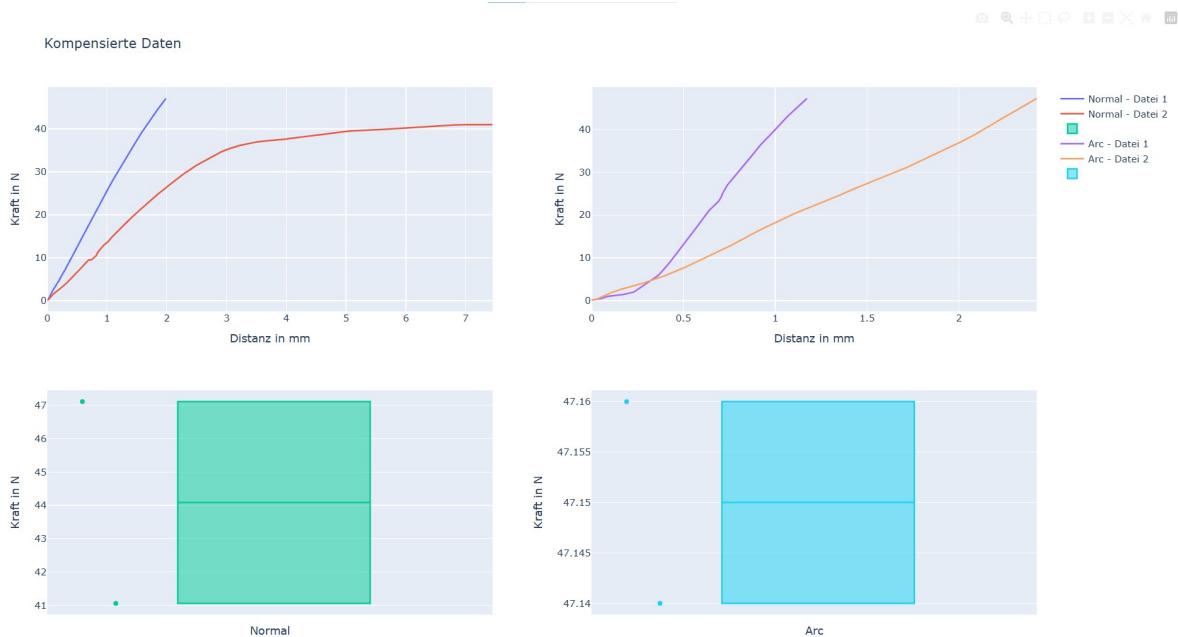


Abb. 148: Krafttestdaten

Die Kraftmessungen zeigen eine deutliche Verbesserung der mechanischen Eigenschaften durch die Faserintegration. Das faserverstärkte Bauteil nimmt bereits bei geringerer Verformung hohe Kräfte auf, während das unverstärkte Bauteil erst bei größerer Dehnung vergleichbare Belastungen erreicht. Die

steilere Steigung der Kraft-Weg-Kurve bei der faserverstärkten Variante weist auf ein höheres Biegemodul und eine erhöhte Steifigkeit hin, während die flachere Steigung des unverstärkten Bauteils eine größere elastische Verformung unter gleicher Belastung zeigt.

Das darunter liegende Boxplot liefert keine relevanten Erkenntnisse zur mechanischen Charakterisierung der Testobjekte, sondern ist lediglich als Bestandteil der automatisierten Visualisierung durch das verwendete Python-Skript dargestellt.

8 Lastenheft

Ziel und Zweck dieses Dokuments

Das vorliegende Lastenheft enthält die an das auszuarbeitende Projekt gestellten funktionalen sowie nicht-funktionalen Anforderungen. Die Auftraggeber werden in den Anforderungen die grundlegenden Entwicklungsvorgaben definieren, die von den Auftragnehmern im Pflichtenheft im Detail ausgearbeitet werden. Bei diesem Projekt werden die Auftraggeber auch die Auftragnehmer sein.

8.1 Funktionale Anforderungen

8.1.1 Anforderungen Druckqualität

Nr. / ID	000	Nichttechnischer Titel	Festgelegte Ziele für die Druckqualität
Priorität	Sehr hoch	Quellen	Marktforschung

Das vorrangige Ziel des Projekts ist die Erreichung einer hochwertigen Druckqualität. Um sicherzustellen, dass der Druckkopf in der Praxis anwendbar und sinnvoll ist, muss die Festigkeit des Bauteils mit Fasern größer sein als die ohne Fasern. Darüber hinaus wird angestrebt, dass der mit Fasern verstärkte Druck optisch keine signifikanten Unterschiede zu einem Bauteil ohne Fasern aufweist. Die Anforderungen an die Druckqualität werden auf folgende Punkte aufgeteilt:

- Verbesserung der mechanischen Festigkeit durch Faserverlegung
- Optische Ästhetik von Faser-verstärkten Drucken

Verbesserung der mechanischen Festigkeit durch Faserverlegung

Nr. / ID	001	Nichttechnischer Titel	Belastbare Bauteile durch gezielte Stärkung mit Fasern
Priorität	Hoch	Quellen	Erfahrungen aus der Branche im Bereich Faserverstärkung im 3D-Druck

Beschreibung	Sicherstellung, dass mit Fasern verstärkte Bauteile eine höhere Festigkeit und Stabilität aufweisen als vergleichbare Bauteile ohne Faserverlegung.
Wechselwirkungen	Direkte Auswirkung auf die mechanischen Eigenschaften der gedruckten Bauteile, insbesondere auf deren Festigkeit und Stabilität.
Risiken	Risiko von fehlerhafter Faserverlegung, was zu ungleichmäßigen Festigkeitseigenschaften im gedruckten Bauteil führen könnte.
Vergleich mit bestehenden Lösungen	Vergleich mit anderen Faserverlegungstechnologien und deren Auswirkungen auf die Druckqualität und mechanischen Eigenschaften von Bauteilen.
Schätzung des Aufwands	14 Wochen

Optische Ästhetik von Faser-verstärkten Drucken

Nr. / ID	002	Nichttechnischer Titel	Schöne Bauteile bei Faserverstärkung
Priorität	Mittel	Quellen	Design- und Ästhetikstandards im 3D-Druck
Beschreibung		Gewährleistung, dass ein mit Fasern verstärkter Druck keine signifikanten optischen Unterschiede zu einem Bauteil ohne Fasern aufweist, um eine ansprechende und ästhetisch hochwertige Oberfläche zu erreichen.	
Wechselwirkungen		Direkte Auswirkung auf das äußere Erscheinungsbild des gedruckten Bauteils.	
Risiken		Risiko von sichtbaren Faser-Unregelmäßigkeiten oder Farbabweichungen, die die optische Ästhetik beeinträchtigen könnten.	
Vergleich mit bestehenden Lösungen		Vergleich mit anderen Faser-verstärkten Drucken und deren visuellen Eigenschaften, um best practices für die optische Ästhetik zu identifizieren.	
Schätzung des Aufwands		4 Wochen	

8.1.2 Hotend

Nr. / ID	010	Nichttechnischer Titel	Unterbau Druckkopf
Priorität	Sehr hoch	Quellen	Marktforschung

Das Hotend ist der Unterbau des Druckkopfes. Sein Name leitet sich davon ab, dass es von entscheidender Bedeutung ist, dass es das zugeführte Filament schmilzt. Im Hotend werden Filament und Faser zusammengeführt, was einer der wichtigsten Schritte im ganzen Projekt ist. Weil dieser Schritt aber auch einer der schwierigsten ist, ist es nötig das Hotend wartungsfreundlich zu gestalten. Die Anforderungen an das Hotend werden auf folgende Punkte aufgeteilt:

- Schmelzverfahren für Filament
- Fusionsprozess von Filament und Fasern
- Wartungsfreundliche Gestaltung des Hotends

Schmelzverfahren für Filament

Nr. / ID	011	Nichttechnischer Titel	3D-Druckmaterial-Schmelzprozess im Hotend
Priorität	Sehr hoch	Quellen	Literatur über Schmelzprozesse in der additiven Fertigungstechnologie

Beschreibung	Gewährleistung eines effizienten und präzisen Schmelzprozesses für das zugeführte Filament im Hotend
Wechselwirkungen	Direkte Auswirkung auf die Materialhomogenität und -viskosität während des Schmelzvorgangs
Risiken	Risiko von Schmelzproblemen, wie unzureichende Schmelztemperatur oder ungleichmäßiges Schmelzen, was zu Druckfehlern und Qualitätsminderung führen könnte
Vergleich mit bestehenden Lösungen	Vergleich mit anderen bewährten Filament-Schmelztechniken und Hotend-Designs, die eine effiziente Filament-Extrusion ermöglichen
Schätzung des Aufwands	8 Wochen

Fusionsprozess von Filament und Fasern

Nr. / ID	012	Nichttechnischer Titel	Verschmelzung von Druckmaterialien und Fasern
Priorität	Sehr hoch	Quellen	Kombination Druckmaterialien und Fasern
Beschreibung		Hotend, welches die Fusion von Filament und Faser bereits innerhalb des Hotends ermöglicht.	
Wechselwirkungen		Die präzise Zusammenführung der Fasern mit dem Filament im Zentrum des Hotends bestimmt maßgeblich die endgültige Stabilität des gedruckten Bauteils. Eine sorgfältige Abstimmung mit anderen Druckerkomponenten ist dabei von entscheidender Bedeutung.	
Risiken		Die Herausforderung besteht darin, eine effiziente und zuverlässige Methode zu entwickeln, um Filament und Faser im Hotend zu vereinen. Zusätzlich besteht ein Verstopfungsrisiko.	
Vergleich mit bestehenden Lösungen		Im Gegensatz zum Markforged Hotend, das zunächst das Filament druckt und anschließend die Fasern über eine separate Düse in den Druck einbringt, wird das Filament und die Fasern direkt im Hotend kombiniert.	
Schätzung des Aufwands		Die Entwicklung des Hotends erfordert eine sorgfältige Konzeption und Implementierung und wird inklusive intensiver Testungen voraussichtlich etwa 32 Wochen Entwicklungszeit in Anspruch nehmen.	

Wartungsfreundliche Gestaltung des Hotends

Nr. / ID	013	Nichttechnischer Titel	Benutzerfreundlichkeit bei der Hotend-Wartung
Priorität	Sehr hoch	Quellen	Expertenwissen über Wartungsfreundlichkeit von Hotends, Erfahrungen aus der Industrie
Beschreibung		Gewährleistung einer Konstruktion des Hotends, die eine einfache Wartung und Reinigung ermöglicht.	
Wechselwirkungen		Direkte Auswirkung auf die Zugänglichkeit und Austauschbarkeit von Verschleißteilen sowie die Gesamtkomplexität des Wartungsprozesses.	
Risiken		Risiko von schwer zugänglichen Komponenten und komplizierten Demontageprozessen, was zu längeren Wartungszeiten und höheren Instandhaltungskosten führen könnte.	
Vergleich mit bestehenden Lösungen		Vergleich mit anderen Hotend-Designs und -Technologien, die eine wartungsfreundliche Gestaltung implementiert haben, um bewährte Praktiken zu identifizieren und zu übernehmen.	
Schätzung des Aufwands		10 Wochen	

8.1.3 Coldend

Nr. / ID	020	Nichttechnischer Titel	Überbau Druckkopf
Priorität	Hoch	Quellen	Marktforschung

Das Coldend ist der Überbau des Druckkopfes. Sein Name leitet sich davon ab, dass es von entscheidender Bedeutung ist, dass es nicht überhitzt. Im Coldend werden Filament und Faser zum Hotend transportiert, die Fasern geschnitten und die Fasern werden geschoben. Die Anforderungen an das Coldend werden auf folgende Punkte aufgeteilt:

- Filament- und Faserführung
- Wärmedämmung
- Schneidmechanismus
- Faservorschub

Filament- und Faserführung

Nr. / ID	021	Nichttechnischer Titel	Überbau Design für Filament und Faser
Priorität	Hoch	Quellen	Marktforschung
Beschreibung	Design des Coldends zur Filament- und Faserführung zum Hotend.		
Wechselwirkungen	Enge Abstimmung mit der Wärmedämmung und Führung im Hotend.		
Risiken	Mangelhafte, unzuverlässige Führung der Fasern und des Filaments vom Coldend zum Hotend können zur fehlerhaften Zentrierung der Fasern im Filament im Hotend führen.		
Vergleich mit bestehenden Lösungen	Analyse bestehender Coldend-Designs und Vergleich mit ähnlichen Lösungen.		
Schätzung des Aufwands	20 Wochen		

Wärmedämmung

Nr. / ID	022	Nichttechnischer Titel	Wärmedämmung und Kühlung des Coldends
Priorität	Hoch	Quellen	Industriestandards für Wärmedämmung und Kühlung
Beschreibung	Gewährleistung einer effektiven Wärmedämmung und Kühlung im Coldend.		
Wechselwirkungen	Einfluss auf die Temperaturstabilität und Führung im Hotend.		
Risiken	Überhitzungsrisiko und Beeinträchtigung der Filamentführung.		
Vergleich mit bestehenden Lösungen	Vergleich mit bewährten Kühlungslösungen und Wärmedämmmaterialien.		
Schätzung des Aufwands	4 Wochen		

Schneidmechanismus

Nr. / ID	023	Nichttechnischer Titel	Schneidmechanismus für Fasern im Coldend
Priorität	Hoch	Quellen	Industriestandards für Faserbearbeitung
Beschreibung	Entwicklung eines zuverlässigen Schneidmechanismus für die Fasern im Coldend.		
Wechselwirkungen	Direkter Einfluss auf die Führung und Fusion im Hotend.		
Risiken	Risiko des unzureichenden Faserschnitts und folglich beeinträchtigter Führung.		
Vergleich mit bestehenden Lösungen	Vergleich mit bewährten Faserschneidlösungen und -technologien.		
Schätzung des Aufwands	8 Wochen		

Faservorschub

Nr. / ID	024	Nichttechnischer Titel	Faser-Extrusion im Coldend
Priorität	Hoch	Quellen	Industriestandards für Faserbearbeitung, Marktforschung
Beschreibung	Gewährleistung eines präzisen und zuverlässigen Vorschubs der Faser im Coldend.		
Wechselwirkungen	Direkte Auswirkung auf die Führung und Trennung der Fasern, im weiteren Stabilität und Festigkeit gedruckter Teile bei nicht mitgedruckter Faser.		
Risiken	Risiko unzureichenden Vorschubs oder Verstopfung der Faser und daraus resultierender Minderung der Bauteilqualität.		
Vergleich mit bestehenden Lösungen	Vergleich mit etablierten Faser-Vorschubmechanismen.		
Schätzung des Aufwands	12 Wochen		

8.1.4 Plattformunabhängige Integration

Nr. / ID	030	Nichttechnischer Titel	Aufbau auf andere 3D-Drucker
Priorität	Hoch	Quellen	Branchenstandards für 3D-Drucker, Marktanalysen

Um eine plattformunabhängige Integration zu gewährleisten, müssen bestimmte Aspekte berücksichtigt werden. Dazu zählt die Kompatibilität mit gängigen 3D-Druckern, welche z.B. verschiedene Antriebsformen berücksichtigt. Außerdem ist es wichtig, die Größe und das Gewicht des Druckkopfes so zu wählen, dass er sich mit üblichen Geschwindigkeiten bewegen kann. Eine reibungslose Integration in Standard-3D-Drucker ist ebenfalls von Bedeutung. Die Anforderungen an die plattformunabhängige Integration werden auf folgende Punkte aufgeteilt:

- Marktübliche 3D-Drucker Interoperabilität
- Designanpassungen für Druckkopfgröße und -gewicht
- Nahtlose Einbindung in Standard-3D-Drucker

Marktübliche 3D-Drucker Interoperabilität

Nr. / ID	031	Nichttechnischer Titel	Kompatibilität mit handelsüblichen 3D-Druckern
Priorität	Hoch	Quellen	Branchenstandards für 3D-Drucker, Marktanalysen

Beschreibung	Gewährleistung einer Kompatibilität mit handelsüblichen 3D-Druckern.
Wechselwirkungen	Direkte Auswirkung auf die Kompatibilität und Montage des Drucksystems, insbesondere in Bezug auf die Druckgenauigkeit und Stabilität.
Risiken	Risiko von Inkompabilität und mangelnder Leistung aufgrund unzureichender Anpassung an verschiedene Druckerplattformen.
Vergleich mit bestehenden Lösungen	Vergleich mit anderen modularen Systemen für handelsübliche 3D-Drucker.
Schätzung des Aufwands	10 Wochen

Designanpassungen für Druckkopfgröße und -gewicht

Nr. / ID	032	Nichttechnischer Titel	Größen- und Gewichtsbeschränkung für die Druckkopfbewegung
Priorität	Hoch	Quellen	Empfohlene Spezifikationen für 3D-Druckgeschwindigkeiten
Beschreibung		Beschränkung des Gewichts und der Größe des Druckkopfes um mit üblichen Bewegungsgeschwindigkeiten drucken zu können.	
Wechselwirkungen		Direkte Auswirkung auf die Präzision der Bewegungen des Druckkopfes während des Druckvorgangs und dadurch Auswirkungen auf die Bauteilqualität.	
Risiken		Risiko von Bewegungseinschränkungen und Geschwindigkeitsverlusten aufgrund eines zu schweren oder sperrigen Druckkopfes.	
Vergleich mit bestehenden Lösungen		Vergleich mit anderen leichten und kompakten Druckkopfdesigns, die für hohe Druckgeschwindigkeiten optimiert sind.	
Schätzung des Aufwands		10 Wochen	

Nahtlose Einbindung in Standard-3D-Drucker

Nr. / ID	033	Nichttechnischer Titel	Simple Montage auf handelsüblichen 3D-Druckern
Priorität	Hoch	Quellen	Erfahrungen mit Montageanforderungen für 3D-Drucker
Beschreibung		Sicherstellung einer unkomplizierten Montage des Systems auf handelsüblichen 3D-Druckern.	
Wechselwirkungen		Direkte Auswirkung auf die Benutzerfreundlichkeit und die Zeit für die Systemintegration.	
Risiken		Risiko einer übermäßig komplexen Montage, die die Akzeptanz und Nutzung des Systems einschränken könnte.	
Vergleich mit bestehenden Lösungen		Vergleich mit anderen Systemen, die eine einfache Montage auf handelsüblichen 3D-Druckern ermöglichen.	
Schätzung des Aufwands		8 Wochen	

8.1.5 Software

Nr. / ID	040	Nichttechnischer Titel	Steuerungsprogramm
Priorität	Sehr hoch	Quellen	Branchenstandards, Softwareentwicklungsdocumentation

Software bezieht sich auf die Programme und Anwendungen, die für die Steuerung und Verwaltung des 3D-Druckprozesses verwendet werden. Im Kontext dieses Projekts beinhaltet die Software die präzise Steuerung der Faserposition und -ausrichtung, die Visualisierung des Layouts, eine intuitive grafische Benutzeroberfläche (GUI) für die Steuerung der Faserposition und -ausrichtung sowie die Konvertierung in G-Code. Die Anforderungen an die Software werden auf folgende Punkte aufgeteilt:

- Präzise Steuerung der Faserposition und -ausrichtung
- Visualisierung des Layouts
- Intuitive GUI für die Steuerung der Faserposition und -ausrichtung
- Automatisierung der Faserverlegung mittels Algorithmen
- Exportieren in G-Code

Präzise Steuerung der Faserposition und -ausrichtung

Nr. / ID	041	Nichttechnischer Titel	Positionierung und Ausrichtung der Fasern
Priorität	Hoch	Quellen	Softwaretechnische Literatur über 3D-Drucksteuerung
Beschreibung	Gewährleistung, dass die Software in der Lage ist, die genaue Position und Ausrichtung der Fasern auf einem Bauteil festzulegen.		
Wechselwirkungen	Direkter Einfluss auf die Faseranordnung und -verbindung im gedruckten Bauteil, was sich auf dessen mechanische Eigenschaften und Leistung auswirkt.		
Risiken	Risiko von Fehlausrichtungen und Ungenauigkeiten bei der Faserpositionierung kann zu Fehlern in der physischen Fertigung und dadurch zu teuren Materialverschwendungen führen.		
Vergleich mit bestehenden Lösungen	Es gibt keine bestehenden Möglichkeiten, zu kontrollierne wie bei einem Bauteil die Fasern verlegt werden sollen.		
Schätzung des Aufwands	12 Wochen		

Visualisierung des Layouts

Nr. / ID	042	Nichttechnischer Titel	Grafische Darstellung des Faserlayouts
Priorität	Sehr Hoch	Quellen	Literatur über Benutzeroberflächen im 3D-Druck
Beschreibung	Ein unmittelbares visuelles Feedback bei Faserverlegungen		
Wechselwirkungen	Direkte Auswirkung auf die Benutzererfahrung und -effizienz beim Festlegen des Faserlayouts, was sich auf die Gesamtnutzerzufriedenheit auswirkt.		
Risiken	Risiko einer unzureichenden Visualisierung, was die Benutzerzufriedenheit schwächen könnte.		
Vergleich mit bestehenden Lösungen	Vergleich mit anderen 3D-Druck-Softwarelösungen, die benutzerfreundliche Schnittstellen für die Anpassung von Druckeigenschaften bieten.		
Schätzung des Aufwands	10 Wochen		

Intuitive GUI für die Steuerung der Faserposition und -ausrichtung

Nr. / ID	043	Nichttechnischer Titel	Benutzerfreundliche grafische Oberfläche für Faserverlegung
Priorität	Mittel	Quellen	Empfohlene Praktiken für benutzerfreundliche Software
Beschreibung	Gewährleistung einer intuitiven und leicht verständlichen grafischen Oberfläche, um eine benutzerfreundliche Interaktion zu ermöglichen.		
Wechselwirkungen	Direkte Auswirkung auf die Benutzererfahrung und -effizienz bei der Festlegung des Faserlayouts, was sich auf die Gesamtnutzerzufriedenheit auswirkt.		
Risiken	Risiko einer komplizierten oder unklaren Benutzeroberfläche, was zu Verwirrung und möglicher Fehlinterpretation des Faserlayouts führen könnte.		
Vergleich mit bestehenden Lösungen	Vergleich mit anderen 3D-Druck-Softwarelösungen, die benutzerfreundliche Schnittstellen für die Anpassung von Druckeigenschaften bieten.		
Schätzung des Aufwands	10 Wochen		

Automatisierung der Faserverlegung mittels Algorithmus

Nr. / ID	102	Nichttechnischer Titel	Automatisierte Faserverlegung für einfachere Handhabung
Priorität	Hoch	Quellen	Forschungspublikationen über automatisierte (Faserverlegungs-)Algorithmen
Beschreibung		Implementierung eines Algorithmus zur automatischen Verlegung von Fasern in Bauteilen, um den Prozess zu vereinfachen und zu beschleunigen.	
Wechselwirkungen		Direkte Auswirkung auf die Effizienz und Geschwindigkeit der Faserverlegung, was sich auf die Gesamtleistung des Druckprozesses auswirkt.	
Risiken		Risiko von Zeit- und Ressourcenverlusten bei der Implementierung und Optimierung des Algorithmus, insbesondere wenn komplexe Geometrien berücksichtigt werden müssen.	
Vergleich mit bestehenden Lösungen		Vergleich mit anderen automatisierten Faserverlegungslösungen und Algorithmen, die eine effiziente und präzise Faserverlegung ermöglichen.	
Schätzung des Aufwands		16 Wochen	

Exportieren von G-Code

Nr. / ID	044	Nichttechnischer Titel	Übersetzung in Maschinensprache
Priorität	Sehr Hoch	Quellen	G-Code-Standards, Dokumentationen zu Firmware-Kompatibilität
Beschreibung		Gewährleistung einer reibungslosen und korrekten Konvertierung des definierten Faserlayouts in G-Code, der von der eigenen Firmware verarbeitet werden kann	
Wechselwirkungen		Direkte Auswirkung auf die Druckqualität und -präzision, abhängig von der Genauigkeit des konvertierten G-Codes.	
Risiken		Risiko von Konvertierungsfehlern, die zu Druckfehlern und Qualitätsminderung führen könnten, und Risiko von Inkompatibilitäten mit der Firmware.	
Vergleich mit bestehenden Lösungen		Vergleich mit anderen 3D-Druck-Softwarelösungen, die eine effiziente Konvertierung von Layouts in G-Code ermöglichen.	
Schätzung des Aufwands		6 Wochen	

8.1.6 Firmware

Nr. / ID	050	Nichttechnischer Titel	Steuerungssoftware
Priorität	Hoch	Quellen	Firmware-Dokumentationen, Entwicklerforen

Die Firmware ist die Software, die in elektronischen Geräten eingebettet ist und die Hardware steuert. In Bezug auf den 3D-Drucker ist die Firmware verantwortlich für die Interpretation von G-Code-Anweisungen und die Steuerung der Bewegungen und Prozesse des Druckers. Die benutzte Firmware soll den G-Code, welcher mit der Software generiert wird, interpretieren und verarbeiten können. Zudem soll der Prozess, die Firmware zu implementieren, simpel gehalten werden. Die Anforderungen an die Firmware werden auf folgende Punkte aufgeteilt:

- G-Code-Verarbeitung der Firmware
- Unkomplizierte Firmware-Integration für 3D-Drucker

G-Code-Verarbeitung der Firmware

Nr. / ID	051	Nichttechnischer Titel	Firmware-basierte Befehlsverarbeitung für den 3D-Druck
Priorität	Hoch	Quellen	G-Code-Standards, Firmware-Dokumentationen
Beschreibung	Gewährleistung einer reibungslosen Interpretation und Verarbeitung des durch die Software generierten G-Codes mit der verwendeten Firmware.		
Wechselwirkungen	Direkte Auswirkung auf die Druckpräzision und -qualität, abhängig von der Genauigkeit und Effizienz der G-Code-Verarbeitung der Firmware.		
Risiken	Risiko von Interpretationsfehlern, die zu Fehlern im Druckprozess und Qualitätsminderung führen könnten, und Risiko von Inkompatibilitäten zwischen G-Code und Firmware.		
Vergleich mit bestehenden Lösungen	Vergleich mit anderen 3D-Drucker-Firmwarelösungen, die eine effiziente Interpretation und Verarbeitung von G-Codes gewährleisten.		
Schätzung des Aufwands	8 Wochen		

Unkomplizierte Firmware-Integration für 3D-Drucker

Nr. / ID	052	Nichttechnischer Titel	Einfache Einbindung der Steuerungssoftware
Priorität	Hoch	Quellen	Empfohlene Spezifikationen für 3D-Druckgeschwindigkeiten
Beschreibung	Gewährleistung einer benutzerfreundlichen und unkomplizierten Implementierung der Firmware, um den Einrichtungsprozess zu erleichtern und zu beschleunigen.		
Wechselwirkungen	Direkte Auswirkung auf die Benutzererfahrung und die Effizienz bei der Einrichtung des Druckers, was sich auf die Gesamtnutzerzufriedenheit auswirkt.		
Risiken	Risiko einer komplexen Implementierung, die zu Verzögerungen bei der Inbetriebnahme und möglichen Fehlern führen könnte.		
Vergleich mit bestehenden Lösungen	Vergleich mit anderen 3D-Drucker-Firmwarelösungen, die eine einfache Implementierung und Einrichtung gewährleisten.		
Schätzung des Aufwands	6 Wochen		

9 Anhang

9.1 Simulationslasten

Im Folgenden wird dargestellt, an welchen Stellen die Simulationslasten am simulierten Objekt angreifen. Dabei dient die Version 5.0.0 als Referenzmodell, da die Lastverteilung für die übrigen Versionen in ähnlicher Weise gilt.

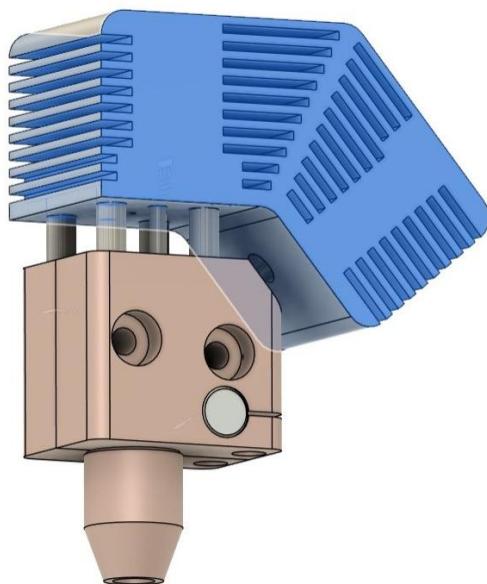


Abb. 149: Konvektion 1



Abb. 150: Konvektion 2

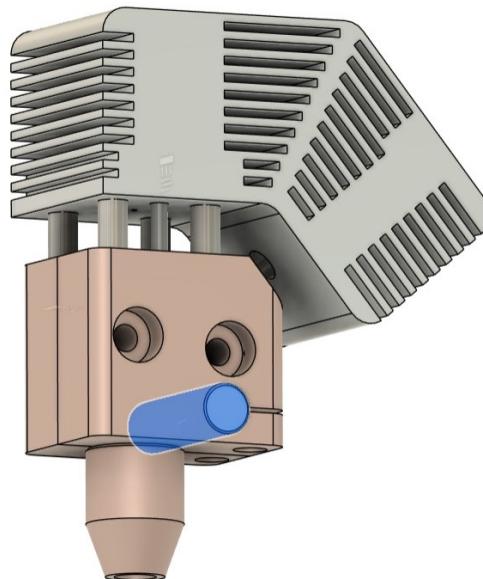


Abb. 151: Angewendete Temperatur

9.2 Materialliste

In diesem Abschnitt wird die Materialien die zum testen und fertien gekauft wurden aufgelistet

Tabelle 22: Übersicht der Materialkosten mit Lieferanten

Bezeichnung	Lieferant	Preis/Stk.	Stückzahl	Gesamtpreis
Bohrbuchse	Norelem	€ 2,42	2	€ 4,84
Passstift 3m6 x 16	Norelem	€ 0,16	5	€ 0,80
Passstift 2,5m6 x 28	Norelem	€ 0,21	2	€ 0,42
Hot Block Hardware	3D Jake	€ 7,19	1	€ 7,19
Heatbreak	3D Jake	€ 35,99	1	€ 35,99
Fan	Amazon	€ 8,58	2	€ 17,16
Nema 14	OMC	€ 9,91	1	€ 9,91
Servo	Amazon	€ 7,04	1	€ 7,04
2 x 1mm Kugelkopffräser	Amazon	€ 8,48	1	€ 8,48
5 x 1mm 4Z Schaftfräser	Amazon	€ 13,22	1	€ 13,22
DIN 4762 M3x12	Norelem	€ 0,30	8	€ 2,40
5 x 1mm 1Z Schaftfräser	Amazon	€ 15,56	1	€ 15,56
Gewinde Fräser M5 - M10	Sorotec	€ 44,90	1	€ 44,90
Carbon Faser 3k	easycomposites	€ 10,93	1	€ 10,93
Raspberry Pi Display	Amazon	€ 40,33	1	€ 40,33
Microbohrer	Amazon	€ 6,04	1	€ 6,04
Kanülen 22G 100stk	Amazon	€ 17,14	1	€ 17,14
PTFE tube 1mm AD 0.5mm ID	Amazon	€ 11,55	1	€ 11,55
Alu-Rohr 3 x 0.45 x 305 mm 4 Stk.	Kirchert	€ 6,39	1	€ 6,39
5pcs 24V 70W Heizelement	Amazon	€ 18,14	1	€ 18,14
5 Stücke Hartmetall Schaftfräser 1mm, 4 Zähne	Amazon	€ 11,00	1	€ 11,00
Sägeblatt Kreissägeblatt Set JTENG für Dremel	Amazon	€ 8,05	1	€ 8,05
UniTak3D Ender 3 All Metal Smooth Heatbreak	Amazon	€ 9,06	1	€ 9,06
5 Stk 3D-Drucker Thermistor Temperatursensor NTC 3950 100K	Amazon	€ 7,99	1	€ 7,99
6 Stück Messingrohr	Amazon	€ 10,07	1	€ 10,07
2 Stück Aramid 1313 feuerfeste flammhemmende Nähgarn	Aliexpress	€ 24,01	1	€ 24,01
200 g Glasfaser-Nähgarn	Aliexpress	€ 10,53	1	€ 10,53
Sägeblatt Dorn	Aliexpress	€ 12,82	1	€ 12,82
16mm-110mm Wolframstahl-Fräsblatt Vollhartmetall-Kreissägeblatt-Fräser	Aliexpress	€ 9,64	1	€ 9,64
Kevlar Aramid Nähgarn	Aliexpress	€ 10,01	1	€ 10,01
Gesamtpreis				€ 391.61

9.3 Stückliste

In diesem Abschnitt wird die Stückliste der Elektronikkomponenten aufgelistet, die für den Bau des FiberPrinters benötigt werden. Die Stückliste enthält die Bezeichnung, die Seriennummer, den Kaufort, die Anzahl und den Preis der Komponenten.

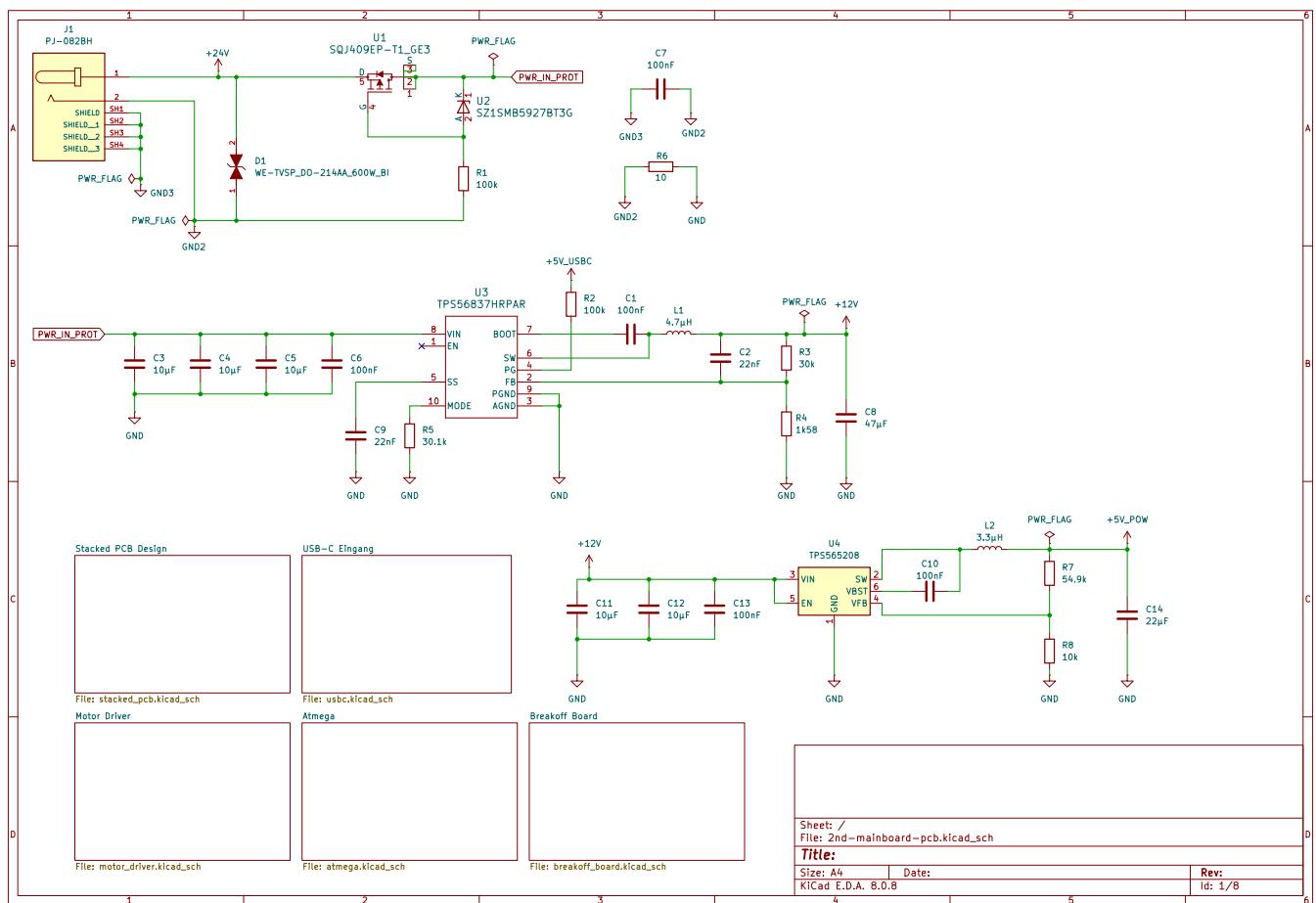
Tabelle 23: Stückliste der Elektronikkomponenten

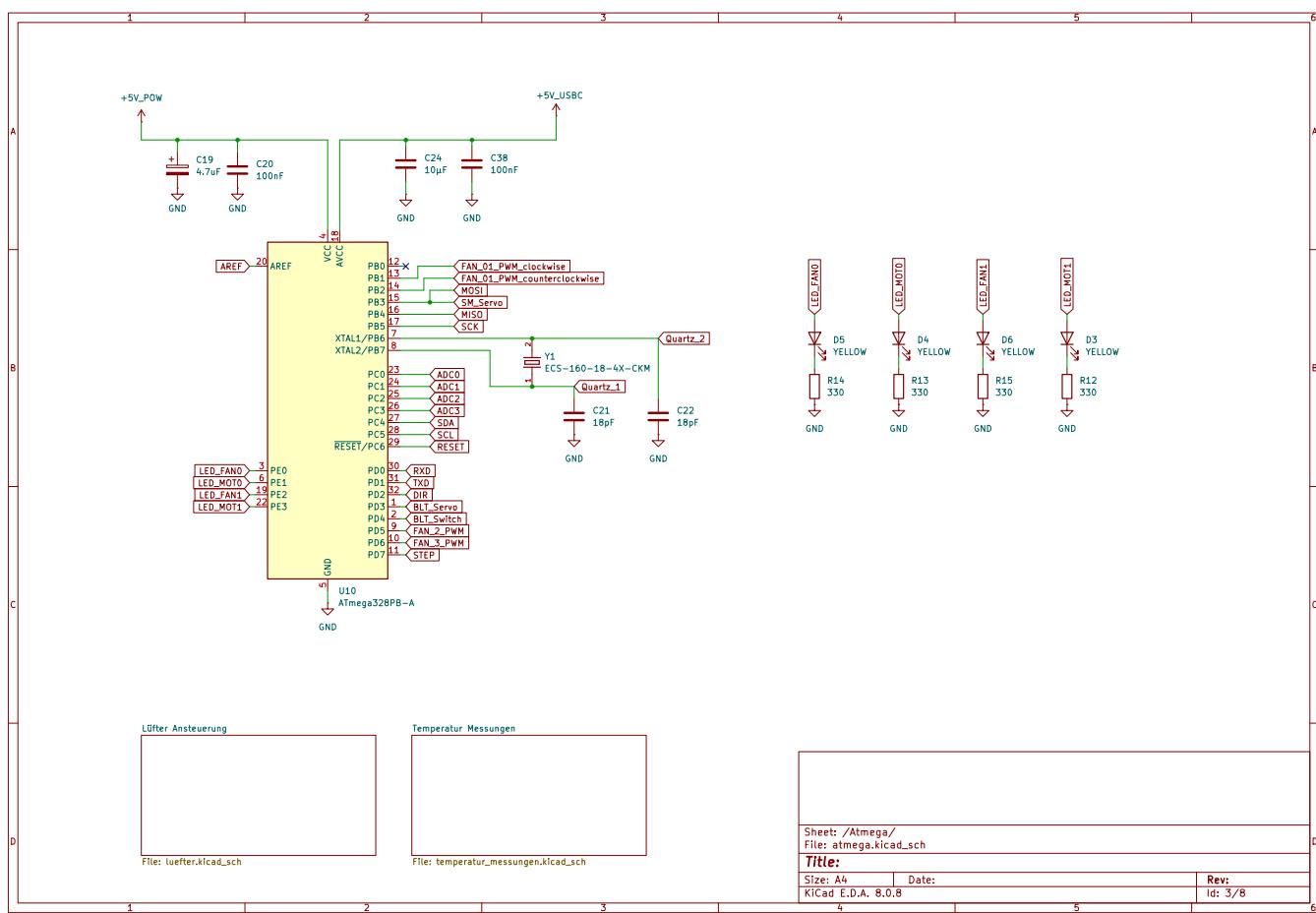
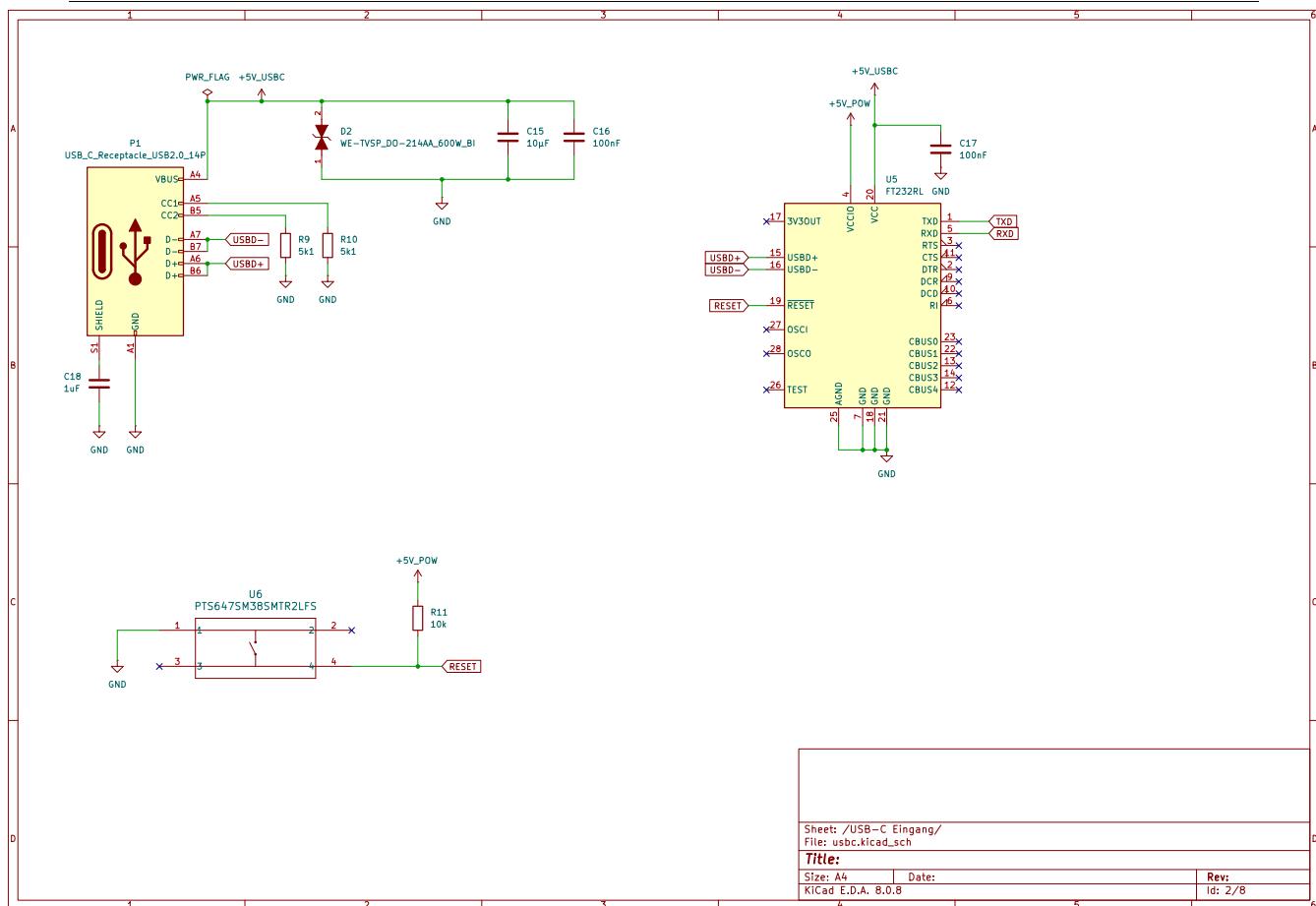
Bezeichnung	Seriennummer	Lieferant	St.	Preis
Raspberry Pi	Raspberry Pi 3B+	DigiKey	1	€ 33.37 (Digikey, 2025)
Microchip	ATMEGA328P-AU	DigiKey	1	€ 2.55 (DigiKey, 2025h)
USB-Serial Wandler	FT232RNL-TUBE	DigiKey	1	€ 4.60 (DigiKey, 2025n)
Quarz-Oszillator	ECS-160-18-4X-CKM	DigiKey	1	€ 0.41 (DigiKey, 2025m)
Reset-Button	PTS647SM38SMTR2 LFS	DigiKey	1	€ 0.23 (DigiKey, 2025t)
Schrittmotor	14HM08-0504S	Stepperonline	1	€ 9.91 (Stepperonline, 2024)
Schrittmotor-Treiber	DRV8825	AZ-Delivery	1	€ 6.99 (AZ-Delivery, 2025a)
Servo	MG90S	AZ-Delivery	1	€ 6.99 (AZ-Delivery, 2025c)
Nivellierungssensor	BLTouch	3D-Jake	1	€ 39.99 (3DJake, 2019)
Beschleunigungssensor	GY-521 MPU-6050	AZ-Delivery	1	€ 5.99 (AZ-Delivery, 2025b)
Lüfter	CFM-2507CF-1140-313	DigiKey	2	€ 13.46 (DigiKey, 2025k)
Lüfter	CFM-2507CF-0140-313	DigiKey	2	€ 14.80 (DigiKey, 2025j)
Temperaturelement	B57550G1103F005	DigiKey	4	€ 10.00 (DigiKey, 2025l)
2 Pin Stecker	6.65002E+11	DigiKey	4	€ 0.52 (DigiKey, 2025b)
2 Pin Buchse	6.65102E+11	DigiKey	4	€ 2.44 (DigiKey, 2025c)
12-40V zu 40V Buck	TPS56837HRPAR	DigiKey	1	€ 2.34 (DigiKey, 2025ad)
12V zu 5V Buck	TPS565208DDCR	DigiKey	1	€ 1.17 (DigiKey, 2025ac)
Spule 12V Buck	SPM10040T-4R7M-HZ	DigiKey	1	€ 1.78 (DigiKey, 2025y)
Spule 5V Buck	IHLP2525CZER3R3M11	DigiKey	1	€ 0.95 (DigiKey, 2025q)
P-MOSFET	SQJ409EP-T1_GE3	DigiKey	1	€ 1.56 (DigiKey, 2025z)
Zener Diode	SZ1SMB5927BT3G	DigiKey	1	€ 0.47 (DigiKey, 2025aa)
TSV Diode	824521241	DigiKey	2	€ 0.60 (DigiKey, 2025d)
N-MOSFET	RJK0456DPB-00#J5	DigiKey	2	€ 5.82 (DigiKey, 2025u)
Bipolar Motor Driver	TPM16050-S6TR	DigiKey	1	€ 0.60 (DigiKey, 2025ab)
USB-C Kabel	A-USB31C-20A-150A	DigiKey	1	€ 9.25 (DigiKey, 2025g)
USB-C Buchse	USB4085-GF-A	DigiKey	1	€ 0.84 (DigiKey, 2025af)
V _{in} -Buchse	PJ-082BH	DigiKey	1	€ 1.70 (DigiKey, 2025s)
Vin-Stecker	50-00619	DigiKey	1	€ 0.94 (DigiKey, 2025a)
Leistungskabel	CF881-15-02	DigiKey	6	€ 17.22 (DigiKey, 2025i)
Buchse 4 pin	S4B-XH-A-1	DigiKey	1	€ 0.22 (DigiKey, 2025x)
Buchse 2 pin	S3B-XH-A-1	DigiKey	4	€ 0.72 (DigiKey, 2025w)
Buchse 3 pin	S2B-XH-A-1	DigiKey	2	€ 0.28 (DigiKey, 2025v)
3x2 Stecker	FTSH-103-01-L-DH	DigiKey	1	€ 1.24 (DigiKey, 2025o)
2 Pin Bridge	M50-1900005	DigiKey	3	€ 0.96 (DigiKey, 2025r)
3 Pin Header	FTS-103-01-F-S	DigiKey	2	€ 1.40 (DigiKey, 2025c)
Abstandhalter	971070154	DigiKey	8	€ 7.04 (DigiKey, 2025e)
Abstandhalter	971110154	DigiKey	4	€ 3.00 (DigiKey, 2025f)
Steckverbinder Female	HLE-107-02-L-DV	DigiKey	2	€ 5.20 (DigiKey, 2025p)
Steckverbinder Male	TSM-107-04-L-DV	DigiKey	1	€ 1.53 (DigiKey, 2025ae)
Steckverbinder Male	HW-07-08-L-D-250-SM	Mouser	1	€ 2.17 (Electronics, 2025)
Gesamtpreis				€ 221.25

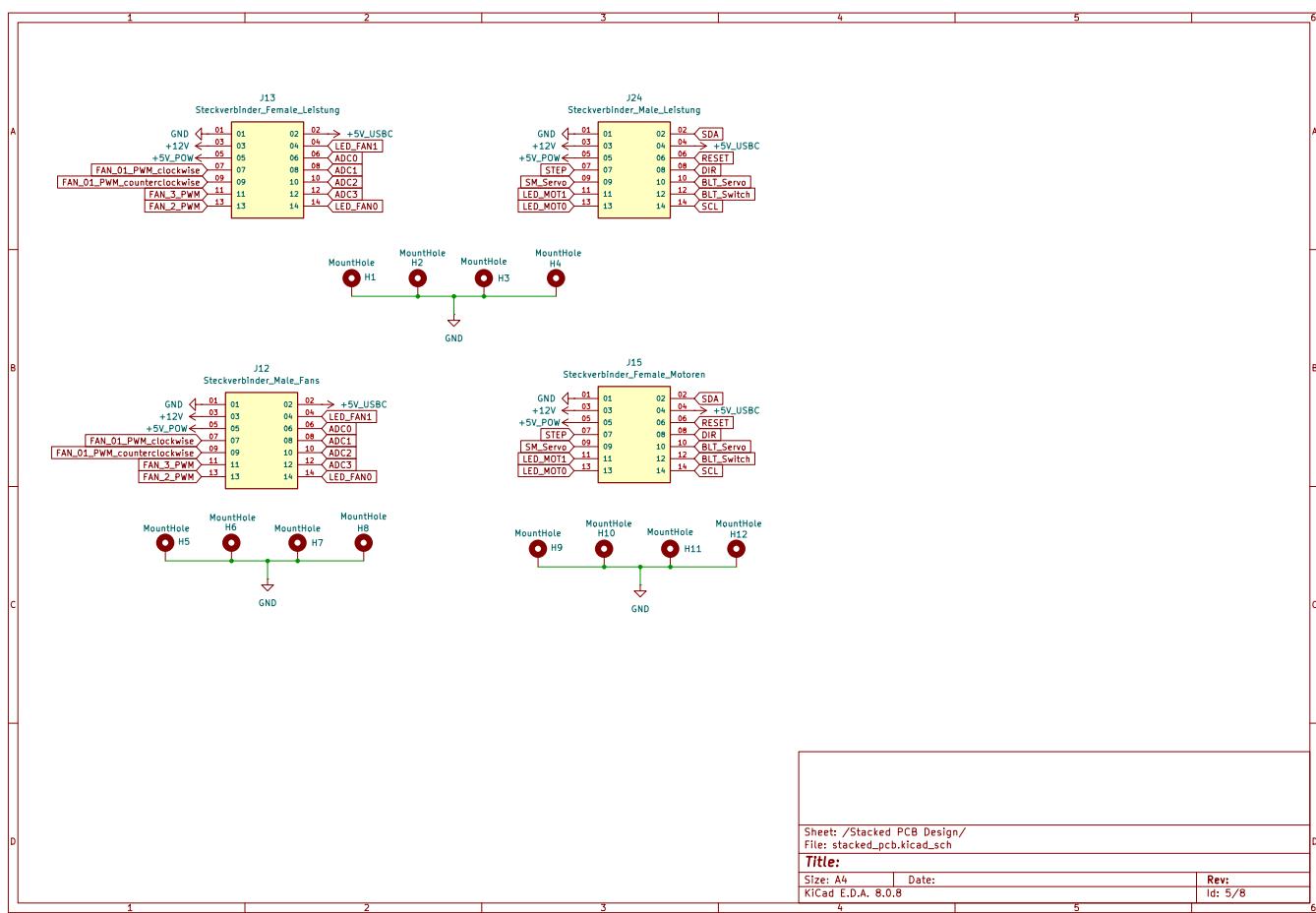
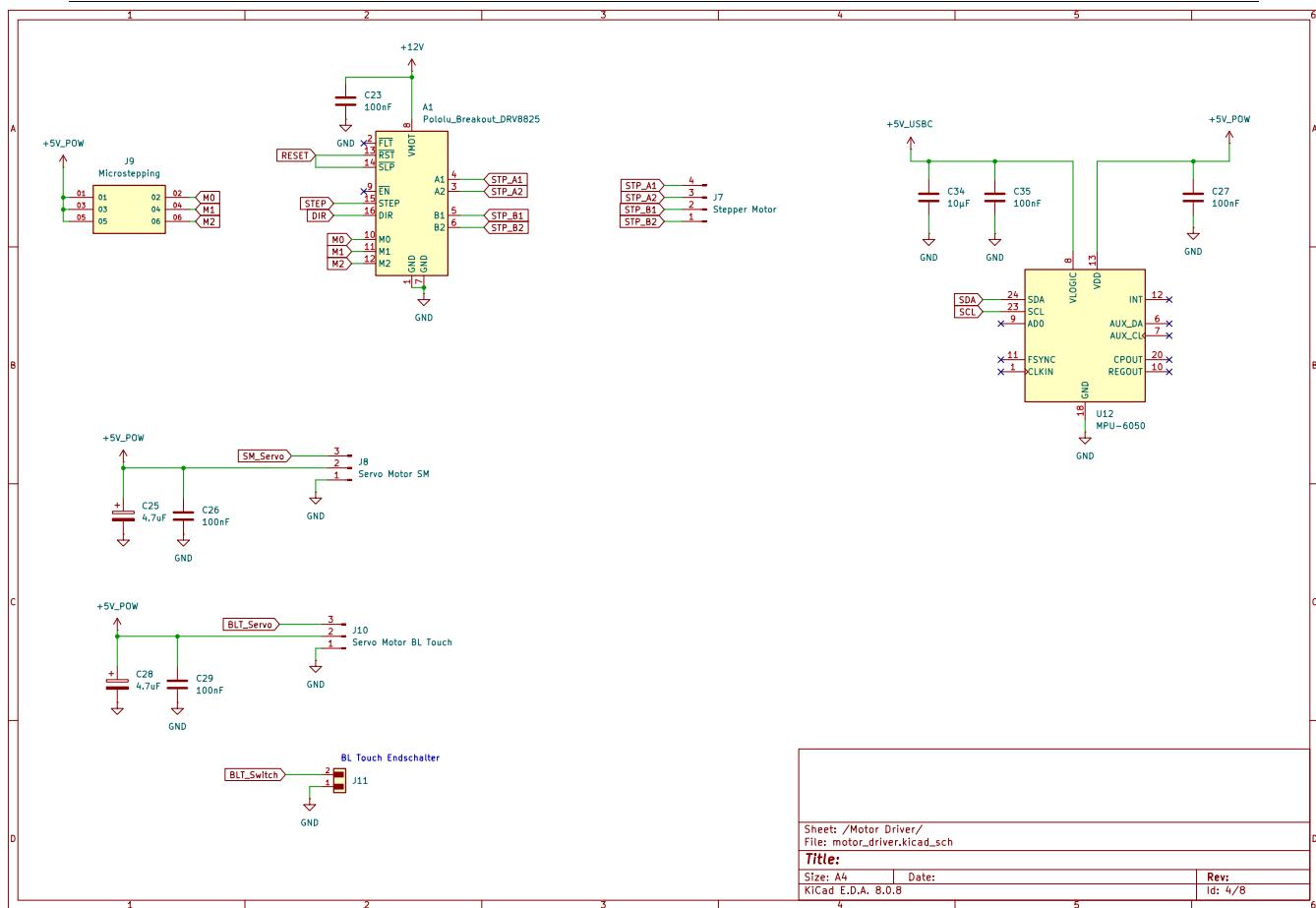
Die Stückliste 23 zeigt dass die Elektronikkomponenten für den FiberPrinter insgesamt € 221.25 kosten.

9.4 Vollständiger Schaltplan

Im folgenden Abschnitt wird der vollständige Schaltplan des FiberPrinters gezeigt. Der Schaltplan ist in mehrere Abschnitte unterteilt, um die Übersichtlichkeit zu verbessern.

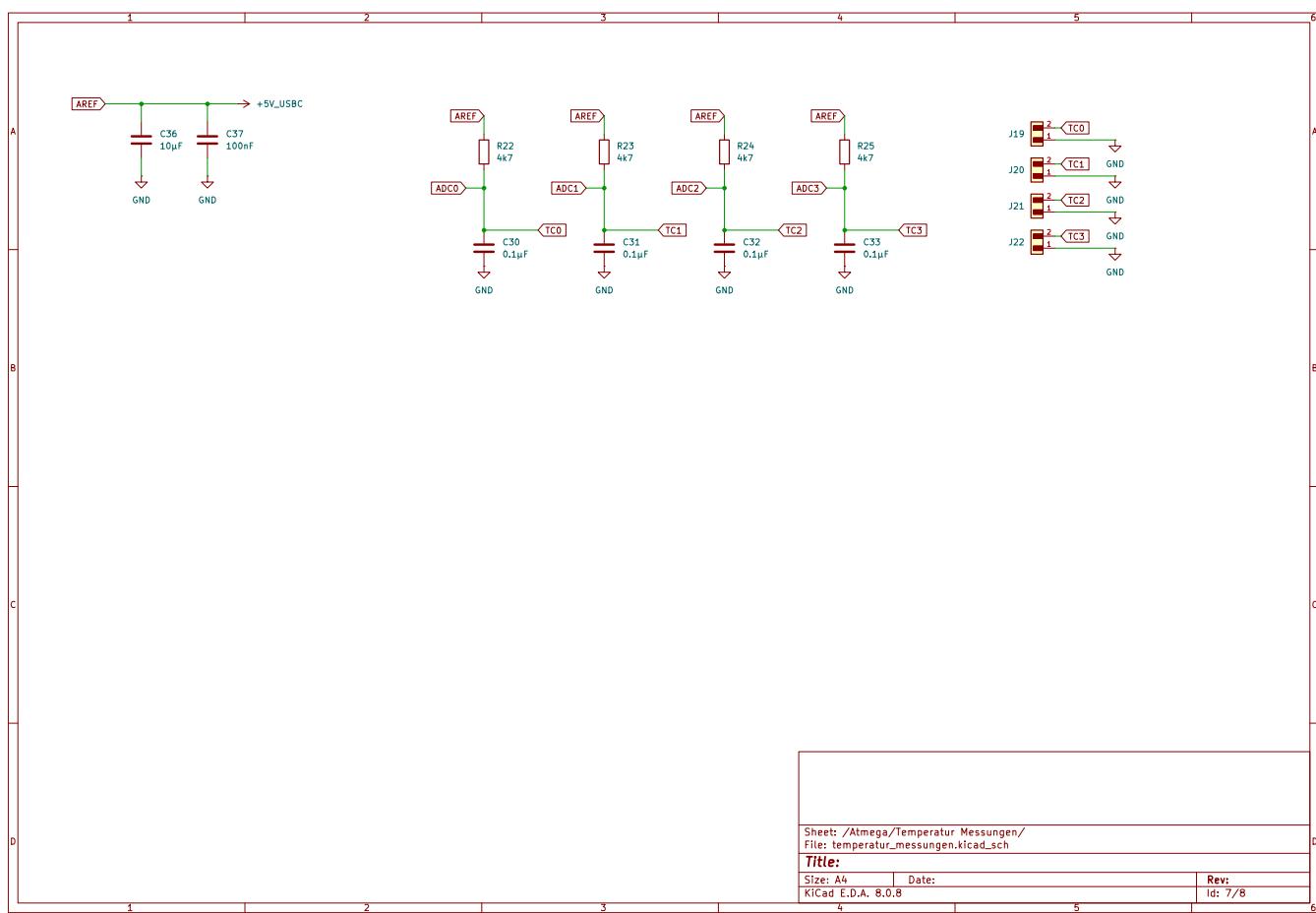
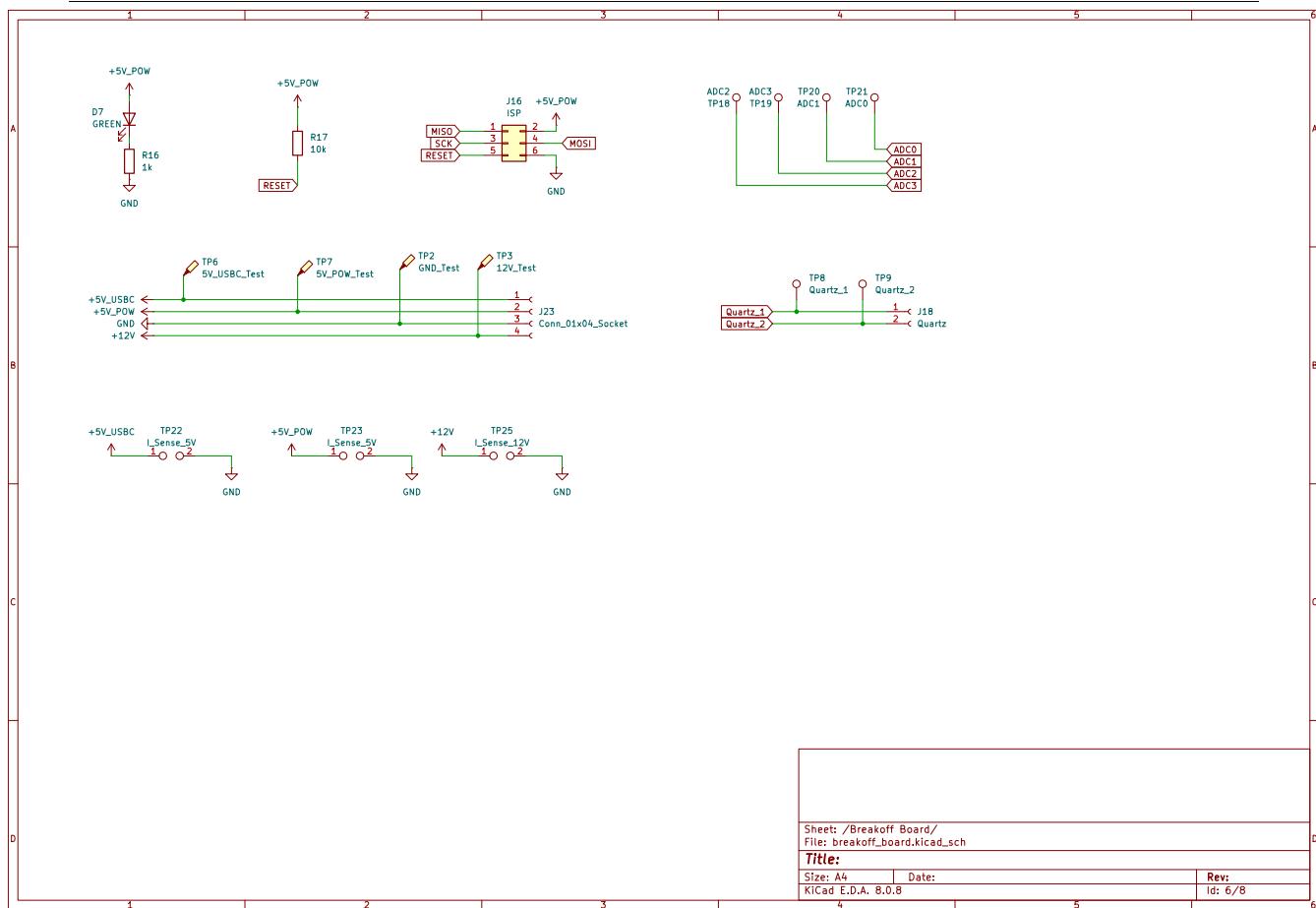


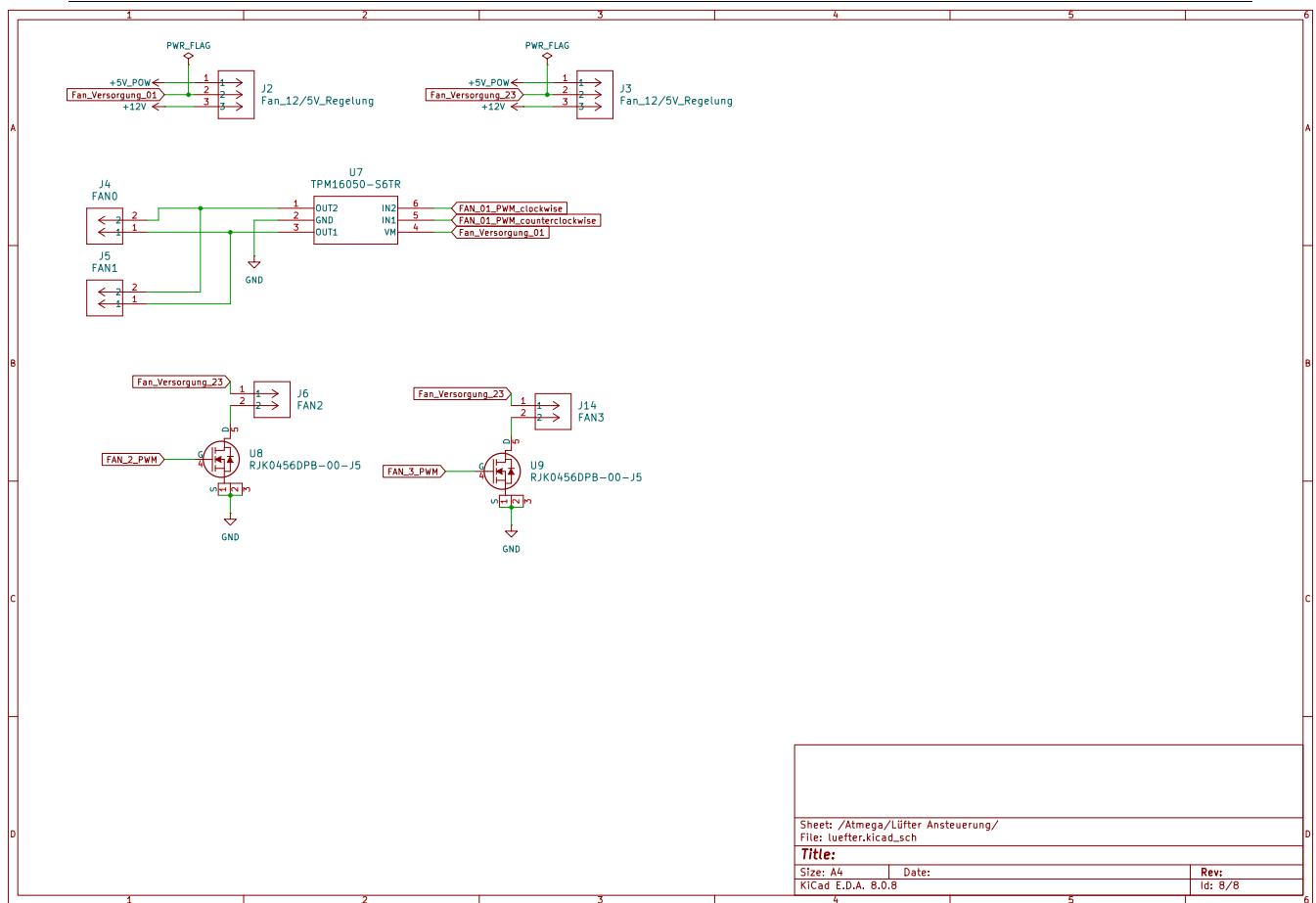




9.4 Vollständiger Schaltplan

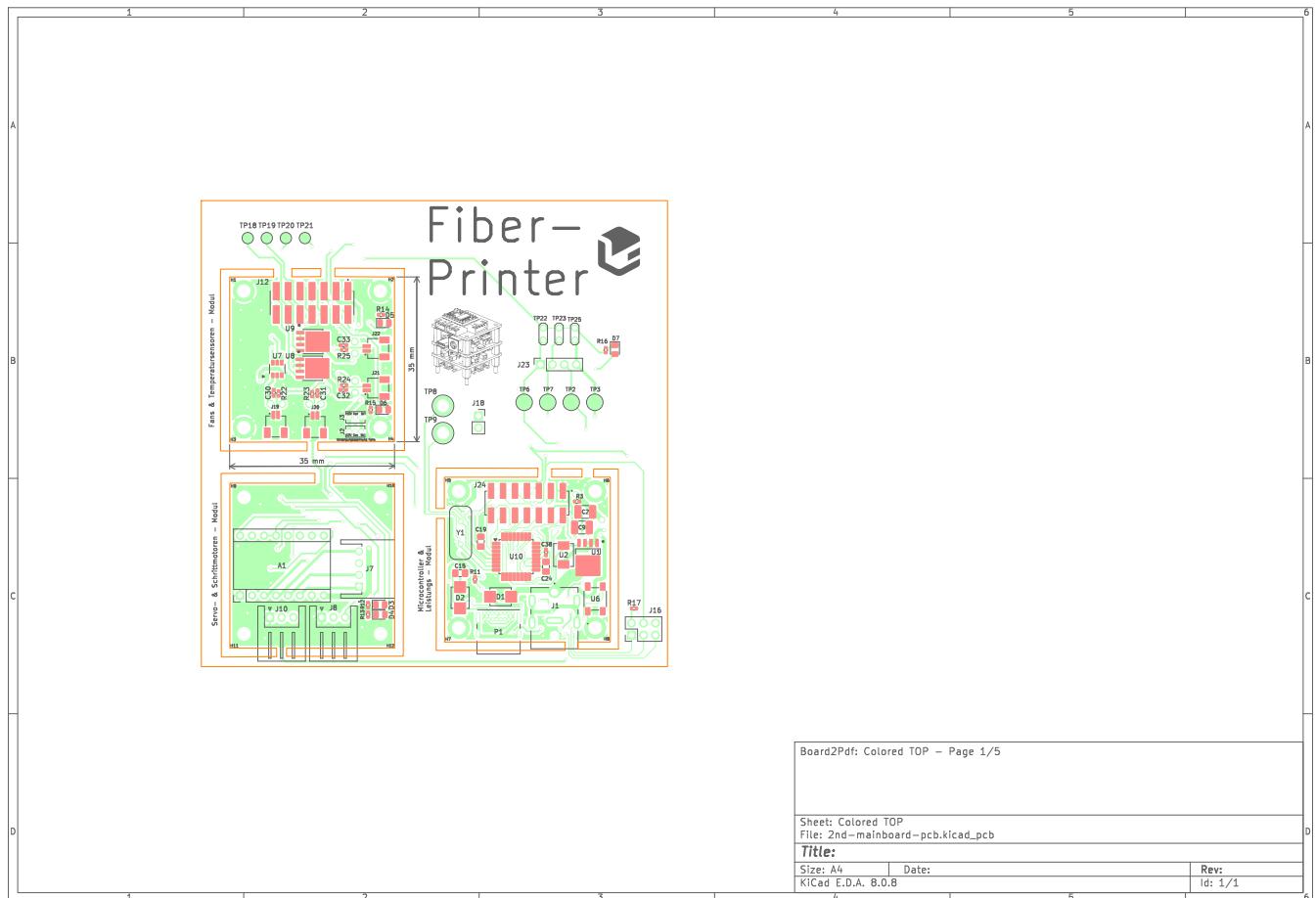
9 ANHANG

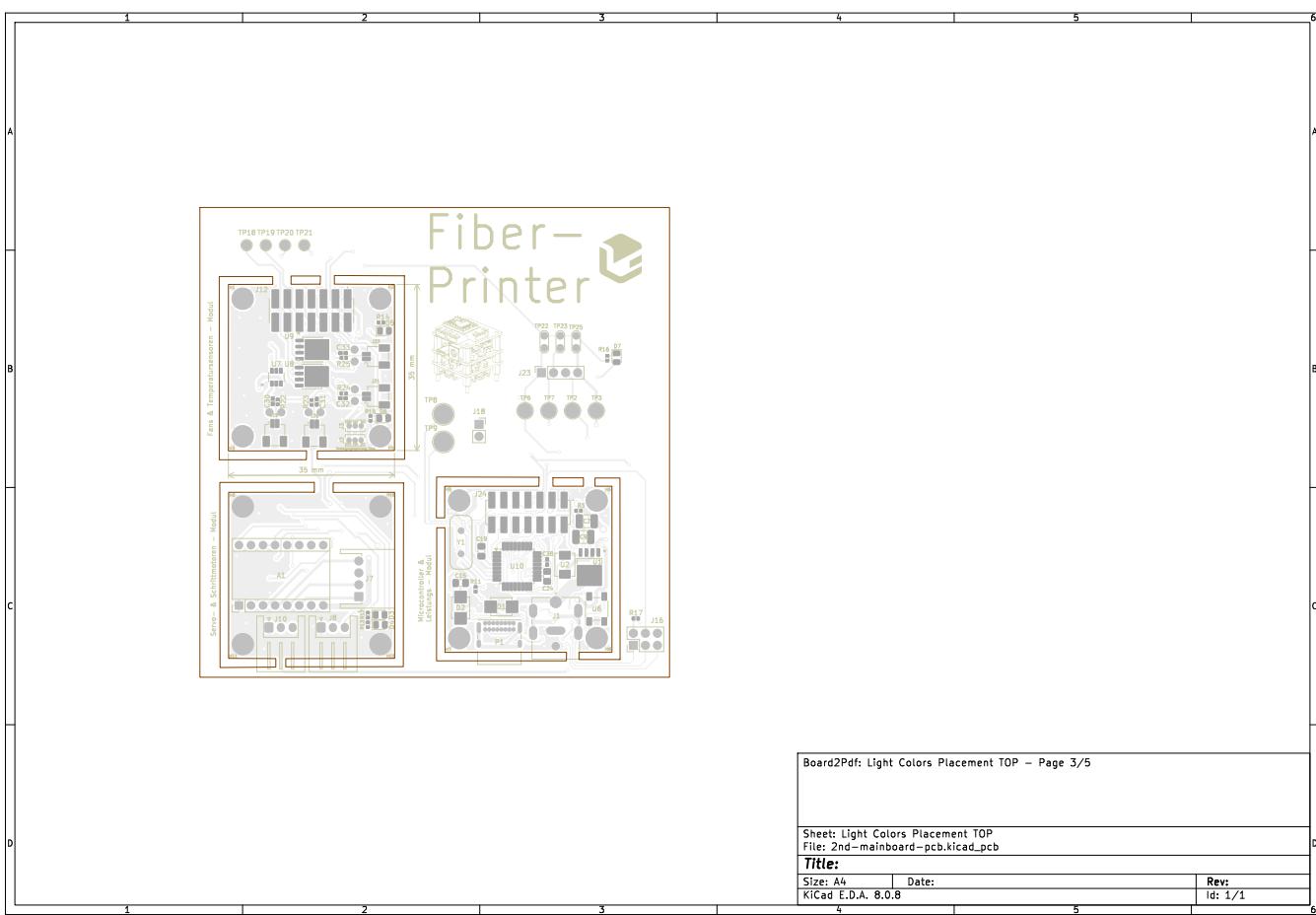
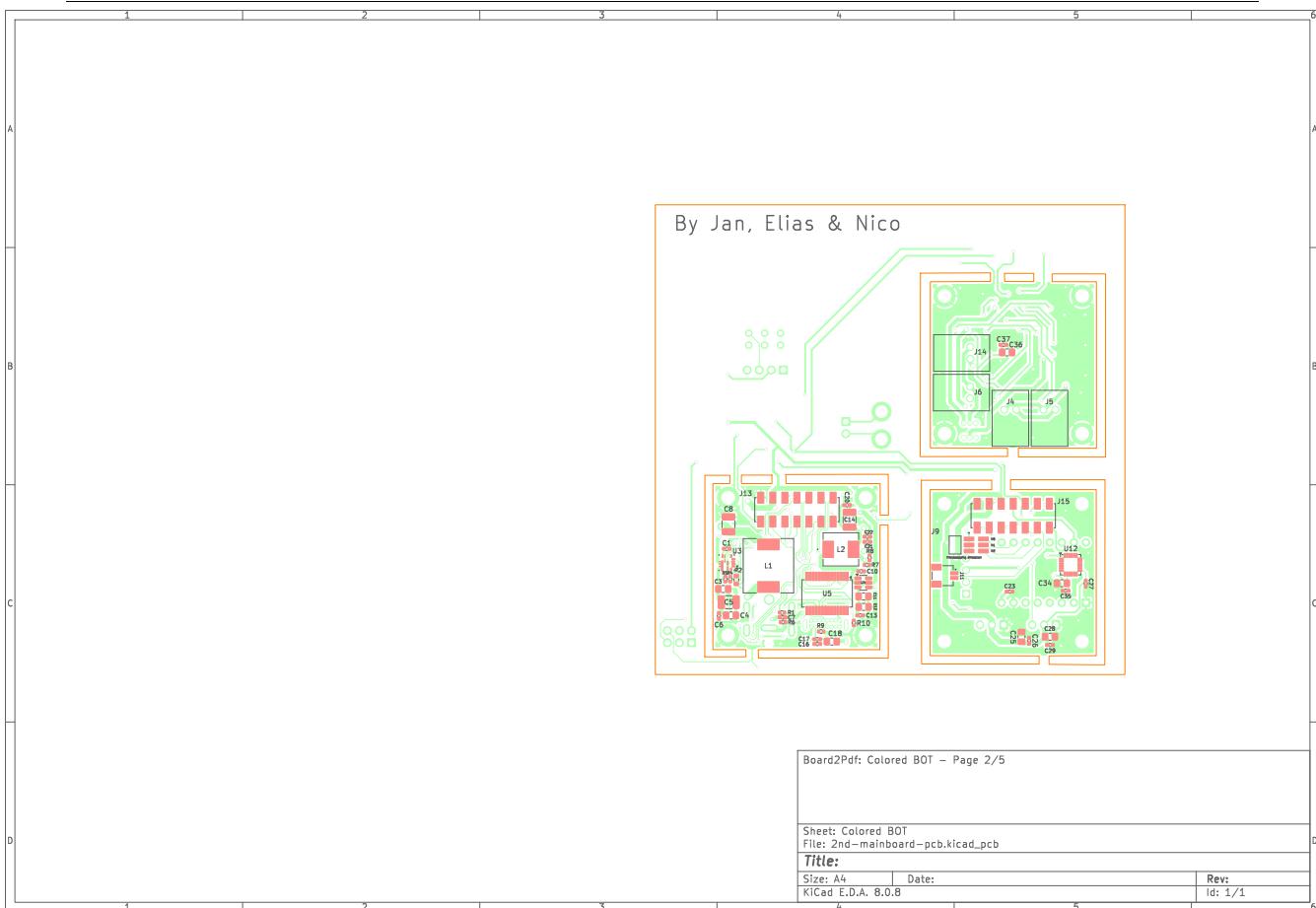


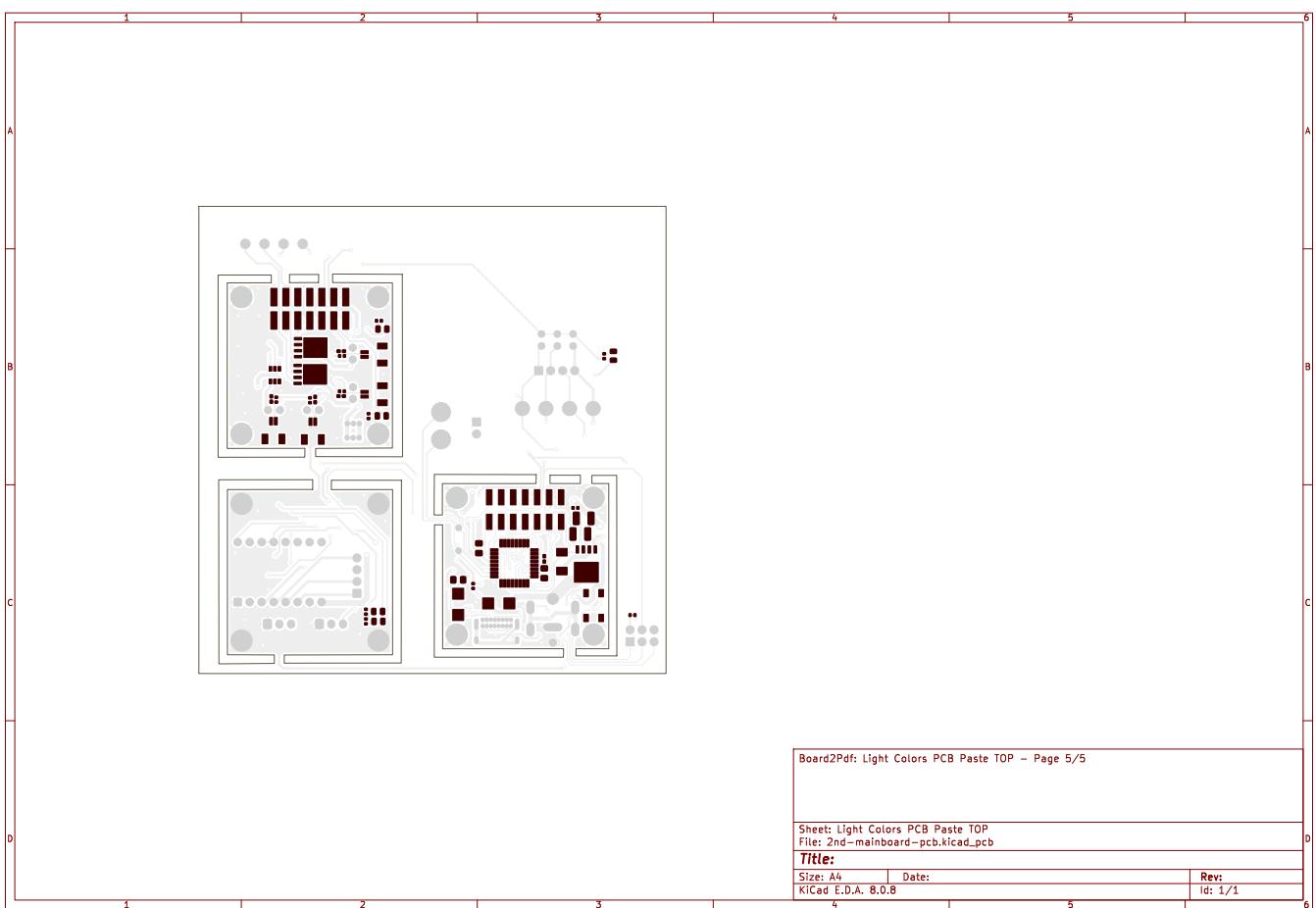
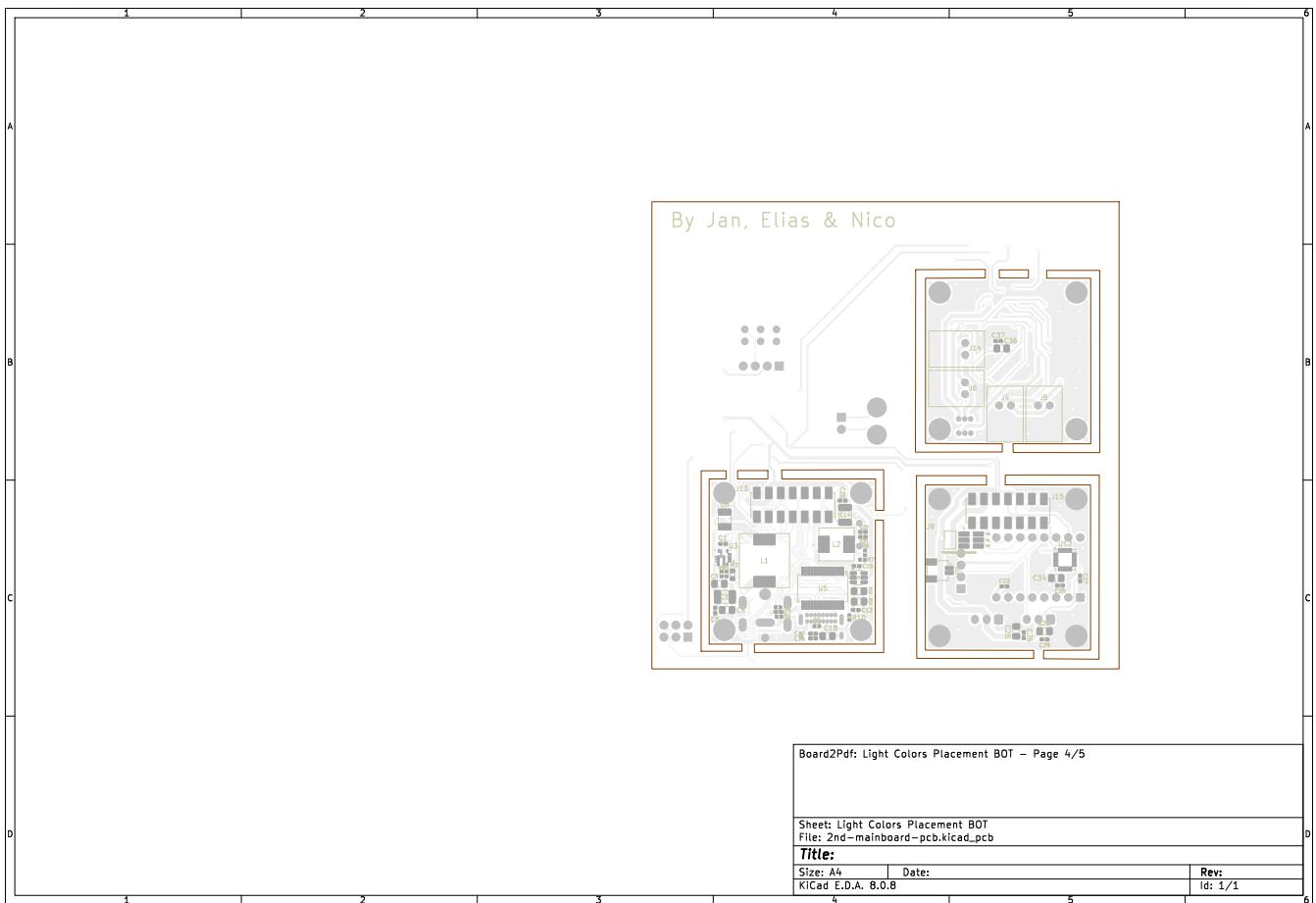


9.5 Vollständiges Layout

Im folgenden Abschnitt wird das vollständige Layout des FiberPrinters gezeigt. Das Layout ist in mehreren Perspektiven dargestellt, um die Übersichtlichkeit zu verbessern.







9.6 Datenblatt des DC-Motor-Controllers TPM16050

Im folgenden Abschnitt werden die wichtigsten Informationen aus dem Datenblatt des DC-Motor-Controllers TPM16050 (DigiKey, 2025ab) gezeigt.



Features

- 17-V H-bridge Driver
- MOSFET On-resistance $R_{ds(on)}$ HS + LS 0.85 Ω
- 0.5-A Max Output Current
- Supports 3-V to 17-V Operating Supply Voltage
- Support 3.3-V, 5-V PWM (IN1/IN2) interface
- Protection
 - Undervoltage Lockout (UVLO)
 - Over-Current Protection (OCP)
 - Thermal Shutdown (TSD)
- Small Package Footprint
 - SOT23-6 Package

Description

The TPM16050 is a high-voltage H-bridge driver dedicated for IR-CUT control in surveillance camera. It is optimized to control the IR-CUT with 20Ω impedance and 3.3V, 5V or 12V supply voltage. It can provide up-to 0.5-A drive current. TPM16050 can sustain max 17V supply voltage. It also supports an external 26.9Ω/2.5W resistor in serial of supply pin VM, in order to protect low-voltage loads such as IR-CUT from voltage over-stress with 12V supply. Internal protection features such as overcurrent protection, short circuit protection undervoltage lockout and over temperature improve reliability of the whole system.

Applications

- Surveillance Cameras
- E-Lock
- Consumer devices
- Toys

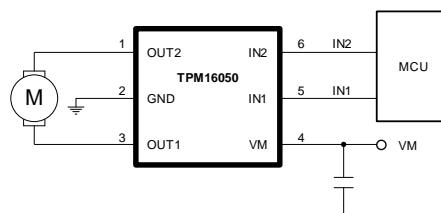


Figure 1 Application Diagram with Direct Supply

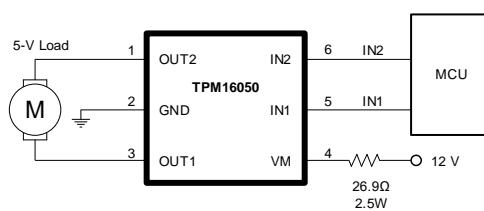


Figure 2 Application Diagram with Power Resistor Ballasting

9.7 Layout Empfehlungen aus dem Datenblatt des TPS56387

Im folgenden Abschnitt werden die wichtigsten Layout Empfehlungen aus dem Datenblatt des TPS56387 (DigiKey, 2025ad) gezeigt.

TPS56837H, TPS56837HA
SLVSHH7A – JANUARY 2024 – REVISED JUNE 2024

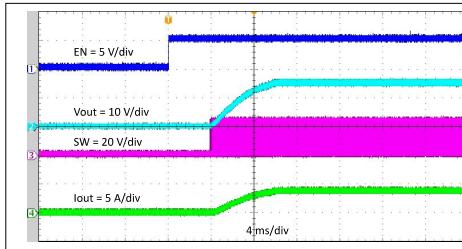


Figure 7-14. Enable Relative to EN

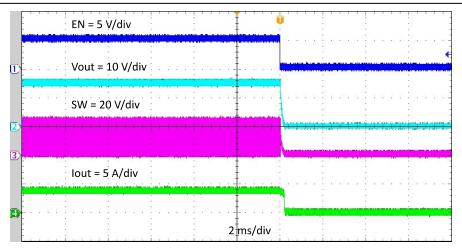


Figure 7-15. Disable Relative to EN

7.3 Power Supply Recommendations

The TPS56837H and TPS56837HA are designed to operate from input supply voltage in the range of 4.5V to 28V. Buck converters require the input voltage to be higher than the output voltage for proper operation. Input supply current must be appropriate for the desired output current. If the input voltage supply is located far from the TPS56837H and TPS56837HA circuit, TI recommends some additional input bulk capacitance.

7.4 Layout

7.4.1 Layout Guidelines

1. Use a four-layer PCB with maximum ground plane partitioning possible for good thermal performance. A 76mm × 76mm, four-layer PCB with 2-1-1-2 oz copper is used as example.
2. Make VIN and PGND traces as wide as possible to reduce trace impedance. The wide areas are also of advantage from the view point of heat dissipation.
3. Put at least two vias for PGND pad for better thermal performance.
4. Place the input capacitor and output capacitor as close to the device as possible to minimize trace impedance.
5. Provide sufficient vias for the input capacitor and output capacitor.
6. Keep the SW trace as physically short and wide as practical to minimize radiated emissions.
7. Do not allow switching current to flow under the device.
8. Keep the SS trace as far as possible to SW trace to minimize coupling during soft start.
9. Connect a separate VOUT path to the upper feedback resistor.
10. Keep the voltage feedback loop away from the high-voltage switching trace, and preferably has ground shield.
11. Make the trace of the VFB node as small as possible to avoid noise coupling. Also keep feedback resistors and the feedforward capacitor near the IC.
12. Make the PGND trace between the output capacitor and the PGND pin as wide as possible to minimize the trace impedance.
13. Note that inner layer 1 is PGND and AGND with the single point net tie.
14. Note that inner layer 2 is PGND for better heat dissipation.

9.8 Layout Empfehlungen aus dem Datenblatt des TPS565208

Im folgenden Abschnitt werden die wichtigsten Layout Empfehlungen aus dem Datenblatt des TPS565208 (DigiKey, 2025ac) gezeigt.



10 Layout

10.1 Layout Guidelines

1. VIN and GND traces should be as wide as possible to reduce trace impedance. The wide areas are also of advantage from the view point of heat dissipation.
2. The input capacitor and output capacitor should be placed as close to the device as possible to minimize trace impedance.
3. Provide sufficient vias for the input capacitor and output capacitor.
4. Keep the SW trace as physically short and wide as practical to minimize radiated emissions.
5. Do not allow switching current to flow under the device.
6. A separate VOUT path should be connected to the upper feedback resistor.
7. Make a Kelvin connection to the GND pin for the feedback path.
8. Voltage feedback loop should be placed away from the high-voltage switching trace, and preferably has ground shield.
9. The trace of the VFB node should be as small as possible to avoid noise coupling.
10. The GND trace between the output capacitor and the GND pin should be as wide as possible to minimize its trace impedance.

10.2 Layout Example

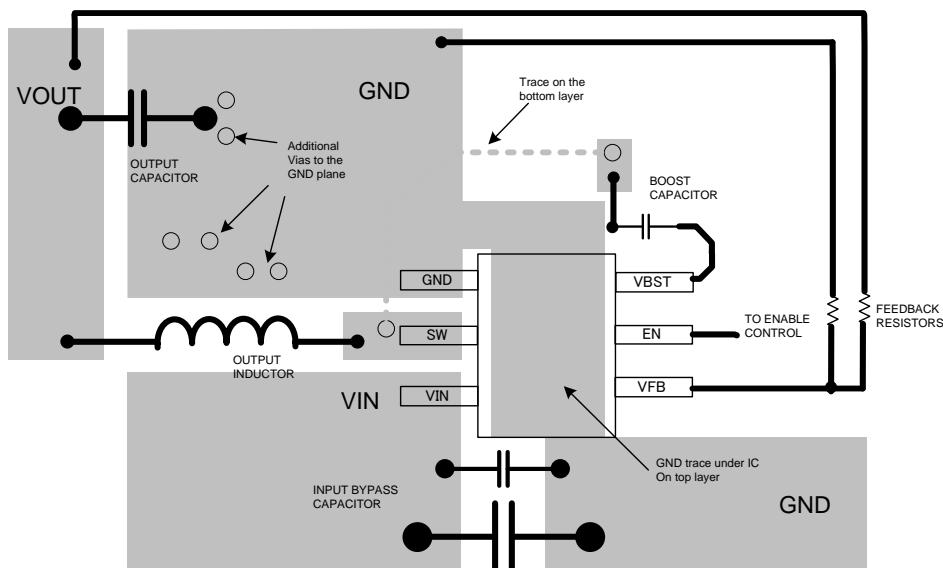


Figure 30. TPS565208 Layout Example

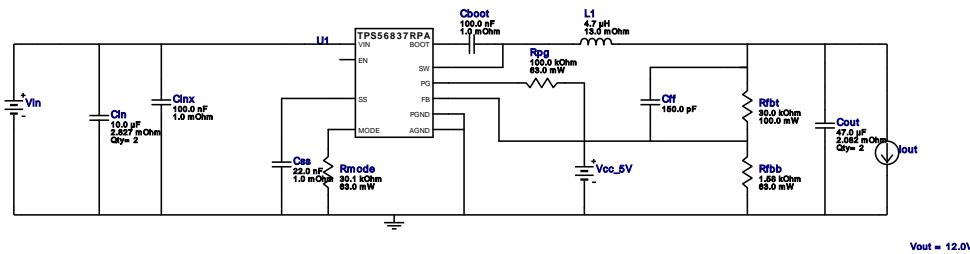
9.9 12 V-Schaltregler Simulation

Im folgenden Abschnitt wird die gesamte Simulation des 12 V-Schaltreglers gezeigt. Die Simulation wurde mit dem Programm TI Power Systems durchgeführt.



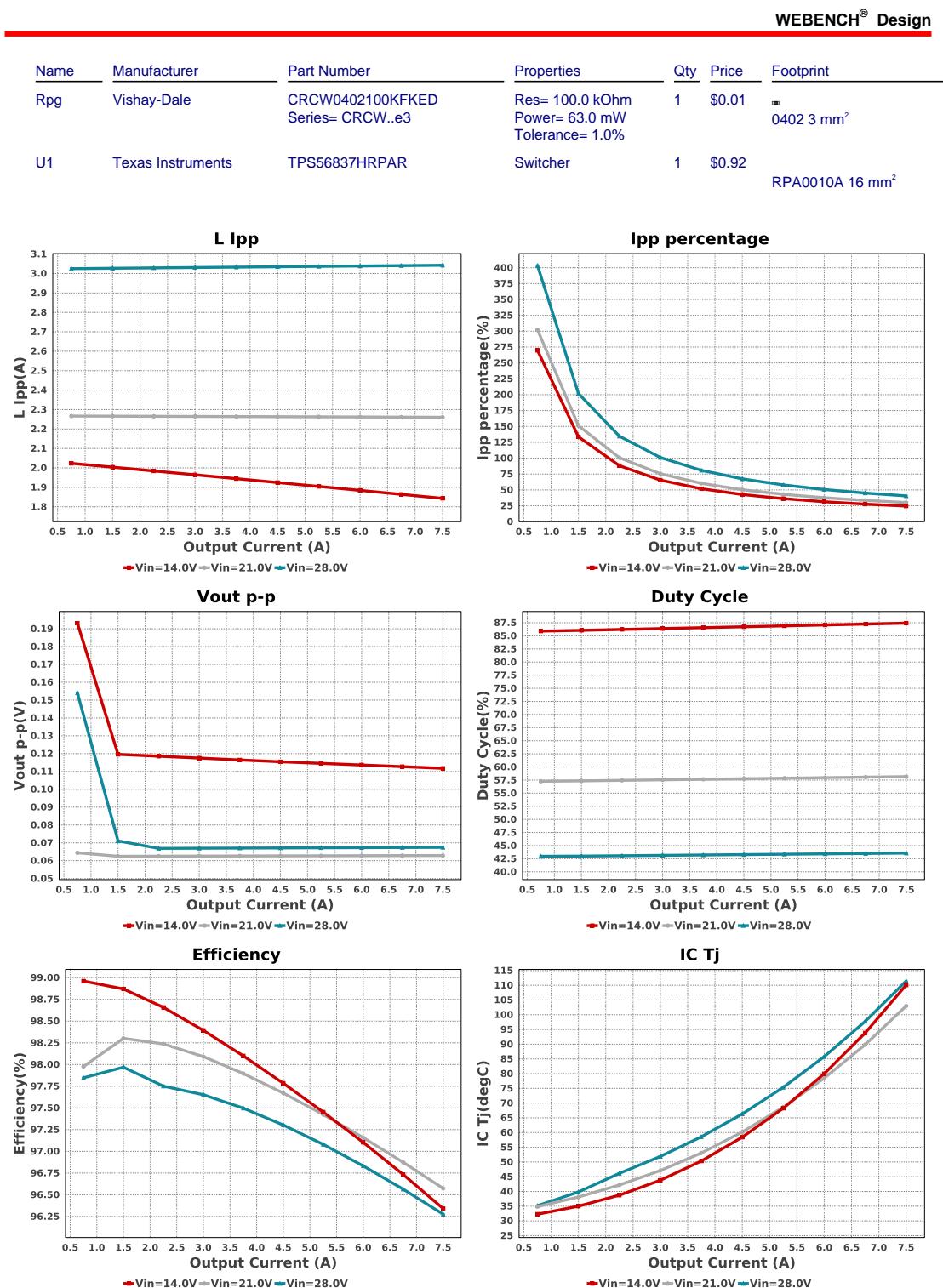
VinMin = 14.0V
 VinMax = 28.0V
 Vout = 12.0V
 Iout = 7.5A
 Device = TPS56837HRPAR
 Topology = Buck
 Created = 2025-02-15 11:35:53.686
 BOM Cost = \$3.54
 BOM Count = 14
 Total Pd = 3.48W

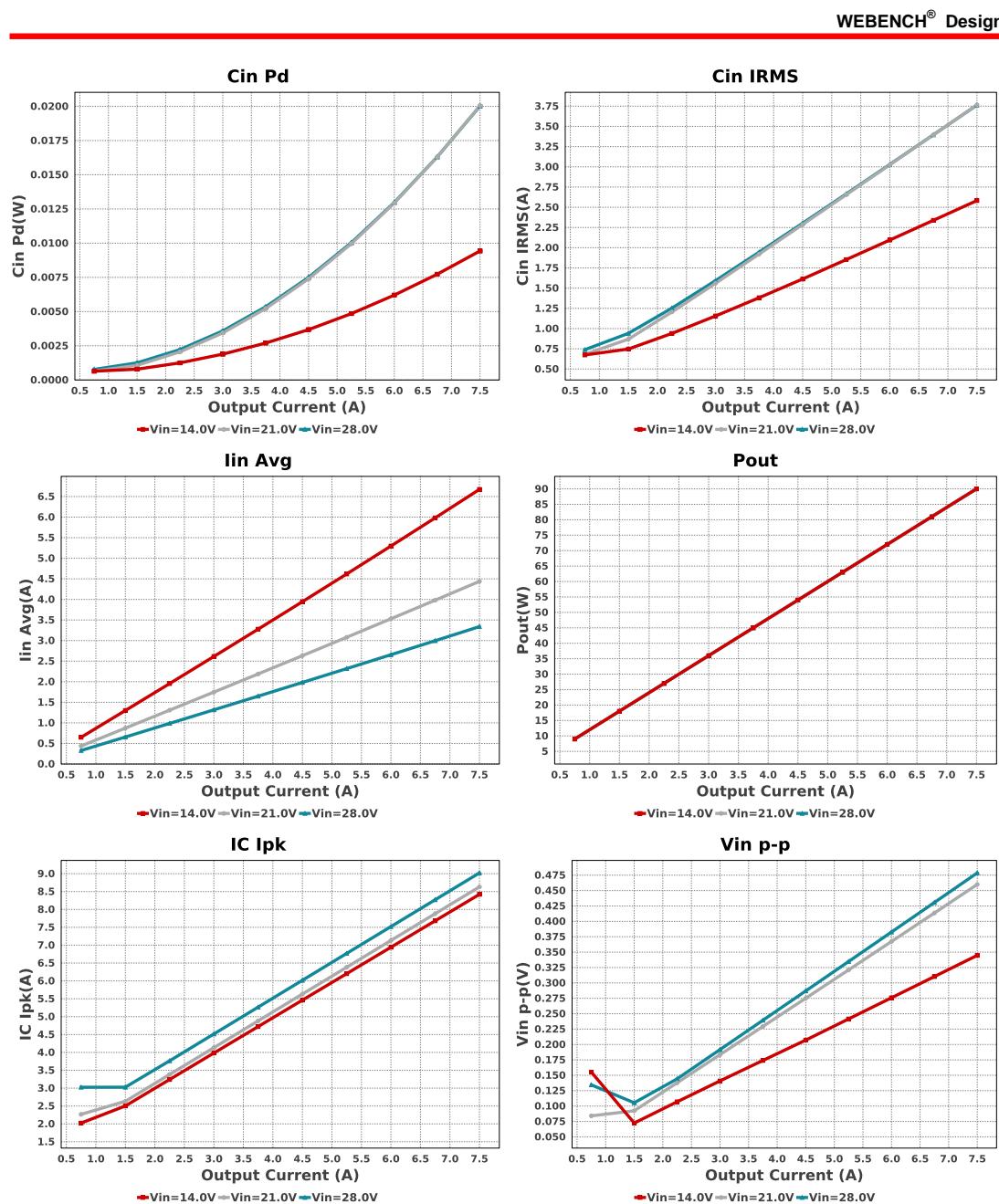
Design : 19 TPS56837HRPAR
 TPS56837HRPAR 14V-28V to 12.00V @ 7.5A

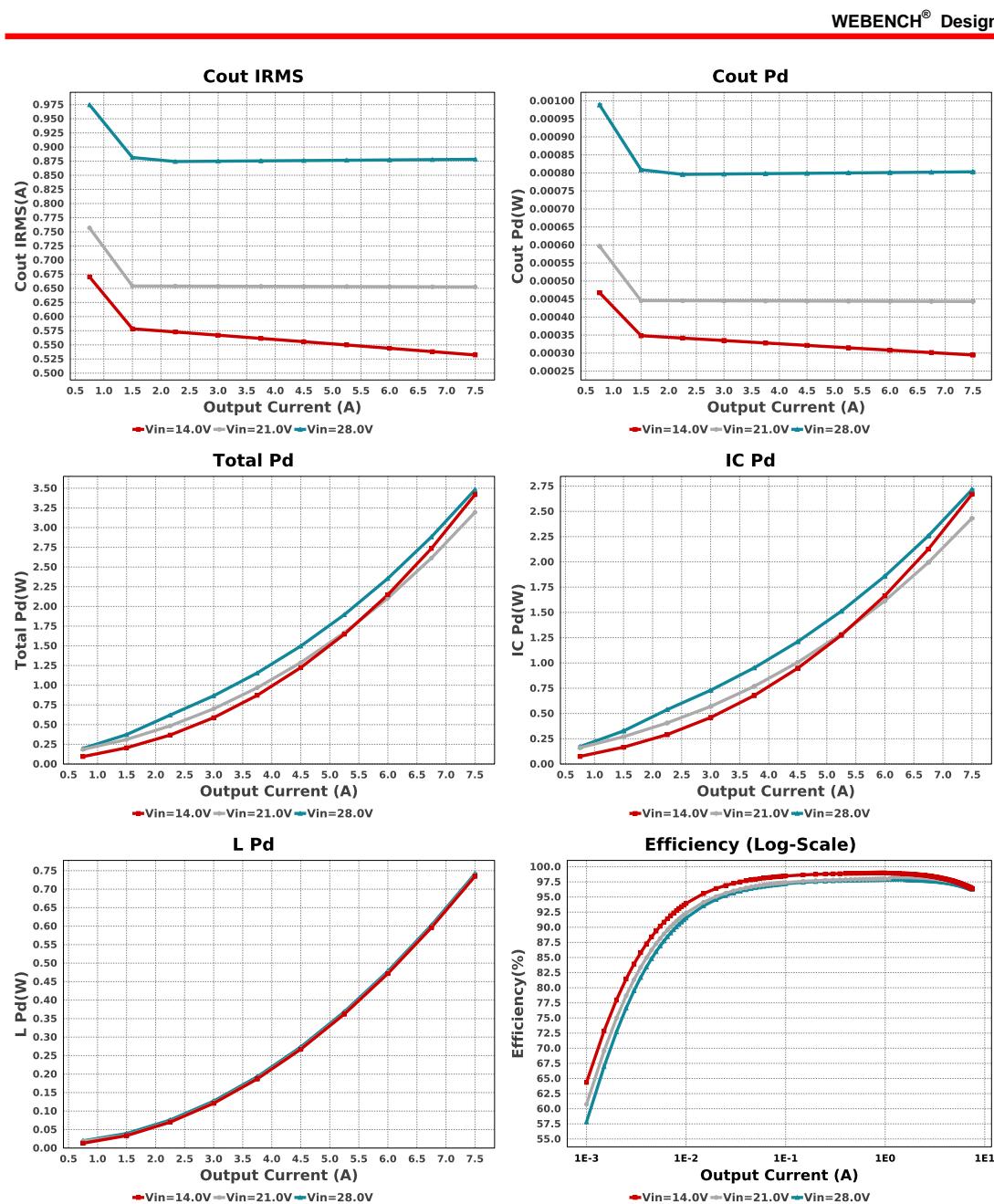


Electrical BOM

Name	Manufacturer	Part Number	Properties	Qty	Price	Footprint
Cboot	MuRata	GRM155R71C104KA88D Series= X7R	Cap= 100.0 nF ESR= 1.0 mOhm VDC= 16.0 V IRMS= 0.0 A	1	\$0.01	0402 3 mm ²
Cff	Kemet	C0603C151K3GACTU Series= C0G/NP0	Cap= 150.0 pF VDC= 25.0 V IRMS= 0.0 A	1	\$0.02	0603 5 mm ²
Cin	TDK	CGA6P3X7S1H106K250AB Series= X7S	Cap= 10.0 uF ESR= 2.827 mOhm VDC= 50.0 V IRMS= 4.3729 A	2	\$0.31	1210_280 15 mm ²
Cinx	MuRata	GRM21BR71H104KA01L Series= X7R	Cap= 100.0 nF ESR= 1.0 mOhm VDC= 50.0 V IRMS= 3.85 A	1	\$0.03	0805 7 mm ²
Cout	TDK	C3216X5R1E476M160AC Series= X5R	Cap= 47.0 uF ESR= 2.082 mOhm VDC= 25.0 V IRMS= 5.0279 A	2	\$0.35	1206 11 mm ²
Css	MuRata	GRM155R71C223KA01D Series= X7R	Cap= 22.0 nF ESR= 1.0 mOhm VDC= 16.0 V IRMS= 0.0 A	1	\$0.01	0402 3 mm ²
L1	Coilcraft	XAL7070-472MEB	L= 4.7 μH 13.0 mOhm	1	\$1.19	XAL7070 87 mm ²
Rfbb	Vishay-Dale	CRCW04021K58FKED Series= CRCW..e3	Res= 1.58 kOhm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	0402 3 mm ²
Rfbt	Yageo	RC0603FR-0730KL Series= ?	Res= 30.0 kOhm Power= 100.0 mW Tolerance= 1.0%	1	\$0.01	0603 5 mm ²
Rmode	Vishay-Dale	CRCW040230K1FKED Series= CRCW..e3	Res= 30.1 kOhm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	0402 3 mm ²







Operating Values

#	Name	Value	Category	Description
1.	Cin IRMS	3.764 A	Capacitor	Input capacitor RMS ripple current
2.	Cin Pd	20.024 mW	Capacitor	Input capacitor power dissipation
3.	Cout IRMS	878.279 mA	Capacitor	Output capacitor RMS ripple current
4.	Cout Pd	803.0 μ W	Capacitor	Output capacitor power dissipation
5.	IC Ipk	9.021 A	IC	Peak switch current in IC
6.	IC Pd	2.714 W	IC	IC power dissipation
7.	IC Tj	111.426 degC	IC	IC junction temperature
8.	IC Tolerance	6.0 mV	IC	IC Feedback Tolerance
9.	ICThetaJA	30.0 degC/W	IC	IC junction-to-ambient thermal resistance
10.	lin Avg	3.339 A	IC	Average input current

WEBENCH® Design

#	Name	Value	Category	Description
11.	Ipp percentage	40.566 %	Inductor	Inductor ripple current percentage (with respect to average inductor current)
12.	L Ipp	3.042 A	Inductor	Peak-to-peak inductor ripple current
13.	L Pd	741.28 mW	Inductor	Inductor power dissipation
14.	Cin Pd	20.024 mW	Power	Input capacitor power dissipation
15.	Cout Pd	803.0 μ W	Power	Output capacitor power dissipation
16.	IC Pd	2.714 W	Power	IC power dissipation
17.	L Pd	741.28 mW	Power	Inductor power dissipation
18.	Total Pd	3.481 W	Power	Total Power Dissipation
19.	BOM Count	14	System Information	Total Design BOM count
20.	Duty Cycle	43.569 %	System Information	Duty cycle
21.	Efficiency	96.276 %	System Information	Steady state efficiency
22.	FootPrint	186.0 mm ²	System Information	Total Foot Print Area of BOM components
23.	Frequency	479.865 kHz	System Information	Switching frequency
24.	Iout	7.5 A	System Information	Iout operating point
25.	Mode	CCM	System Information	Conduction Mode
26.	Pout	90.0 W	System Information	Total output power
27.	Total BOM	\$3.54	System Information	Total BOM Cost
28.	Vin	28.0 V	System Information	Vin operating point
29.	Vin p-p	478.654 mV	System Information	Peak-to-peak input voltage
30.	Vout	12.0 V	System Information	Operational Output Voltage
31.	Vout Actual	11.992 V	System Information	Vout Actual calculated based on selected voltage divider resistors
32.	Vout Tolerance	2.938 %	System Information	Vout Tolerance based on IC Tolerance (no load) and voltage divider resistors if applicable
33.	Vout p-p	67.441 mV	System Information	Peak-to-peak output ripple voltage

Design Inputs

Name	Value	Description
Iout	7.5	Maximum Output Current
VinMax	28.0	Maximum input voltage
VinMin	14.0	Minimum input voltage
Vout	12.0	Output Voltage
base_pn	TPS56837H	Base Product Number
source	DC	Input Source Type
Ta	30.0	Ambient temperature

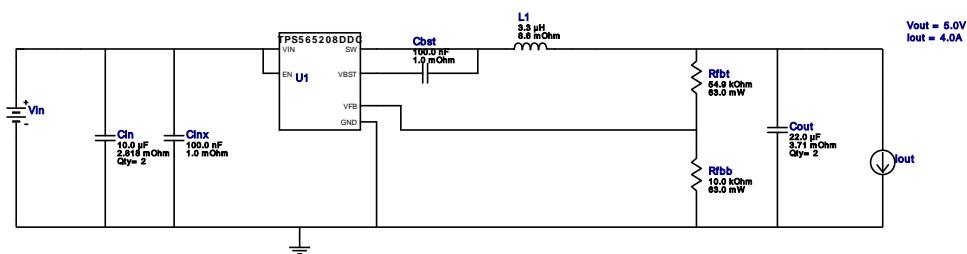
9.10 5 V-Schaltregler Simulation

Im folgenden Abschnitt wird die gesamte Simulation des 5 V-Schaltreglers gezeigt. Die Simulation wurde mit dem Programm TI Power Systems durchgeführt.



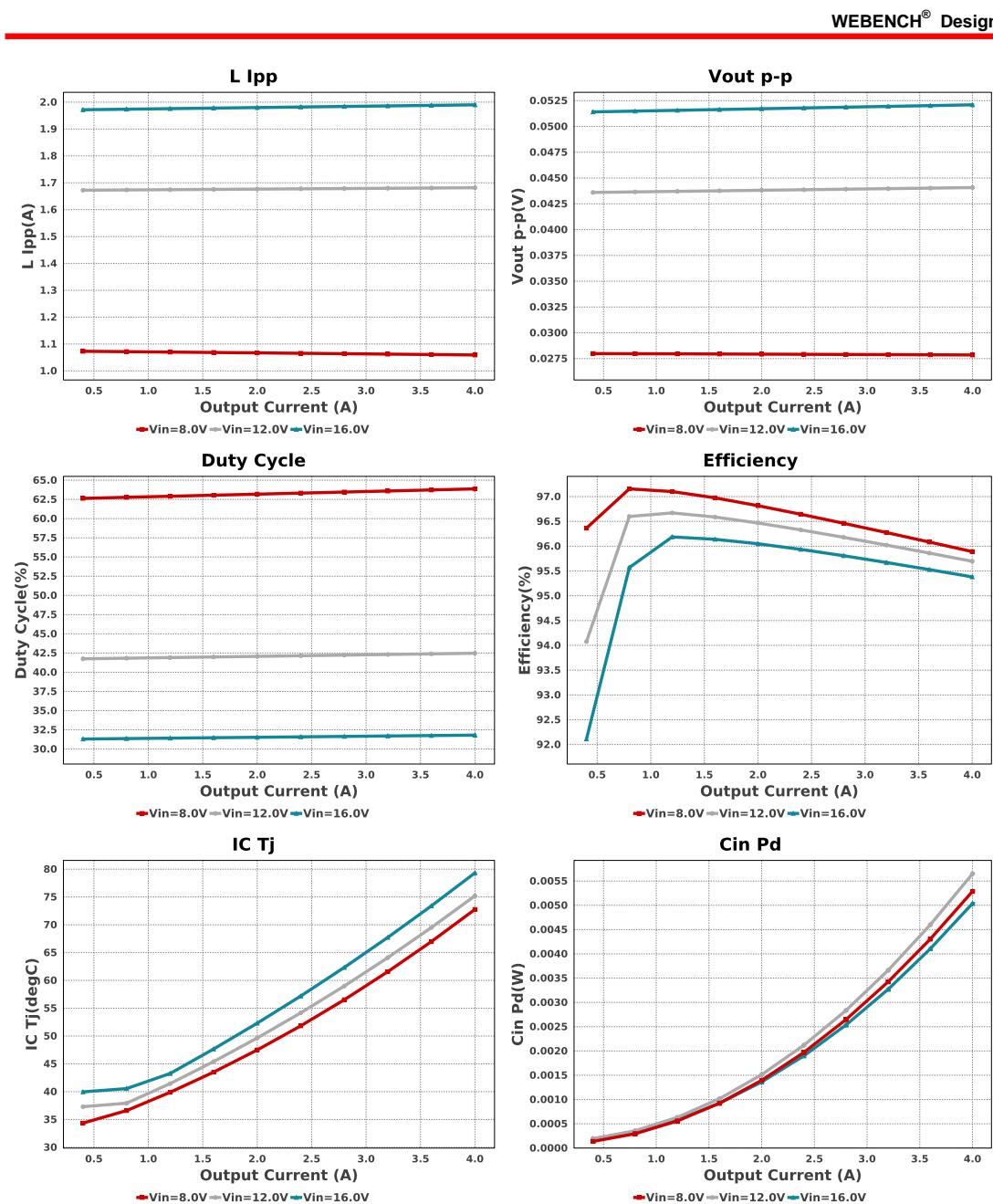
VinMin = 8.0V
 VinMax = 16.0V
 Vout = 5.0V
 Iout = 4.0A
 Device = TPS565208DDCR
 Topology = Buck
 Created = 2025-02-15 11:29:17.246
 BOM Cost = \$2.05
 BOM Count = 10
 Total Pd = 0.97W

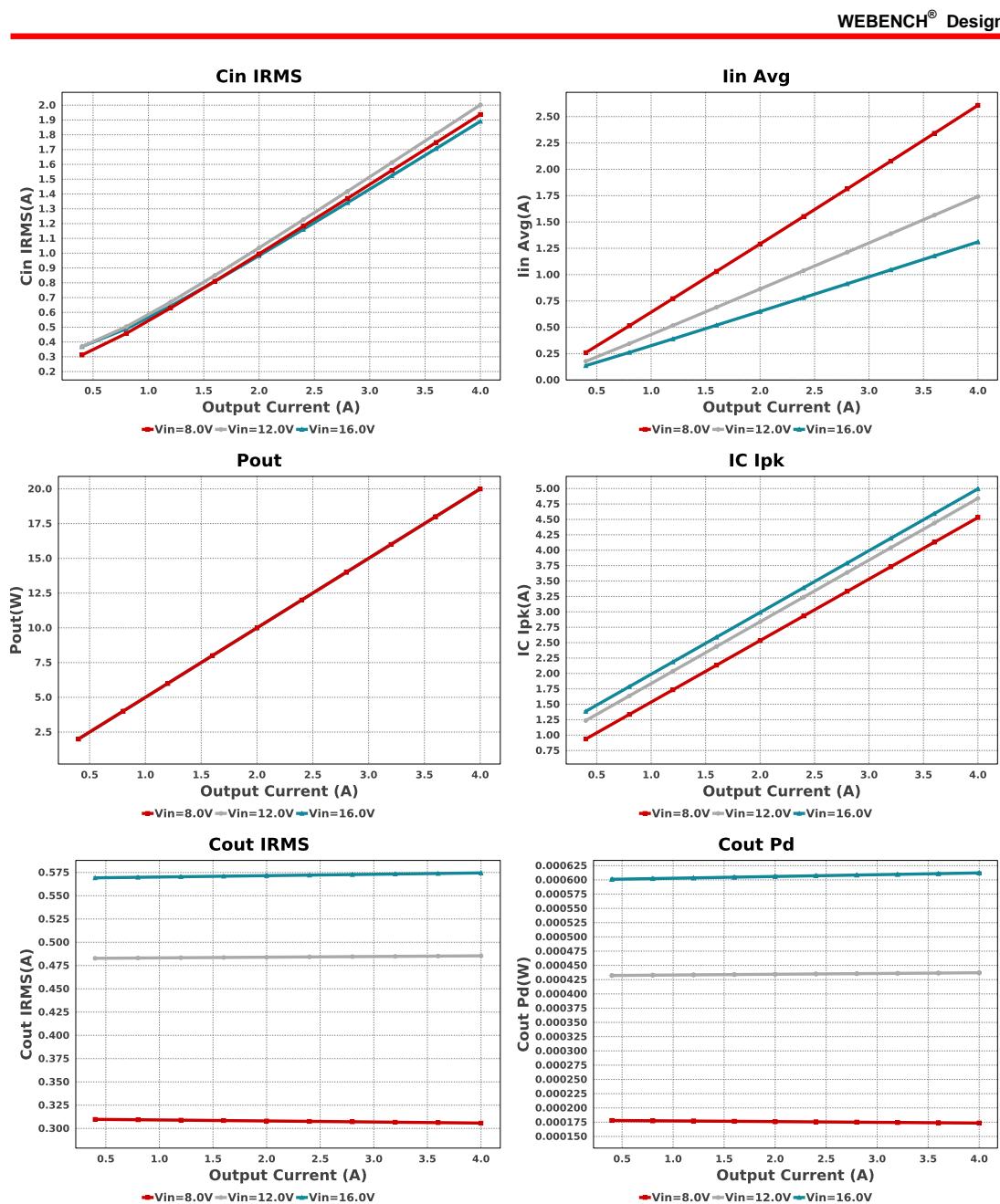
Design : 18 TPS565208DDCR
 TPS565208DDCR 8V-16V to 5.00V @ 4A

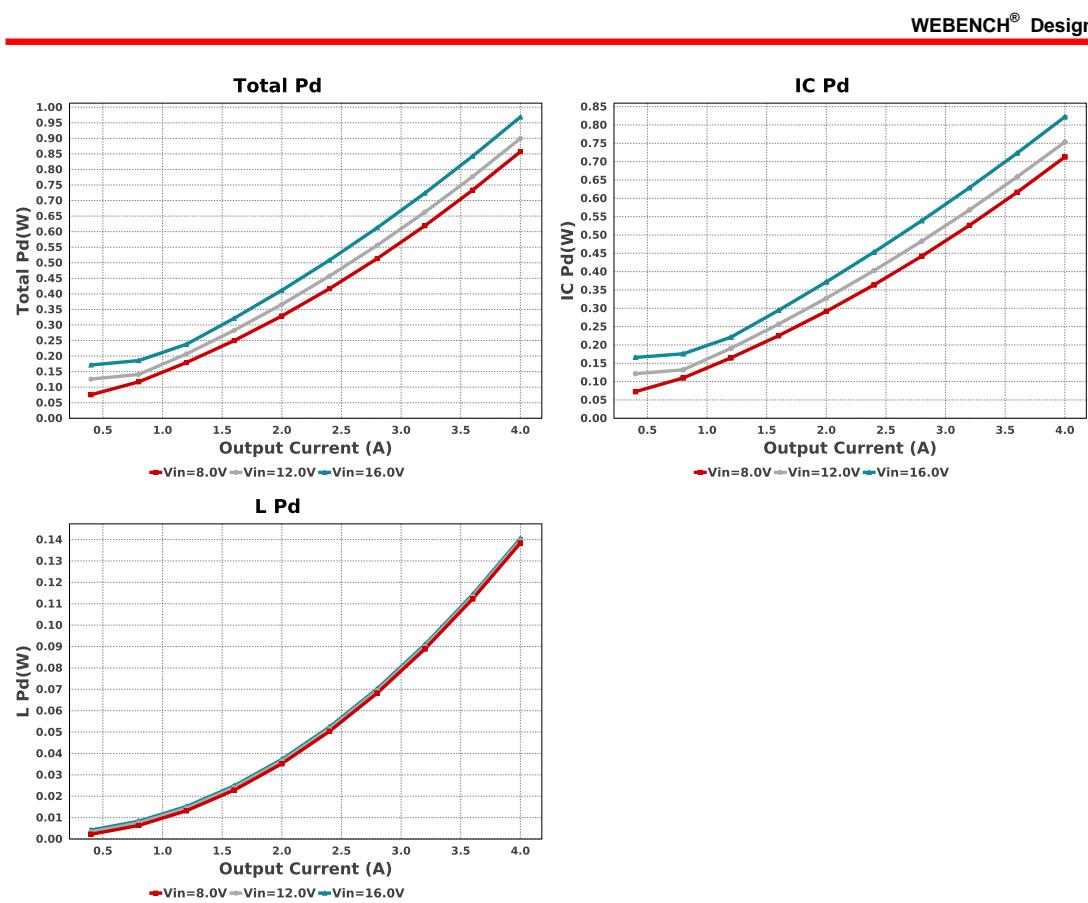


Electrical BOM

Name	Manufacturer	Part Number	Properties	Qty	Price	Footprint
Cbst	Kemet	C0603C104M3VACTU Series= Y5V	Cap= 100.0 nF ESR= 1.0 mOhm VDC= 25.0 V IRMS= 0.0 A	1	\$0.01	0603 5 mm ²
Cin	TDK	C2012X5R1V106K085AC Series= X5R	Cap= 10.0 uF ESR= 2.818 mOhm VDC= 35.0 V IRMS= 3.8868 A	2	\$0.12	0805 7 mm ²
Cinx	MuRata	GRM155R71C104KA88D Series= X7R	Cap= 100.0 nF ESR= 1.0 mOhm VDC= 16.0 V IRMS= 0.0 A	1	\$0.01	0402 3 mm ²
Cout	TDK	C1608X5R1A226M080AC Series= X5R	Cap= 22.0 uF ESR= 3.71 mOhm VDC= 10.0 V IRMS= 2.69936 A	2	\$0.08	0603 5 mm ²
L1	Coilcraft	XAL7070-332MEB	L= 3.3 uH 8.6 mOhm	1	\$1.19	XAL7070 87 mm ²
Rfb	Vishay-Dale	CRCW040210K0FKED Series= CRCW..e3	Res= 10.0 kOhm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	0402 3 mm ²
Rfbt	Vishay-Dale	CRCW040254K9FKED Series= CRCW..e3	Res= 54.9 kOhm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	0402 3 mm ²
U1	Texas Instruments	TPS565208DDCR	Switcher	1	\$0.42	DDC0006A_N 10 mm ²







Operating Values

#	Name	Value	Category	Description
1.	Cin IRMS	1.891 A	Capacitor	Input capacitor RMS ripple current
2.	Cin Pd	5.038 mW	Capacitor	Input capacitor power dissipation
3.	Cout IRMS	574.483 mA	Capacitor	Output capacitor RMS ripple current
4.	Cout Pd	612.21 μ W	Capacitor	Output capacitor power dissipation
5.	IC Ipk	4.995 A	IC	Peak switch current in IC
6.	IC Pd	822.08 mW	IC	IC power dissipation
7.	IC Tj	79.325 degC	IC	IC junction temperature
8.	IC Tolerance	10.0 mV	IC	IC Feedback Tolerance
9.	ICThetaJA	60.0 degC/W	IC	IC junction-to-ambient thermal resistance
10.	Iin Avg	1.31 A	IC	Average input current
11.	L Ipp	1.99 A	Inductor	Peak-to-peak inductor ripple current
12.	L Pd	140.44 mW	Inductor	Inductor power dissipation
13.	Cin Pd	5.038 mW	Power	Input capacitor power dissipation
14.	Cout Pd	612.21 μ W	Power	Output capacitor power dissipation
15.	IC Pd	822.08 mW	Power	IC power dissipation
16.	L Pd	140.44 mW	Power	Inductor power dissipation
17.	Total Pd	968.512 mW	Power	Total Power Dissipation
18.	BOM Count	10	System Information	Total Design BOM count
19.	Duty Cycle	31.813 %	System Information	Duty cycle
20.	Efficiency	95.381 %	System Information	Steady state efficiency
21.	FootPrint	134.0 mm ²	System Information	Total Foot Print Area of BOM components
22.	Frequency	526.493 kHz	System Information	Switching frequency
23.	Iout	4.0 A	System Information	Iout operating point
24.	Mode	CCM	System Information	Conduction Mode

WEBENCH® Design

#	Name	Value	Category	Description
25.	Pout	20.0 W	System Information	Total output power
26.	Total BOM	\$2.05	System Information	Total BOM Cost
27.	Vin	16.0 V	System Information	Vin operating point
28.	Vout	5.0 V	System Information	Operational Output Voltage
29.	Vout Actual	4.965 V	System Information	Vout Actual calculated based on selected voltage divider resistors
30.	Vout Tolerance	3.038 %	System Information	Vout Tolerance based on IC Tolerance (no load) and voltage divider resistors if applicable
31.	Vout p-p	52.096 mV	System Information	Peak-to-peak output ripple voltage

Design Inputs

Name	Value	Description
Iout	4.0	Maximum Output Current
VinMax	16.0	Maximum input voltage
VinMin	8.0	Minimum input voltage
Vout	5.0	Output Voltage
base_pn	TPS565208	Base Product Number
source	DC	Input Source Type
Ta	30.0	Ambient temperature

9.11 Bildschirmaufnahmen (Slicing-Software)

Im Folgenden sind Bildschirmaufnahmen der Slicing-Software dargestellt, die die Benutzeroberfläche und verfügbaren Funktionen zeigen.

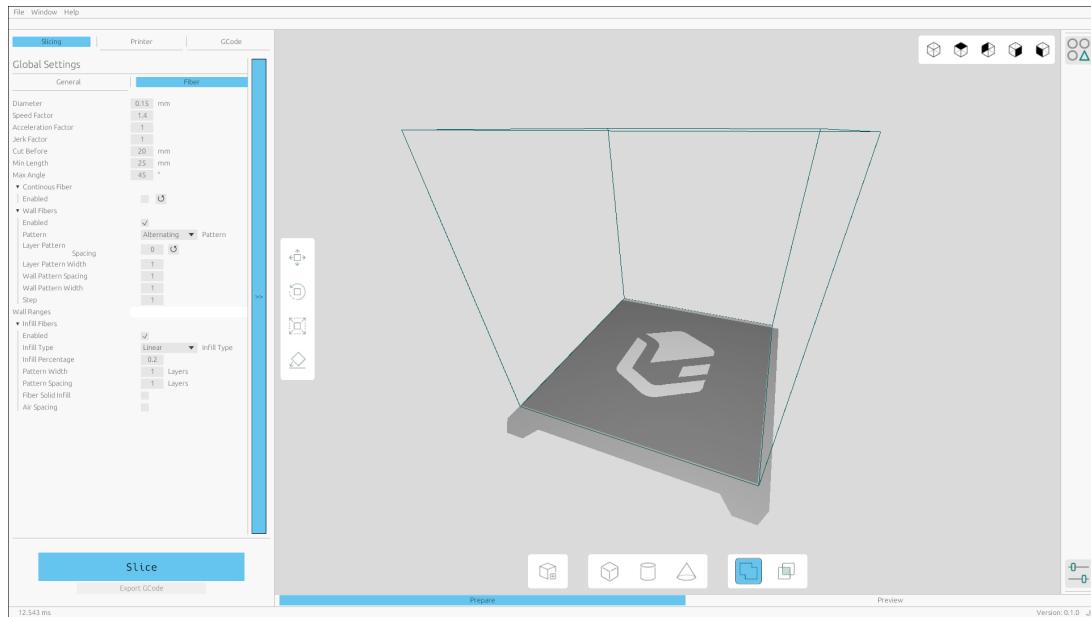


Abb. 152: Applikation Überblick

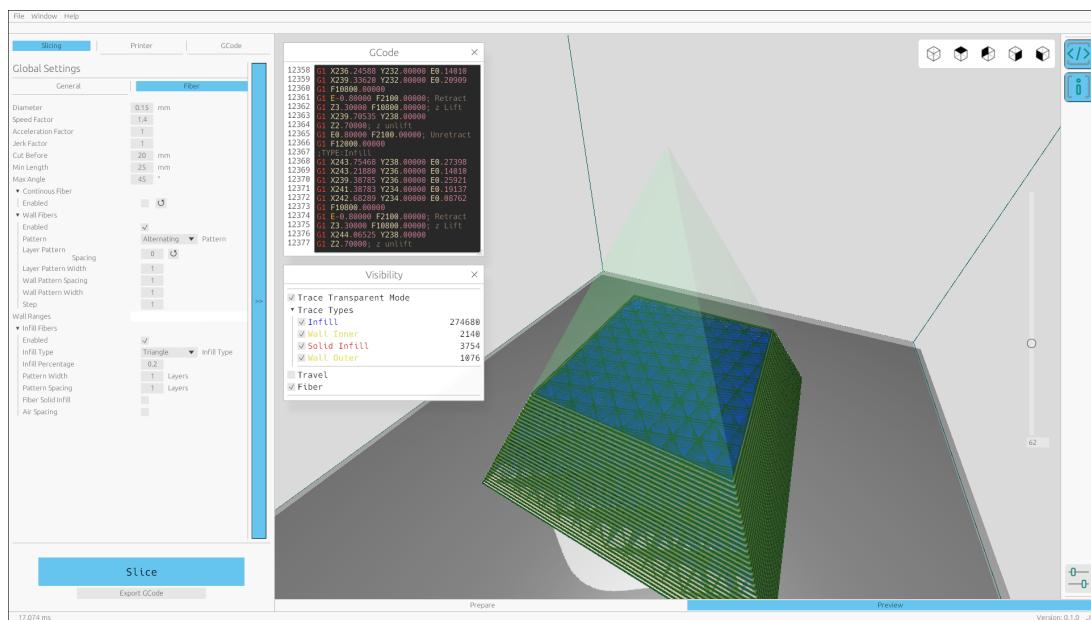


Abb. 153: Applikation Überblick mit Fasevisualisierung

10 Literaturverzeichnis

- 3DJake. (2019). *BLTouch Nivellierungssensor* [[Online; accessed 28. Jan. 2025]]. <https://www.3djake.at/antclabs/bltouch-nivellierungssensor>
- 3Dnatives. (2024, November). *Kurzfasern vs. Endlosfasern: Welche Verstärkung für 3D-Druck mit Verbundwerkstoffen?* [[Online; accessed 08. März 2025]]. <https://www.3dnatives.com/de/kurzfasern-vs-endlosfasern-fuer-verbundwerkstoffe-121120241/>
- Alex. (2021). *Verpolungsschutz Schaltungen* [[Online; accessed 9. Aug. 2024]]. <https://www.aeq-web.com/verpolungsschutz-verschiedene-schaltungen-und-moeglichkeiten>
- AliExpress. (2025). *Kevlar-Aramid-Nähgarnseil mit hoher Temperatur- und Zugfestigkeit* [[Online; accessed 8. März 2025]]. <https://de.aliexpress.com/item/1005004530366533.html>
- All3DP. (2018). *Netzteile für 3D-Drucker – Auswahl der richtigen Leistung* [[Online; accessed 28. Feb. 2025]]. <https://all3dp.com/2/3d-printer-power-supply-how-to-choose-the-right-one>
- Anisoprint. (2025). *Anisoprint Composite Fiber Co-Extrusion*. <https://anisoprint.com/>
- AZ-Delivery. (2025a). *DRV8825 Schrittmotor-Treiber Modul mit Kühlkörper, z. B. für RAMPS 1.4, CNC Shield, 3D-Drucker, Prusa Mendel* [[Online; accessed 5. Jan. 2025]]. <https://www.az-delivery.de/en/products/drve8825-schrittmotor-treiber-modul-mit-kuhlkorper?variant=37100118866>
- AZ-Delivery. (2025b). *GY-521 MPU-6050 3-Achsen-Gyroskop und Beschleunigungssensor* [[Online; accessed 4. Jan. 2025]]. <https://www.az-delivery.de/en/products/gy-521-6-achsen-gyroskop-undbeschleunigungssensor>
- AZ-Delivery. (2025c). *MG90S Micro-Servo kompatibel mit Arduino* [[Online; accessed 8. Feb. 2025]]. <https://www.az-delivery.de/en/products/mg90s-micro-servomotor?variant=32344287150176>
- Developers, T. R. P. (2013). *Rust by Example*. <https://doc.rust-lang.org/rust-by-example/> [Online; zugegriffen am 01.08.2023].
- Developers, T. R. P. (2014a). *Crates.io*. <https://crates.io/> [Online; zugegriffen am 01.08.2023].
- Developers, T. R. P. (2014b). *The Rust Programming Language*. <https://doc.rust-lang.org/book/> [Online; zugegriffen am 01.08.2023].
- Developers, T. R. P. (2015). *Rust Reference*. <https://doc.rust-lang.org/reference/> [Online; zugegriffen am 01.08.2023].
- DigiKey. (2025a). *50-00619 - Tensility International Corp Power Connector* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.at/en/products/detail/tensility-international-corp/50-00619/10261135>
- DigiKey. (2025b). *665002113322 - Würth Elektronik Rechteckiger Steckverbinder-Gehäuse* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.at/en/products/detail/w%BCrth-elektronik/665002113322/5047510>
- DigiKey. (2025c). *665102131822 - Würth Elektronik Stiftleiste* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.at/en/products/detail/wurth-electronics-inc/665102131822/5047561>
- DigiKey. (2025d). *824521241 - Würth Elektronik TVS-Diode* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.de/en/products/detail/w%BCrth-elektronik/824521241/5870552>
- DigiKey. (2025e). *971070154 - Würth Elektronik Platinenabstandshalter* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.de/en/products/detail/w%BCrth-elektronik/971070154/9488609>
- DigiKey. (2025f). *971110154 - Würth Elektronik Platinenabstandshalter* [[Online; accessed 28. Jan. 2025]]. <https://www.digikey.de/en/products/detail/w%BCrth-elektronik/971110154/9488627>
- DigiKey. (2025g). *A-USB31C-20A-150A - Assmann WSW Components USB-Kabel* [[Online; accessed 28. Jan. 2025]]. <https://www.digikey.de/en/products/detail/assmann-wsw-components/A-USB31C-20A-150A/10408414>
- DigiKey. (2025h). *ATMEGA328P-AU - Microchip Technology* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.de/en/products/detail/microchip-technology/ATMEGA328P-AU/1832260>
- DigiKey. (2025i). *CF881-15-02 - Igus Multileiter-Kabel* [[Online; accessed 28. Jan. 2025]]. <https://www.digikey.de/en/products/detail/igus/CF881-15-02/21280642>
- DigiKey. (2025j). *CFM-2507CF-0140-313 - Same Sky (ehemals CUI Devices) DC-Brushless-Lüfter* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.at/en/products/detail/same-sky-formerly-cui-devices/CFM-2507CF-0140-313/19524735>
- DigiKey. (2025k). *CFM-2507CF-1140-313 - Same Sky (ehemals CUI Devices) DC-Brushless-Lüfter* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.at/en/products/detail/same-sky-formerly-cui-devices/CFM-2507CF-1140-313/19524850>
- DigiKey. (2025l). *Datenblatt: B57550G1103F005 – NTC-Thermistor* [[Online; accessed 1. Mar. 2025]]. <https://www.digikey.at/de/products/detail/epcos-tdk-electronics/B57550G1103F005/3500387>

- DigiKey. (2025m). *ECS-160-18-4X-CKM Quarz - ECS Inc.* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.at/de/products/detail/ecs-inc/ECS-160-18-4X-CKM/12349451>
- DigiKey. (2025n). *FT232RNL-TUBE - USB-zu-Seriell Wandler von FTDI* [[Online; accessed 28. Jan. 2025]]. <https://www.digikey.at/en/products/detail/ftdi-future-technology-devices-international-ltd/FT232RNL-TUBE/16836164>
- DigiKey. (2025o). *FTSH-103-01-L-DH - Samtec Inc. Stifteleiste* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.at/en/products/detail/samtec-inc/FTSH-103-01-L-DH/1110175>
- DigiKey. (2025p). *HLE-107-02-L-DV - Samtec Inc. Buchsenleiste* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.de/en/products/detail/samtec-inc/HLE-107-02-L-DV/6693078>
- DigiKey. (2025q). *IHLP2525CZER3R3M11 - Vishay Dale Festinduktivität* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.at/en/products/detail/vishay-dale/IHLP2525CZER3R3M11/5419141>
- DigiKey. (2025r). *M50-1900005 - Harwin Inc. Jumper/Shunt* [[Online; accessed 28. Jan. 2025]]. <https://www.digikey.at/en/products/detail/harwin-inc/M50-1900005/2890862>
- DigiKey. (2025s). *PJ-082BH - Same Sky (ehemals CUI Devices) Power Connector* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.de/en/products/detail/same-sky-formerly-cui-devices/PJ-082BH/3477156>
- DigiKey. (2025t). *PTS647SM38SMTR2 LFS - C&K Tactile Schalter* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.at/en/products/detail/c-k/PTS647SM38SMTR2-LFS/9649861>
- DigiKey. (2025u). *RJK0456DPB-00#J5 - Renesas Electronics Corporation MOSFET* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.de/en/products/detail/renesas-electronics-corporation/RJK0456DPB-00-J5/2694914>
- DigiKey. (2025v). *S2B-XH-A-1 - JST Sales America Inc. Stifteleiste* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.at/en/products/detail/jst-sales-america-inc/S2B-XH-A-1/9961922>
- DigiKey. (2025w). *S3B-XH-A-1 - JST Sales America Inc. Stifteleiste* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.at/en/products/detail/jst-sales-america-inc/S3B-XH-A-1/1556255?s=N4IgTCBcDaIMoGYBCBaAGgCRQQRQRhAF0BfIA>
- DigiKey. (2025x). *S4B-XH-A-1 - JST Sales America Inc. Stifteleiste* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.at/en/products/detail/jst-sales-america-inc/S4B-XH-A-1/9961923>
- DigiKey. (2025y). *SPM10040T-4R7M-HZ - TDK Corporation Festinduktivität* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.at/en/products/detail/tdk-corporation/SPM10040T-4R7M-HZ/8637498>
- DigiKey. (2025z). *SQJ409EP-T1-GE3 - Vishay Siliconix MOSFET* [[Online; accessed 28. Jan. 2025]]. <https://www.digikey.at/de/products/detail/vishay-siliconix/SQJ409EP-T1-GE3/7326286>
- DigiKey. (2025aa). *SZ1SMB5927BT3G - onsemi Zenerdiode* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.de/en/products/detail/onsemi/SZ1SMB5927BT3G/9960092>
- DigiKey. (2025ab). *TPM16050-S6TR - 3PEAK Motor Driver Controller* [[Online; accessed 5. Jan. 2025]]. <https://www.digikey.at/en/products/detail/3peak/TPM16050-S6TR/22229143>
- DigiKey. (2025ac). *TPS565208DDCR - Texas Instruments DC-DC-Spannungsregler* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.at/en/products/detail/texas-instruments/TPS565208DDCR/7776393>
- DigiKey. (2025ad). *TPS56837HRPAR - Texas Instruments DC-DC-Spannungsregler* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.at/de/products/detail/texas-instruments/TPS56837HRPAR/22188306>
- DigiKey. (2025ae). *TSM-107-04-L-DV - Samtec Inc. Stifteleiste* [[Online; accessed 4. Jan. 2025]]. <https://www.digikey.de/en/products/detail/samtec-inc/TSM-107-04-L-DV/7638934>
- DigiKey. (2025af). *USB4085-GF-A - GCT USB-Steckverbinder* [[Online; accessed 8. Feb. 2025]]. <https://www.digikey.at/en/products/detail/gct/USB4085-GF-A/9859662>
- DigiKey. (2025). *Raspberry Pi 3 Model B+ – Technische Spezifikationen und Verfügbarkeit* [[Online; accessed 1. Mar. 2025]]. <https://www.digikey.de/de/products/detail/raspberry-pi/SC0073/8571724>
- EB-TEC. (2025). *EB-TEC industrielle Faserverstärkung*. <https://eb-tec.de/>
- Electronics, M. (2025). *HW-07-08-L-D-250-SM - Samtec Board-to-Board Steckverbinder* [[Online; accessed 5. Jan. 2025]]. <https://www.mouser.de/ProductDetail/Samtec/HW-07-08-L-D-250-SM?qs=3%252BjIH0OdpA8a62QJxWcTNw%3D%3D>
- emilk. (2020). *equi*. <https://github.com/emilk/egui> [Online; zugegriffen am 01.08.2023].
- et al., L. C. (2020). A review of FDM 3D printing in composite materials. *Composite Science and Technology*, 194.
- et al., S. L. (2012). Enhancing mechanical properties of FDM parts with fiber integration. *Journal of Manufacturing Science*, 45.

- Filament2Print. (2025). *Welche Versorgungsspannung wählen: 12 V oder 24 V?* [[Online; accessed 28. Feb. 2025]]. <https://filament2print.com/de/blog/qual-voltagem-elscolher-12v-ou-24v-gfx-rs>.
- gfx-rs. (2019). *wgpu*. <https://github.com/gfx-rs/wgpu> [Online; zugegriffen am 01.07.2024].
- Gibson, I., Rosen, D., & Stucker, B. (2015). *Additive Manufacturing Technologies*. Springer.
- Harald. (2025). *E3D Netzteil, 24V 16,5A 400W* [[Online; accessed 18. Jan 2025]]. <https://www.3djake.at/e3d/netzteil>
- Hart, C. (2022). *glam*. <https://github.com/bitshifter/glam-rs> [Online; zugegriffen am 01.07.2024].
- Ince, L. (2021). *GladiusSlicer*. <https://github.com/GladiusSlicer/GladiusSlicer> [Online; zugegriffen am 05.10.2024].
- Institut für Fertigungstechnik, TU Graz. (2025). *IFT - Additive Fertigung* [[Online; accessed 15. Jan. 2025]]. <https://www.tugraz.at/institute/ift/forschung/additive-fertigung>
- Jariwala, A., & Ding, R. (2019). Stereolithography: Current trends and future potential. *Materials Today*, 33.
- JLCPCB. (2025). *PCB Manufacturing & Assembly Capabilities* [[Online; accessed 3. Mar. 2025]]. <https://jlcpcb.com/capabilities/pcb-capabilities>
- Kruth, J. (2005). Selective laser sintering: Principles and applications. *CIRP Annals*, 54.
- Mark3D GmbH. (2022). *Wie funktioniert der Markforged Endlosfaser 3D-Druck?* [[Online; accessed 12. März 2025]]. YouTube. <https://www.youtube.com/watch?v=5CXaZY90VUs>
- Markforged. (2025). *Markforged 3D-Drucksysteme*. <https://markforged.com/de/3d-printers/mark-two>
- Mashayekhi, F., Bardon, J., Berthé, V., Perrin, H., Westermann, S., & Addiego, F. (2021). Fused Filament Fabrication of Polymers and Continuous Fiber-Reinforced Polymer Composites: Advances in Structure Optimization and Health Monitoring. *Polymers*. <https://www.mdpi.com/2073-4360/13/5/789>
- Matthew. (2021). Designing with USB-C: Lessons Learned [[Online; accessed 6. Apr. 2021]]. *Dubious Creations*. <https://dubiouscreations.com/2021/04/06/designing-with-usb-c-lessons-learned>
- Pieters, A. (2022). *DRV8825 - Pinout* - [[Online; accessed 1. Mar. 2024]]. <https://www.studiopieters.nl/driv8825-pinout>
- Prototec GmbH. (2025). *Faserverstärkte 3D-Druck Bauteile* [[Online; accessed 08. März 2025]]. <https://prototec.de/faserverstaerkte-3d-druck-bauteile>
- Schweizer-fn-Luft. (2025, Februar). Wärmeübergangskoeffizienten von Gasen [[Online; accessed 13. Feb. 2025]]. https://www.schweizer-fn.de/stoff/wuebergang_gase/wuebergang_gase.php
- Schweizer-fn-Wasser. (2025, Februar). Wärmeübergangskoeffizienten von Flüssigkeiten [[Online; accessed 27. Feb. 2025]]. https://www.schweizer-fn.de/stoff/wuebergang_fluessigkeit/wuebergang_fluessigkeit.php
- Sliceengineering. (2025, Februar). Mosquito® Hotend [[Online; accessed 19. Feb. 2025]]. <https://www.sliceengineering.com/products/the-mosquito-hotend>
- Source, O. (2015). *winit*. <https://github.com/rust-windowing/winit> [Online; zugegriffen am 01.08.2023].
- Source, O. (2016). *tokio*. <https://github.com/tokio-rs/tokio.git> [Online; zugegriffen am 01.10.2023].
- Stepperonline. (2024). *Nema 14 Bipolar 0.9Grad 5Ncm(7.08oz.in) 0.5A 35x35x20mm* [[Online; accessed 4. Jan. 2025]]. <https://www.omc-stepperonline.com/de/nema-14-bipolar-0-9deg-5ncm-7-08oz-in-0-5a-5v-35x35x20mm-4-draehete-14hm08-0504s>
- Systems, V. (2025). *Venox Systems V-Rex*. <https://venox.systems/produkte/v-rex>
- Tian, X. (2016). 3D printing with continuous fiber reinforcements. *Materials and Design*, 102.
- Wikipedia [[Online; accessed 13. Mar. 2025]]. (2024, September). https://en.wikipedia.org/w/index.php?title=Input_shaping&oldid=1248006299
- Wikipedia. (2025). *3D-Druck* [[Online; accessed 08. März 2025]]. <https://de.wikipedia.org/wiki/3D-Druck>

11 Abbildungsverzeichnis

1	Grafische Darstellung der Aufgabenverteilung innerhalb des Projekts	3
2	Autodesk Inventor	4
3	Fusion 360	4
4	SimScale	4
5	KiCad	4
6	Webench Power Designer	4
7	LTspice	4
8	WireViz	4
9	Visual Studio Code	5
10	GeoGebra	5
11	GitHub	5
12	MiniProtocol	5
13	YouTrack	5
14	L <small>A</small> T <small>E</small> X	5
15	yEd	5
16	Schematische Darstellung des FDM-Druckverfahrens (nach Institut für Fertigungstechnik, TU Graz, 2025)	6
17	Verschiedene Verfahren zur Faserintegration im 3D-Druck (nach Mashayekhi et al., 2021)	9
18	Darstellung der Fasereinbettung beim kontinuierlichen Faserverstärkungsverfahren CFR (nach Mark3D GmbH, 2022)	10
19	Die ausgewählten Kevlar-Fasern (nach AliExpress, 2025)	11
20	Ein Überblick des vollständigen CAD-Modells	12
21	Wärmeübergangskoeffizienttabelle (Schweizer-fn-Luft, 2025)	14
22	Version 1.1.0	16
23	Mosquito Hotend by SliceEngineering (Sliceengineering, 2025)	16
24	Thermische Simulation	17
25	Version 2.1.1	19
26	Thermische Simulation V2.0.0	20
27	Thermische Simulation V2.1.1	20
28	Version 3.1.1	23
29	Version 3.1.1, interne Struktur	23
30	Wärmeübergangskoeffizient Wasser (Schweizer-fn-Wasser, 2025)	24
31	Thermische Simulation V3.1.1	24
32	Version 4.1.1	26
33	SplitHotend	27
34	Insert	27
35	Halbschnitt, Version 4.1.1	28
36	Gesamtansicht des Druckkopfes, Version 5.5.0	30
37	Druckkopf, Version 5.5.0	31
38	Schneidmechanismus mit Bohrbuchse, Passstift und Gehäuse	31
39	SMN-Aktuator mit Elektromagnet	32
40	Insert mit ursprünglicher Verstrebung	32
41	Thermische Simulation V5.0.0 - Temperatur	33
42	Thermische Simulation V5.0.0 - Temperaturgradient	34
43	Thermische Simulation V5.3.0 - Temperatur	35
44	Thermische Simulation V5.3.0 - Temperaturgradient	36
45	Flusssimulation mit ungleichmäßiger Verteilung, 3D-Ansicht	37
46	Optimierte Flusssimulation mit gleichmäßiger Strömung, Draufsicht	37
47	Strömungssimulation Ansicht 1, V5.5.0	38
48	Luftentweichung	38
49	Strömungssimulation Ansicht 2, V5.5.0	38
50	Luftenabführung	38
51	Gesamtansicht des Druckkopfes, Version 6.0.1	41
52	Druckkopf, Version 6.0.1	42
53	Faserkanal	43
54	Montage des Druckkopfes mit Isolationsstiften	44

55	Schneidmechanismus der neuen Version	44
56	Halbschnitt des Schneidmechanismus	44
57	SMN-Aktor und Hebelmechanismus	45
58	Hardstops des SMN für exakte Positionierung	45
59	Optimierte Düse mit verlängertem Filamentkanal	46
60	Thermische Simulation der Isolationsstifte	47
61	Temperaturverteilung des SMN	48
62	Druckkopf in Version 7.0.0	50
63	Verbessertes Hotend	51
64	Hotend, Version 7.0.0	51
65	Hotend, Version 7.0.1	51
66	Coldend Version 6.0.1	52
67	Reduzierte Kühlrippen am Coldend in Version 7.0.1	52
68	Kanülenabschnitt als Faserführung im Hotend	52
69	Zusätzlicher Bauteilkühler, Version 7.1.0	53
70	Luftströmung im Druckkopf, Version 7.1.0	53
71	Thermische Simulation für das vereinfachte Coldend in Version 7.0.1	54
72	Darstellung der elektrischen Komponenten im Druckkopf	56
73	Systemübersicht der Elektronik und deren Verbindungen	57
74	Funktionale Modulaufteilung der Platine	64
75	Verbindung zwischen Leistungs- und Lüfter-Modul	64
76	Verbindung zwischen Leistungs- und Motoren-Modul	65
77	Netzteil eines 3D-Druckers (Harald, 2025)	65
78	Übersicht der Komponenten mit Strom- und Spannungsbedarf	66
79	Raspberry Pi 3 Model B+ (nach Digikey, 2025)	68
80	ATMEGA328P-AU Mikrocontroller aus (nach DigiKey, 2025h)	69
81	Schrittmotor 14HM08-0504S aus Stepperonline, 2024	70
82	DRV8825 Schrittmotor-Treiber aus (nach AZ-Delivery, 2025a)	71
83	Pinout des Schrittmotor-Treibers DRV8825 (nach Pieters, 2022)	71
84	Servo MG90S (nach AZ-Delivery, 2025c)	72
85	TPS56837HRPAR Spannungswandler aus (nach DigiKey, 2025ad)	73
86	Anwendungsschaltung für den Spannungswandler TPS56837HRPAR aus 9.7 <i>Layout Empfehlungen aus dem Datenblatt des TPS56387</i> , S.216	73
87	TPS565208DDCR Spannungswandler aus (nach DigiKey, 2025ac)	74
88	Anwendungsschaltung für den Spannungswandler TPS565208DDCR aus 9.8 <i>Layout Empfehlungen aus dem Datenblatt des TPS565208</i> , S.217	74
89	Schaltplan des Leistungseingangs und der Schutzschaltung	75
90	Verbindung zwischen 3D-Drucker-Netzteil und Platine	75
91	Verbindung zwischen Raspberry Pi und Platine über USB-C	76
92	Schaltplan des USB-C-Eingangs und des Schnittstellenwandlers	76
93	Schaltplan des Step-Down-Wandlers	77
94	Schaltplan des Step-Down-Wandlers von 12 V auf 5 V	78
95	Schaltplan der Lüftersteuerung mit PWM-Signalen vom ATmega328P	79
96	Schaltung der Simulation des Verpolungsschutzes mit PMOS-FET und Zener-Diode	81
97	Simulation der Eingangsspannung (grün) und Ausgangsspannung an der Last (blau)	81
98	Simulation der Eingangsspannung (grün) und Gate-Source-Spannung des PMOS-FETs (blau)	82
99	Duty Cycle in Abhängigkeit des Ausgangstroms. (aus 9.7 <i>Layout Empfehlungen aus dem Datenblatt des TPS56387</i> , S.216)	83
100	Effizienz in Abhängigkeit des Ausgangstroms. (aus 9.7 <i>Layout Empfehlungen aus dem Datenblatt des TPS56387</i> , S.216)	83
101	Junction-Temperatur des ICs in Abhängigkeit des Ausgangstroms. (aus 9.7 <i>Layout Empfehlungen aus dem Datenblatt des TPS56387</i> , S.216)	84
102	Ripple-Voltage in Abhängigkeit des Ausgangstroms. (aus 9.7 <i>Layout Empfehlungen aus dem Datenblatt des TPS56387</i> , S.216)	84
103	Duty Cycle in Abhängigkeit des Ausgangstroms. (aus 9.8 <i>Layout Empfehlungen aus dem Datenblatt des TPS565208</i> , S.217)	85
104	Effizienz in Abhängigkeit des Ausgangstroms. (aus 9.8 <i>Layout Empfehlungen aus dem Datenblatt des TPS565208</i> , S.217)	85

105	Junction-Temperatur des ICs in Abhängigkeit des Ausgangstroms. (aus 9.8 <i>Layout Empfehlungen aus dem Datenblatt des TPS565208</i> , S.217)	86
106	Ripple-Voltage in Abhängigkeit des Ausgangstroms. (aus 9.8 <i>Layout Empfehlungen aus dem Datenblatt des TPS565208</i> , S.217)	86
107	Layout Unterseite	88
108	Layout Oberseite	88
109	Layout des 12 V-Spannungsreglers	88
110	Layout des 5 V-Spannungsreglers	89
111	Oberseite des fertigen PCB-Designs	90
112	Unterseite des fertigen PCB-Designs	90
113	Gestapeltes PCB-Design	90
114	Werkzeuge	112
115	Addons	113
116	Transformations Werkzeuge	114
117	Translations Werkzeug	115
118	GCode Leser	116
119	Flussdiagramm-Rendering	125
120	Flussdiagramm-Events	126
121	Traces	134
122	Flussdiagramm-Auswahl	150
123	Auswahl Begrenzungsbox Visualisierung	152
124	Flussdiagramm-Slicing	153
125	Maskierungsfunktion im Prepare-Mode	168
126	Maskierungsfunktion im Preview-Mode	168
127	Hotendferigung des ersten Protoyps	178
128	Druckvalidierung des zweiten Protoyps	179
129	Split-Hotend	180
130	Teigmischung	180
131	Erfolgreiche Faserextrusion	180
132	Graphitmine nach der Fertigung	182
133	Senkerodierprozess	182
134	Oberes Loch	183
135	Unteres Loch	183
136	Krafttestmaschine	184
137	Schneidmechanismus Krafttest	184
138	Schneidmechanismus Krafttestdaten	184
139	Fertigung des Hotends Version 6	185
140	Testaufbau des ersten gefrästen Hotend-Prototyps	185
141	Druckversuch eines quadratischen Musters mit Faserintegration	186
142	Ergebnisse des ersten Hotend-Tests	186
143	Gefertigtes Hotend, Version 7.0.0	188
144	Coldendfertigung, Version 7.0.0	188
145	Zusammengebautes Cold- und Hotend	189
146	Montierter Druckkopf	189
147	Fasertestobjekt	190
148	Krafttestdaten	190
149	Konvektion 1	204
150	Konvektion 2	204
151	Angewendete Temperatur	204
152	Applikation Überblick	228
153	Applikation Überblick mit Fasevisualisierung	228

12 Tabellenverzeichnis

1	Meilensteine des Projekts	3
2	Mechanische Eigenschaften verschiedener Faserarten (Prototec GmbH, 2025).	8
3	Arbeitszeitübersicht der Entwicklungsphasen	18
4	Arbeitszeitübersicht der Entwicklungsphasen von Version 2	21
5	Arbeitszeitübersicht der Entwicklungsphasen von Version 3	25
6	Arbeitszeitübersicht der Entwicklungsphasen von Version 4	29
7	Arbeitszeitübersicht der Entwicklungsphasen von Version 5	39
8	Arbeitszeitübersicht der Entwicklungsphasen von Version 6	48
9	Arbeitszeitübersicht der Entwicklungsphasen von Version 7	55
10	Bewertung der Datenübertragung nach verschiedenen Kriterien	61
11	Bewertung der Bauformen nach verschiedenen Kriterien	63
12	Technische Details des Raspberry Pi 3 Model B+ und der Alternative Raspberry Pi 4 Model B (aus Digikey, 2025)	68
13	Technische Details des ATMEGA328P-AU, ATmega2560 und STM32 (Cortex-M3) (aus DigiKey, 2025h)	69
14	Technische Details des Schrittmotors 14HM08-0504S und der Alternative 17HS10-0704S (aus Stepperonline, 2024)	70
15	Technische Details des Schrittmotor-Treibers DRV8825 und der Alternative TMC2209 (aus AZ-Delivery, 2025a)	71
16	Technische Details des Servos MG90S und der Alternative Lynxmotion Micro 9G (aus AZ-Delivery, 2025c)	72
17	Technische Details der Spannungswandler TPS56837HRPAR und der Alternative LM2679SX-12/NOPB (aus DigiKey, 2025ad)	73
18	Technische Details der Spannungswandler TPS565208DDCR und der Alternative LM22676MRX-5.0/NOPB (aus DigiKey, 2025ac)	74
19	Designvorgaben gemäß JLCPCB (JLCPCB, 2025)	87
20	Vergleich der Libraries zur UI-Entwicklung in Rust	102
21	Vergleich gängiger Rendering-APIs	103
22	Übersicht der Materialkosten mit Lieferanten	205
23	Stückliste der Elektronikkomponenten	206

13 Programmausdrücke

1	Installation von KIAUH	94
2	Beispielhafte Konfigurationsdatei für den Schneid servo	96
3	Python-Implementierung zur Servo-Steuerung	97
4	Beispielhafte Konfigurationsdatei für den Faserextruder	97
5	Registrierung der Callback-Funktionen	97
6	Initialisierung der Trapezbewegungswarteschlange	98
7	Bewegungssteuerung in move	98
8	cargo.toml	101
9	lib.rs	101
10	main.rs	101
11	Start der Applikation	106
12	Die Application-Struktur mit Event-Loop-Implementierung	107
13	Die resumed-Funktion mit der Initialisierung der Module	107
14	Die Struktur des UiAdapter	108
15	Erstellung des UiAdapter	109
16	Definition der UI-Ereignisse	109
17	Verarbeitung der UI-Ereignisse	109
18	Rendering der Benutzeroberfläche	110
19	Screen Definition	110
20	Anzeige der UI-Komponenten	111
21	Hinzufügen von Toast-Benachrichtigungen	111
22	Viewport-Kalkulation	111
23	Definition der Gizmo-Transformationen	113
24	Verwaltung der Werkzeuge	113
25	Darstellung der Werkzeuge mit Icons	114
26	Transformationen je nach Werkzeug	115
27	Definition des EfficientReader	117
28	Definition von ReadSection	117
29	Integration des EfficientReader im GCodeTool	117
30	Struktur des CameraAdapter	118
31	Erstellung des CameraAdapter	118
32	Definition der Kamera-Ereignisse	118
33	Verarbeitung der Kamera-Ereignisse	119
34	Berechnung der Kameraposition in kartesischen Koordinaten	119
35	View- und Projektion-Matrizen der Orbit-Kamera	120
36	Rotation der Kamera durch Anpassung der Winkel	121
37	Zoom-Steuerung der Kamera	121
38	Einschränkungen der Kamerabewegung	121
39	Automatische Anpassung der Kameraentfernung	121
40	Definition der CameraController-Struktur	122
41	Verarbeitung von Fenster-Events	122
42	Verarbeitung von Geräte-Events	123
43	Grundstruktur des RenderAdapter	124
44	Adapter-Initialisierung	124
45	Renderintegration	125
46	Resize-Handling	125
47	Definition der Vertex-Struktur	126
48	Vertex-Buffer-Layout	127
49	Definition der TextureVertex-Struktur	127
50	Erstellung einer Tiefentextur	128
51	Laden einer Bildtextur	128
52	Verarbeitung einer Bildtextur	129
53	Definition der PipelineBuilder-Struktur	129
54	Erstellung einer Render-Pipeline	130
55	Definition des RenderDescriptor	130
56	Render-Pass Initialisierung	131

57	Definition der Model-Struktur	131
58	Transformation eines Modells	132
59	Render-Funktion	133
60	Grundstruktur des TraceTree	134
61	Initialisierung eines TraceTree	135
62	Erstellung von Bewegungspfaden	135
63	Definition des HitboxNode-Traits	135
64	Definition des InteractiveModel-Traits	136
65	Rendering-Funktionalität des TraceTree	136
66	Definition des TraceMesher	136
67	Generierung von Werkzeugpfaden	138
68	Definition der SlicedObject-Struktur	139
69	Erstellung eines SlicedObject	139
70	Verarbeitung eines MoveAndExtrude-Befehls	140
71	Mesh-Erstellung mit TraceMesher	140
72	Vertex Shader zur Sichtbarkeitsprüfung	141
73	Prüfung der Sichtbarkeit nach Layer	142
74	Prüfung der Sichtbarkeit nach Typ	142
75	Fragment Shader mit Blinn-Phong-Beleuchtung	142
76	Rendering der Werkzeugpfade	143
77	Laden eines gesuchten Objekts	143
78	Export von G-Code	144
79	Aktualisierung der Werkzeugpfad-Sichtbarkeit	144
80	Rendering der Modelle	145
81	Laden eines CAD-Modells aus einer Datei	145
82	Verarbeitung der CAD-Daten	146
83	Umrechnung in NDC	147
84	Berechnung der Clip-Koordinaten	147
85	Transformation in den Eye-Space	148
86	Transformation in Weltkoordinaten	148
87	Berechnung des Schnittpunkts mit einer Ebene	149
88	Hierarchische Hitbox-Verwaltung	149
89	Prioritätswarteschlange für Hitboxen	150
90	Selektierer für 3D-Objekte	151
91	AABB	151
92	Transformation von Objekten	151
93	Erstellung der Auswahlbegrenzung	152
94	Löschen einer Selektion von Objekten	152
95	Hauptfunktion zum Slicen eines Objekts (Kernmodul) (Ince, 2021)	153
96	Hauptfunktion zum Slicen eines Objekts (Schichtzerlegung) (Ince, 2021)	154
97	Berechnung des Schnittpunkts (Ince, 2021)	155
98	Iterieren durch den Triangle Tower (Ince, 2021)	155
99	Slice-Struktur zur Repräsentation einer Druckschicht (Ince, 2021)	156
100	<code>Slice::from_single_point_loop</code> (Ince, 2021)	157
101	<code>Slice::from_multiple_point_loop</code> (Ince, 2021)	157
102	Berechnung der Druckzeit (Ince, 2021)	158
103	Definition der Schichthöhe	158
104	Kombinieren von Einstellungen (Ince, 2021)	159
105	Validierung von Einstellungen (Ince, 2021)	159
106	Move-Struktur mit grundlegenden Bewegungsinformationen (Ince, 2021)	160
107	<code>MoveChain</code> als Gruppe zusammenhängender Bewegungen (Ince, 2021)	160
108	Erzeugung der Perimeter (Ince, 2021)	161
109	Bestimmung des Bewegungstyps	161
110	Bestimmung der Nahtstelle	162
111	Funktion zur partiellen Flächenfüllung	163
112	Funktion zur vollständigen Flächenfüllung	163
113	Berechnung der Spurflächen mittels <code>trace_area</code>	164
114	Struktur des <code>MergeFiberPass</code>	165

115	handle_single_move-Funktion	165
116	convert_chain_to_normal-Funktion	166
117	FiberChain-Struktur	166
118	FiberChain::find_next	167
119	FiberChain::find_cut_and_set	167
120	Erstellung einer neuen Maskierung	168
121	Beschneiden der Maske anhand vorhandener Objekte	169
122	Zufällige Variation der Maskierung	169
123	TraceType-Enum zur Kategorisierung von Druckpfaden	170
124	Command-Enumeration	171
125	Hilfsfunktionen zur Erkennung von Filament- und Faserbedarfen	172
126	Struktur für den generierten G-Code	173
127	Erstellung eines neuen G-Code Objekts	173
128	Erweiterung des Write-Traits für G-Code	174
129	Schreiben von G-Code in den Speicher	174
130	Schreiben von G-Code in eine Datei	174
131	Generierung von G-Code	175
132	Bewegung ohne Extrusion	175
133	Bewegung mit Extrusion	175
134	Umwandlung der Eingabekoordinaten in Vektoren	176
135	Berechnung des Richtungsvektors und der Länge	176
136	Bestimmung der Schnittposition	176
137	Segmentierung der Bewegungslänge	176
138	Berechnung des Extrusionsvolumens	176
139	Erste Extrusion bis zum Schnittpunkt	176
140	Schnitt der Faser	177
141	Zweite Extrusion nach dem Schnitt	177

14 Abkürzungsverzeichnis

ABS	Acrylonitrile Butadiene Styrene (3D-Druckmaterial)
API	Application Programming Interface
CFC	Composite Fiber Co-extrusion
CFR	Continuous Fiber Reinforcement
CNC	Computerized Numerical Control
CPU	Central Processing Unit
DLP	Digital Light Processing (3D-Druckverfahren)
FDM	Fused Deposition Modeling
FPU	Floating Point Unit
FRP	Fiber Reinforced Polymer
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
HSHT	High-Strength High-Temperature (Glasfaser)
I ² C	Inter-Integrated Circuit
IP	Internet Protocol
JSON	JavaScript Object Notation
Kevlar	Aramidfaser mit hoher Zugfestigkeit
MSAA	Multisample Anti-Aliasing
NDC	Normalized Device Coordinates
NTC	Negative Temperature Coefficient
Nylon	Polyamid (3D-Druckmaterial)
PA12	Polyamid 12 (SLS-Druckmaterial)
PCA	Principal Component Analysis
PCB	Printed Circuit Board
PCL	Polycaprolacton (biologisch abbaubar)
PEEK	Polyetheretherketon (Hochleistungskunststoff)
PETG	Polyethylenterephthalat-Glykol (3D-Druckmaterial)
PLA	Polylactic Acid (3D-Druckmaterial)
PTFE	Polytetrafluorethylen (Druckkopfbeschichtung)
PWM	Pulse Width Modulation
Resin	Flüssiges Photopolymer für SLA/DLP-Druck
SDA	Serial Data (I ² C-Bus)
SCL	Serial Clock (I ² C-Bus)
SIMD	Single Instruction, Multiple Data
SLA	Stereolithography (3D-Druckverfahren)
SLM	Selective Laser Melting
SLS	Selective Laser Sintering
SMN	Schneidmechanismus
TPU	Thermoplastisches Polyurethan (flexibles 3D-Druckmaterial)
UART	Universal Asynchronous Receiver Transmitter
UV	Ultraviolett (Licht für SLA-Druck)

USB	Universal Serial Bus
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
YAML	Yet Another Markup Language
2D	Zwei-Dimensional
3D	Drei-Dimensional
4D	Vier-Dimensional