Erstellung eines Programms zur Simulation einer zeitlichen Strahlformung mittels doppelbrechender Kristalle ausgehend von gaußförmigen Laserimpulsen

Masterarbeit der Physik

1. Gutachter: Prof. Dr. Wolfgang Lohmann
2. Gutachter: Dr. Tino Rublack

Eingereicht von:

Oliver Anton
Matrikelnummer:2932020

14. September 2015

Inhaltsverzeichnis

Nomenklatur

α	Ausdehnungskoeffizient
$\frac{dn}{dT}$	thermo-optischer Koeffizient
ϵ	Einhüllende der Wellenfunktion
ϵ_0	Dielektriztätskonstate
λ	Wellenlänge
μ_0	magnetische Feldkonstante
τ	Halbwertsbreite
ϕ	Phasenverschiebung
ϕ_L	Phasenverschiebung durch Kristalllänge
ϕ_T	Phasenverschiebung durch Temperatur
ϕ_{ges}	gesamte Phasenverschiebung
ϕ_m	Phasenverschiebung der n-ten Kopie
χ	elektrische Suszeptibilität
ψ	Kristallwinkel
ψ_n	Winkel des n-ten Kristalls
ω	Kreisfrequenz
a	Beschleunigung
c_0	Lichtgeschwindigkeit im Vakuum
c_{Medium}	Lichtgeschwindigkeit im Medium
d	Dicke des Kristalls
$E_m(t)$	Wellenfunktion der m-ten Kopie
E	elektrische Feldstärke
e	Elementarladung
I_m	maximale Intensität der n-ten Kopie
I	Intensität
K_{ges}	Gesamtanzahl der Kristalle

K_n	Kristall Nummer
k	Wellenzahl
m_{ges}	Anzahl der Kopien des Eingangspulses
M	Testpunkt
m	Masse
n_o	Brechungsindex der ordentlichen Achse
n_{ao}	Brechungsindex der außerordentlichen Achse
n_e	Elektronendichte
n	Brechungsindex
P	Polarisation
t_0	Erwartungswert der Kopie
t_{Puls}	zeitliche Länge des Intensitätsprofils
T	Temperatur
t	Zeit
x	Ort

Selbstständigkeitserklärung

Der Verfasser der Arbeit erklärt hiermit, dass die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer, als der angegebenen Hilfsmittel, angefertigt wurde. Gedanken, welche aus fremden Quellen direkt oder indirekt übernommen wurden, sind ausnahmslos als solche gekennzeichnet. Die Arbeit ist in gleicher oder ähnlicher Form im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ort,Datum Unterschrift des Verfassers

1 Einleitung

In der Wissenschaft gibt es verschiedene Möglichkeiten zur Betrachtung kleiner Strukturen. Eine Möglichkeit ist die Verwendung von Röntgenlicht. Hierbei sind jedoch Grenzen durch das Abbe-Limit [1] gesetzt. Dieses sagt aus, dass die minimale Objektgröße, welche betrachtet werden kann, direkt proportional zur Wellenlänge des genutzten Lichts ist. Bei der Betrachtung von Strukturen im Bereich weniger Nanometer muss deshalb Röntgenlicht genutzt werden. Die Energie der Photonen liegt nahe der Zerstörungsschwelle der Proben. Zur Durchführung einer aussagekräftigen Messung vor der Zerstörung der Probe, muss das Röntgenlicht eine hohe Intensität und Brillianz aufweisen. Die Brillianz beschreibt die Bündelung des Strahls.

Für die Erzeugung von Röntgenlicht mit diesen Eigenschaften werden Teilchenbeschleuniger genutzt. Eine Bauform von Beschleunigern nutzt Photokathoden zur Erzeugung der Elektronen. Die Elektronenpakete werden mit Laserpulsen auf einer Photokathode erzeugt. Die Photonen der Laserpulse weisen eine Energie auf, welche größer ist als die Austrittsenergie der Elektronen. Die Elektronen werden durch den Laserpuls in das Vakuumniveau angeregt und durch ein äußeres elektrisches Feld von der Photokathode weg beschleunigt.

Danach werden die Elektronen auf nahezu Lichtgeschwindigkeit beschleunigt und durch die Undulatoren geleitet. Undulatoren sind eine Anordnung von mehreren Dipolmagneten in abwechselnder Nord-Süd-Ausrichtung. Die Elektronen fliegen in ihnen auf einer wellenförmigen Flugbahn. Bei jeder Richtungsänderung geben sie, entsprechend ihrer Energie, Synchrotronstrahlung einer bestimmten Wellenlänge in Flugrichtung ab. Die Synchrotronstrahlung ist gebündeltes Röntgenlicht. Dieses wird zur Bestrahlung der Probe verwendet.

Solche Beschleuniger existieren unter anderem am Deutschen Elektronen Synchrotron (DESY). DESY ist ein Forschungsinstitut in Deutschland, mit den Standorten Hamburg und Zeuthen bei Berlin. Die Quellen für Synchrotronstrahlung sind unter anderem PETRA III, FLASH sowie der noch im Bau befindliche XFEL.

Als Testeinrichtung der Beschleuniger wurde die Gruppe *P*hoto *I*njector *T*est Facility in *Z*euthen (PITZ) gegründet. In dieser Gruppe wird unter anderem erforscht, wie die Emittanz des Elektronenstrahls verringert werden kann. Die Emittanz beschreibt die Fokussierung des Elektronenstrahls. Diese Forschungen sind notwendig, da die Eigenschaften des Elektronenstrahls maßgeblich die Eigenschaften des Röntgenpulses beeinflussen. In der Gruppe PITZ wurden Simulationen durchgeführt, um zu verstehen, welche zeitlichen Laserpulsprofile die Emittanz des Elektronenstrahls in welcher Art verändern [2]. Im Beschleuniger FLASH werden zum Beispiel gaußförmige Laserpulse verwendet.

Es wurde festgestellt, dass sich Laserpulse mit einem Flat-Top-Profil besser als gaußförmige Pulse eignen, um geringe Emittanzen zu erreichen. In Vergleichsmessungen, in welchen Laserpulse mit einem Flat-Top-Profil (siehe Abb.1) verwendet wurden, konnten diese Ergebnisse bestätigt werden [4]. Es wurde in den

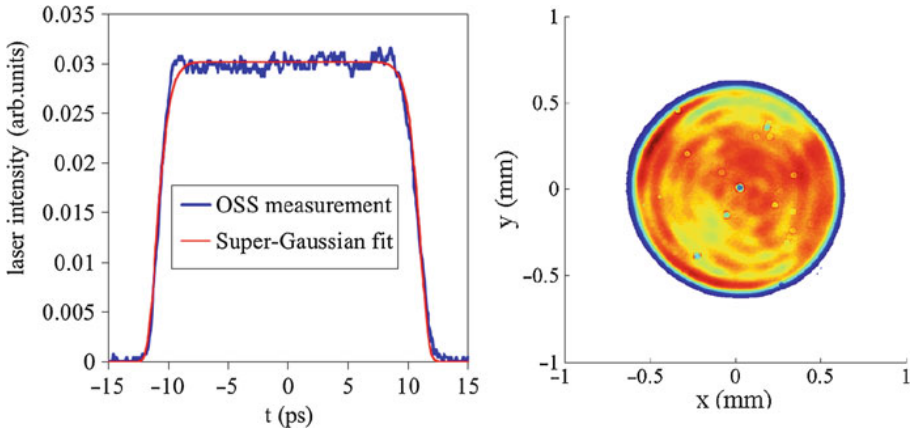


Abbildung 1: Flat-Top-Laserprofil des Lasersystems bei PITZ(links blaue Kurve: zeitliches Profil gemessen im Optical Sampling System(OSS), links rote Kurve: Fit der Messung mit der Supergaußfunktion, rechts: räumliches Profil)[3]

Messungen jedoch auch festgestellt, dass die momentan verwendeten Profile Abweichungen von einem idealen Flat-Top-Profil aufweisen. Dies ist in Abb.1 im linken Diagram zu sehen. Ein Ziel bei PITZ ist, die Abweichung zu minimieren.

Um dieses Ziel zu erreichen, soll die Formung des Intensitätsprofil zunächst simuliert werden. In weiteren Schritten soll das Intensitätsprofil im Beamshaper automatisch eingestellt werden. Dies geschieht im aktuellen Aufbau manuell. Daraus entstehen verschiedene Nachteile. Die manuelle Einstellung der Winkel ist oft nicht optimal und zeitaufwendig.

Das Ziel dieser Arbeit ist die Entwicklung eines Programms, welches das Intensitätsprofil des Beamshapers simulieren und in späteren Entwicklungsschritten den Beamshaper selbstständig steuern kann. Hierzu werden zunächst die auftretenden Prozesse erläutert und das Lasersystem bei PITZ beschrieben. Auf den theoretischen Betrachtungen aufbauend, wird ein Berechnungsalgorithmus zur Simulation des Intensitätsprofils entwickelt [5]. Somit ist es möglich, Einstellungen am Beamshaper, ohne Beeinträchtigungen des Betriebs des Beschleunigers, zu testen. Darauf beruhend wird für den automatischen Betrieb der Software ein Suchalgorithmus entwickelt. Der Suchalgorithmus soll selbstständig, vom Nutzer definierbare, Intensitätsprofile finden können und die dafür notwendigen Einstellungen am Beamshaper ausgeben. Hier werden Entwicklungsmöglichkeiten zur automatischen Korrektur des durch den Beamshaper geformten Intensitätsprofils integriert. Somit kann ein vollautomatisches System entwickelt werden. Abschließend werden berechnete Profile und Messungen vorgestellt.

2 Theoretische Grundlagen

In der Gruppe PITZ wird ein Lasersystem verwendet, welches vom Max Born Institut (MBI) entwickelt wurde. Das Lasersystem wird im Folgenden als MBI-Lasersystem bezeichnet. Im MBI-Lasersystem werden Laserpulse mit einer Wellenlänge von ca. 1030 nm erzeugt. Das zeitliche Profil der Laserpulse wird anschließend in einem Beamshaper manipuliert. Die Laserpulse werden daraufhin in mehreren Stufen verstärkt. Als letzter Schritt wird eine Frequenzkonversion durchgeführt.

Der Laserpuls wird mit Hilfe doppelbrechender Kristalle geformt. Die Verstärkung und die Frequenzkonversion werden mit Hilfe nichtlinearer Prozesse in Kristallen durchgeführt. In den folgenden Abschnitten werden die linearen und nichtlinearen Effekte, welche zwischen der Generierung der Laserpulse und der Messung des zeitlichen Intensitätsprofils auftreten, betrachtet. Es werden zunächst der Jones-Formalismus und die doppelbrechenden Kristalle beschrieben. Darauf folgt die Beschreibung der nichtlinearen Prozesse.

2.1 Jones Formalismus

Im MBI-Lasersystem kann das zeitliche Profil der Laserpulse verändert werden. Dies geschieht in einem Beamshaper mit Hilfe doppelbrechender Kristalle. Zur Formung des Intensitätsprofils in den doppelbrechenden Kristallen werden lineare optische Prozesse genutzt. Für die Beschreibung der stattfindenden Prozesse kann der Jones-Formalismus genutzt werden. Der Jones-Formalismus beschreibt die linearen optischen Gesetzmäßigkeiten. In diesem Formalismus wird Licht als elektromagnetische Welle betrachtet,

$$\vec{E}(x,t) \quad = \quad \vec{E_0} * e^{(i(kx-\omega t))}.$$

Hierbei beschreibt $\vec{E}(x,t)$ das elektrische Feld in Abhängigkeit des Ortes x und der Zeit t, k steht für den Wellenvektor und ω für die Kreisfrequenz. Der Vektor $\vec{E_0}$ beschreibt die Polarisationsebene des einfallenden Lichtes. Für den Fall, dass der Vektor normiert ist, lautet er $\vec{E_0} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ für in x-Richtung polarisiertes Licht und $\vec{E_0} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ für in y-Richtung polarisiertes Licht. Wenn nur lineare Prozesse die elektromagnetische Welle beeinflussen, können nur 3 Eigenschaften verändert werden. Die Polarisationsebene der elektromagnetischen Welle kann gedreht werden. Zudem kann die Welle in der Phase verschoben oder polarisiert werden. Jede dieser Beeinflussungen kann durch jeweils eine Matrix beschrieben werden. Die Matrizen sind in Tabelle 1 aufgelistet. Durch die Kombination dieser drei Matrizen kann jeder lineare Effekt in optischen Bauelementen, welcher die Eigenschaften der elektromagnetischen Welle beeinflusst, beschrieben werden.

Tabelle 1: Matrizen des Jones-Formalismus[6]

Drehmatrix	$D(\psi) = \begin{pmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{pmatrix}$
Phasenverschiebung	$V(\phi) = \begin{pmatrix} exp(i\phi) & 0 \\ 0 & exp(i\phi) \end{pmatrix}$
Polarisation	$P_x = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}; P_y = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$

2.2 Doppelbrechende Kristalle

Im MBI-Lasersystem werden doppelbrechende Kristalle zur Beeinflussung der Laserpulse genutzt. Kristalle werden als doppelbrechend bezeichnet, wenn sie unterschiedliche Brechungsindizes (n_1, n_2, n_3) für verschiedene Raumachsen aufweisen. Die unterschiedlichen Brechungsindizes resultieren aus der Kristallsymmetrie. Aufgrund der unterschiedlichen Anzahl an Atomen, die auf dem Weg des Lichts in der Polarisationsebene liegen, ergeben sich unterschiedliche Ausbreitungsgeschwindigkeiten für verschiedene Polarisationsebenen. Das Photon wird von jedem, in seinem Weg liegenden, Atom absorbiert und wieder emittiert. Nun kann man in Abb.2 die Anzahl der Atome, welche das Photon passieren muss, betrachten. Das Photon auf der linken Seite passiert in seiner Flugbahn nur ein Atom.

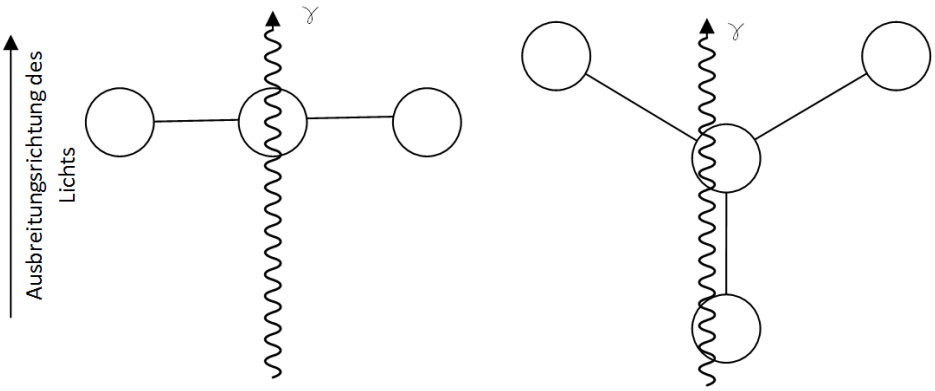


Abbildung 2: Atomare Struktur eines doppelbrechenden Stoffes aus verschiedenen Perspektiven. Die rechte Struktur zeigt die linke Struktur nach einer Drehung um 90°, senkrecht zur Ausbreitungsrichtung des Lichts

Es wird somit nur einmal absorbiert und wieder emittiert. Das Photon auf der rechten Seite muss hingegen zwei Atome passieren. Dies verlängert die Flugzeit des rechten Photons, da es zweimal absorbiert und wieder emittiert wird. Das Licht weist somit unterschiedliche Ausbreitungsgeschwindigkeiten für verschie-

dene Polarisationsebenen auf. Aufgrund des Zusammenhangs von Brechungsindex, Lichtgeschwindigkeit im Vakuum (c_0) und Lichtgeschwindigkeit im Medium (c_{Medium}) ($n = \frac{c_0}{c_{Medium}}$), hat der Festkörper für unterschiedliche Achsen unterschiedliche Brechungsindizes. Um diesen Effekt sehen zu können, muss der Stoff eine Kristallstruktur besitzen. Bei einer ungeordneten Struktur heben sich die Effekte, aufgrund der zufälligen Ausrichtung jedes einzelnen Moleküls, auf. Die unterschiedlichen Brechungsindizes werden als Tensor geschrieben. Es ist möglich, die unterschiedlichen Brechungsindizes als einen Indexellipsoid grafisch darzustellen (Abb.3).

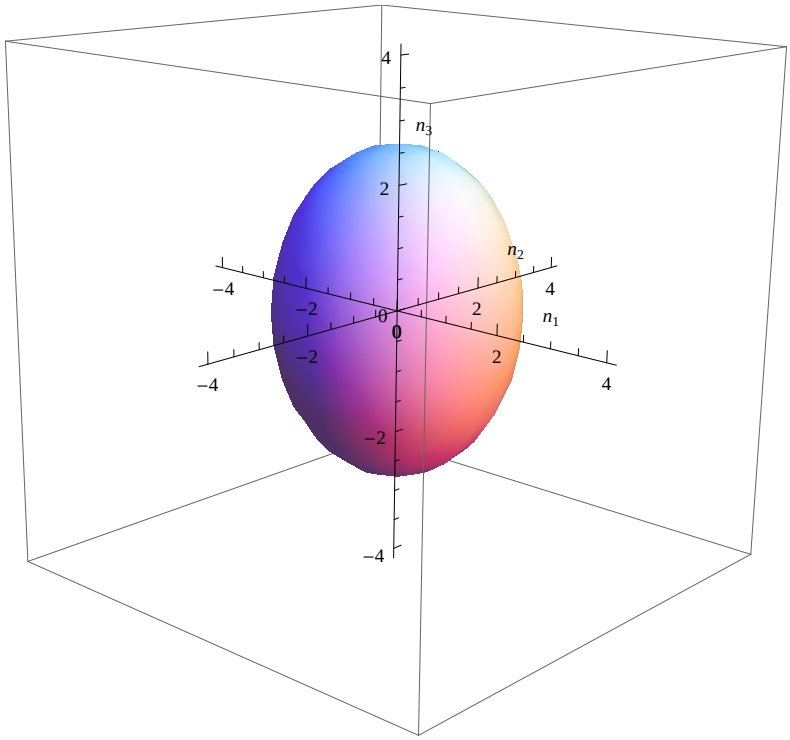


Abbildung 3: Ellipsoid des Hg_2Cl_2-Kristalls($n_1 = n_2 = 1,973$; $n_3 = 2.656$)[7]

Der Ellipsoid wird durch die Formel

$$\frac{x_1^2}{n_1^2} + \frac{x_2^2}{n_2^2} + \frac{x_3^2}{n_3^2} \quad = \quad 1$$

beschrieben. Die Symmetrieachsen des Ellipsoids werden durch die Brechungsindizes definiert. Doppelbrechende Kristalle lassen sich in zwei Arten unterteilen.

Es gibt Kristalle, bei welchen zwei Brechungsindizes gleich sind ($n_1 = n_2 \neq n_3$). Diese Kristalle werden optisch einachsig genannt. So sind zum Beispiel tetragonale Kristalle optisch einachsig. Kristalle, für die gilt $n_{1,2} < n_3$, werden positiv

einachsig genannt. Kristalle, für die $n_{1,2} > n_3$ gilt, heißen negativ einachsig. Eine weitere Möglichkeit ist, dass $n_1 \neq n_2 \neq n_3$ gilt. Diese Kristalle werden optisch zweiachsig genannt. Bei optisch zweiachsigen Kristallen ist nicht bestimmbar, ob sie positiv oder negativ zweiachsig sind. Diese Eigenschaft weisen Kristalle mit orthorhombischem Kristallgitter auf.

In einachsigen Kristallen haben die optischen Achsen verschiedene Eigenschaften. Die ordentliche und die außerordentliche Achse werden durch die Ausrichtung des elektrischen Feldes zur Ausbreitungsrichtung des Lichtes definiert. Die Ausbreitungsrichtung des Lichts im Kristall lässt sich durch Betrachtung der Elementarwellen konstruieren (Abb.4). Die Schwingung des Lichts, welches auf die ordentliche Achse projiziert wurde, wird nur durch die Ausbreitungsgeschwindigkeit auf der ordentlichen Achse beeinflusst. Die Elementarwellen sind

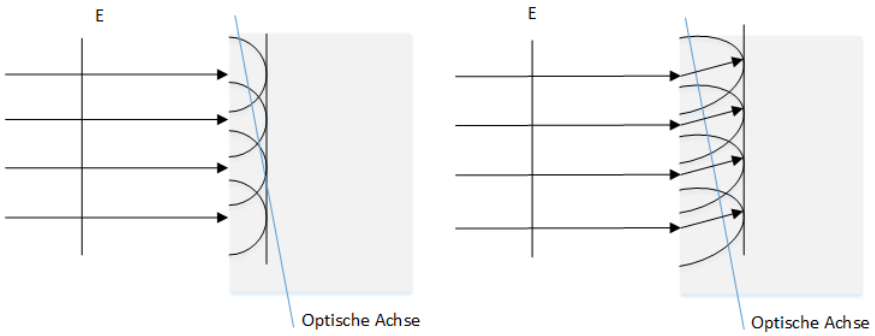


Abbildung 4: Konstruktion des Strahles im doppelbrechenden Kristall (links: ordentliche Achse, rechts außerordentliche Achse)

Kugelwellen (siehe Abb.4 links). Sie werden nicht abgelenkt. Für die außerordentliche Achse gilt dies nicht. Die Elementarwellen setzen sich aus der Ausbreitungsgeschwindigkeit beider optischer Achsen zusammen. Da die Ausbreitungsgeschwindigkeiten unterschiedlich sind, sind die Elementarwellen elliptisch. Je nach Ausrichtung der optischen Achse breiten sich die Elementarwellen, bei senkrechtem Einfall des Lichtes, nicht senkrecht zur Kristalloberfläche aus. Das Licht kann somit gebeugt werden (siehe Abb.4 rechts). Dadurch ist es möglich, einen Lichtpuls in zwei Lichtpulse aufzuspalten.

Der Brechungsindex der doppelbrechenden Kristalle ist auch von der Temperatur T abhängig.

$$n(T) \quad = \quad n + \frac{dn}{dT} * T$$

$\frac{dn}{dT}$ beschreibt den thermo-optischen-Koeffizenten. Durch die Veränderung der Weglänge des Lichtes, aufgrund der thermischen Ausdehnung des doppelbrechenden Kristalls, sowie der Änderung des temperaturabhängigen Brechungsindex,

kann die Phasenverschiebung zwischen der ordentlichen und der außerordentlichen Welle beeinflusst werden. Die temperaturabhängige Änderung des Brechungsindex ist sehr gering gegenüber dem Brechungsindex des Materials z.B. 0,3-16$\frac{10^{-6}}{K}$ für Yttriumvanadat [8].

2.2.1 Intensitätsverteilung in doppelbrechenden Kristallen

Für die Simulation eines Pulses, welcher eine Reihe doppelbrechender Kristalle passiert, ist es notwendig, die Aufteilung der Gesamtintensität des einfallenden Pulses auf die ordentliche und außerordentliche Welle zu kennen. Zur Berechnung der Intensitätsverteilung wird der, in Abschnitt 2.1 erklärte, Jones-Formalismus verwendet.

Im doppelbrechenden Kristall wird der einfallende Laserpuls $\vec{E_0} = \begin{pmatrix} E_{0x} \\ E_{0y} \end{pmatrix}$ auf die beiden optischen Achsen projiziert.

$$\vec{E} = \begin{pmatrix} E_x \\ E_y \end{pmatrix} = \begin{pmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{pmatrix} * \vec{E_0} * \exp(i(kx - \omega t))$$

$$\vec{E} = \begin{pmatrix} E_x \\ E_y \end{pmatrix} = \begin{pmatrix} (E_{0x}cos(\psi) - E_{0y}sin(\psi)) \\ (E_{0x}sin(\psi) + E_{0y}cos(\psi)) \end{pmatrix} \exp(i(kx - \omega t))$$

ψ beschreibt den Kristallwinkel. Es entspricht jeweils eine Komponente des Vektors $\vec{E} = \begin{pmatrix} E_x \\ E_y \end{pmatrix}$ der Amplitude der Projektion der einfallenden elektromagnetischen Welle, auf eine der optischen Achsen des Kristalls. Es wird nun angenommen, dass die Komponente E_y des Vektors der Projektion auf der optischen Achse mit der schnellen Ausbreitungsgeschwindigkeit und E_x der Projektion auf die optische Achse des Kristalls mit der geringeren Ausbreitungsgeschwindigkeit entspricht. Auf der optischen Achse mit der schnellen Ausbreitungsgeschwindigkeit ist der Brechungsindex geringer. Aus den Laufzeitunterschieden wird eine Phasenverschiebung ϕ_L erzeugt

$$\phi_L = \frac{\pi}{\lambda} d(n_{ao}\text{-}n_o).$$

Hierbei steht λ für die Wellenlänge des Lichts, d für die Dicke des Kristalls, n_{ao} für den Brechungsindex der außerordentlichen Welle und n_o für den Brechungsindex der ordentlichen Welle. Zudem wird, durch die Temperatur der Kristalle, der Brechungsindex geändert und somit eine Phasenverschiebung ϕ_T erzeugt. Zur übersichtlicheren Beschreibung des Problems werden diese beiden Anteile der Phasenverschiebung im Weiteren nicht gesondert betrachtet und als gesamte Phasenverschiebung ϕ_{ges} zusammengefasst,

$$\phi_{ges} = \phi_T + \phi_L.$$

Hinzu kommt, dass zum einen die Kristalle in der Realität keine perfekte glatte Oberfläche haben, zum anderen ist die temperaturabhängige Phasenverschiebung rechnerisch für einen realen Kristall nicht exakt zu berechnen. In der

praktischen Anwendung wird die Phasenverschiebung ebenfalls nur insgesamt betrachtet. Die Phasenverschiebung kann nun auf verschiedenen Wegen in die Formel eingebracht werden. Man kann sie entweder auf die beiden Komponenten auf den optischen Achsen aufteilen, oder nur die Welle einer Achse um die gesamte Phasenverschiebung verschieben. In der folgenden Herleitung wird eine Welle um die gesamte Phasenverschiebung verschoben und die andere Welle wird nicht verschoben. Die Phasenverschiebung wird mit 2 Matrizen eingebracht:

$$\begin{aligned} P_x &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \\ P_y &= \begin{pmatrix} 0 & 0 \\ 0 & exp(-i\phi_{ges}) \end{pmatrix}. \end{aligned}$$

Diese Matrizen entsprechen den Matrizen für einen Polarisator in x- bzw. y-Richtung aus dem Jones-Formalismus. Da im Kristall jedoch kein Licht absorbiert wird, dienen sie nur der Auswahl des ordentlichen oder außerordentlichen Strahles. Nach dem Kristall wird der ordentliche und der außerordentliche Strahl durch folgende Formel beschrieben:

$$\begin{aligned} \vec{E_o} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \exp(i(kx - \omega t)), \\ \vec{E_{ao}} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \exp(i(kx - \omega t)) * exp(-i\phi_{ges}). \end{aligned}$$

Nun hat man die Intensitätsverteilung im System des Kristalls. Dies ist unpraktisch, wenn das Ergebnis in späteren Rechnungen weiter verwendet werden soll. Man müsste den Winkel des folgenden Elements relativ zum Winkel des Kristalls angeben. Es ist praktischer, wenn alle Winkel relativ zu einem festen Winkel angegeben werden. Hier wird der Winkel des Eingangspolarisators als Referenzwinkel gewählt. Die beiden Strahlen müssen nun in das Koordinatensystem des Eingangspolarisators zurück transformiert werden.

$$\begin{aligned} \vec{E_o} &= \begin{pmatrix} cos(-\psi) & -sin(-\psi) \\ sin(-\psi) & cos(-\psi) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \\ &\quad \exp(i(kx - \omega t)) \end{aligned}$$

$$\vec{E_o} = \begin{pmatrix} cos(\psi) * (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \exp(i(kx - \omega t)) \\ (-1) * sin(\psi) * (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \exp(i(kx - \omega t)) \end{pmatrix}$$

$$\begin{aligned} \vec{E_{ao}} &= \begin{pmatrix} cos(-\psi) & -sin(-\psi) \\ sin(-\psi) & cos(-\psi) \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \\ &\quad \exp(i(kx - \omega t)) * exp(-i\phi_{ges}) \end{aligned}$$

$$\vec{E_{ao}} = \begin{pmatrix} sin(\psi) * (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \exp(i(kx - \omega t) - i\phi_{ges}) \\ cos(\psi) * (E_{0x}cos(\psi) - E_{0y}sin(\psi)) * \exp(i(kx - \omega t) - i\phi_{ges}) \end{pmatrix}$$

Aus diesen Pulsen kann nun die Intensität des gesamten Pulses berechnet werden. Hierzu wird über alle Teilpulse summiert. Die Gesamtintensität ergibt sich dann zu:

$$I_{ges}(x,t) = \frac{1}{2}c\epsilon_0 \left(\vec{E_o}(x,t) + \vec{E_{ao}}(x,t)\right)^2 \quad (1)$$

2.3 Nichtlineare Prozesse

Die Beeinflussung des Laserpulses durch nichtlineare Prozesse wird im Simulationsprogramm, welches in dieser Arbeit später vorgestellt wird, nicht betrachtet. Die Simulation gibt das Intensitätsprofil direkt nach der Formung im Beamshaper aus. Die Messung des, im Beamshaper eingestellten, Intensitätsprofils findet jedoch erst nach der Verstärkung und der Frequenzvervierfachung statt. Die hier stattfindenden Prozesse sind die Verstärkung, Frequenzverdopplung und die Differenzfrequenzerzeugung. Die Stärke dieser Effekte hängt von der Intensität der verwendeten Laserpulse ab. Sie haben deshalb einen signifikanten Einfluss auf die Pulsform. Es ist somit notwendig, die Einflüsse zu kennen. Dadurch können die Unterschiede der Ergebnisse der Simulation zur Messung erklärt werden. Im MBI-Lasersystem werden nichtlineare Effekte zweiter Ordnung genutzt. Die Herleitungen werden deshalb für die Effekte erster und zweiter Ordnung durchgeführt.

In der Laserphysik werden nichtlineare Prozesse zur Beeinflussung des Laserstrahls genutzt. Nichtlineare Effekte treten immer auf. Ihre Stärke ist jedoch von der Intensität des eingestrahlten elektromagnetischen Feldes abhängig. Die Stärke der nichtlinearen Effekte steigt exponentiell zur Intensität der eingestrahlten elektromagnetischen Welle. Bei geringen elektrischen Feldstärken sind die Effekte so klein, dass sie keinen signifikanten Einfluss haben bzw. nicht messbar sind. Da bei Lasern hohe Feldstärken auftreten können, wird der Einfluss nichtlinearer Prozesse stärker. Die Ursache für nichtlineare Effekte ist die Schwingung des am Atom gebundenen Elektrons. Die Auslenkung ist sehr klein. Zur Herleitung der Effekte wird zunächst die Polarisation P durch das Elektron betrachtet.

$$P = \epsilon_0 \chi E(t)$$

Hier ist ϵ_0 die Dielektriztätskonstate des Vakuums und χ die elektrische Suszeptibilität. Der lineare Zusammenhang gilt nur für kleine Feldstärken. Deshalb muss eine Taylorreihenentwicklung um die Ruhelage des Elektrons durchgeführt und die Terme höherer Ordnung betrachtet werden:

$$P = P(0) + E\left(\frac{dP}{dE}\right)_0 + \frac{1}{2}E^2\left(\frac{d^2P}{dE^2}\right)_0 + ... \quad (2)$$

Für diese Arbeit sind nur Effekte zweiter Ordnung von Bedeutung, weshalb die Reihenentwicklung hier abgebrochen wird. Da $E \ll 1$ gilt, werden die Terme

höherer Ordnung sehr klein. Die aus ihnen resultierenden Effekte sind zum Beispiel die Selbstphasenmodulation oder die Selbstfokussierung. Für ein Material ohne statisches Dipolmoment ist P(0)=0 anzunehmen. Für das Elektron kann die Schwingungsgleichung $E = E_0 e^{i\omega' t}$angenommen werden:

$$\left(\frac{dP}{dE}\right)_0 = \left(\frac{dP}{dE}\frac{dE}{dt}\right)_0 = \epsilon_0 \chi E_0 \omega' \tag{3}$$

$$\left(\frac{d^2P}{dE^2}\right)_0 = \epsilon_0 \chi E_0 \omega'^2. \tag{4}$$

Wenn die Gleichungen 3 und 4 und $\chi = \chi_1 + \chi_2 + ...$ sowie P(0)=0 in die Gleichung 2 eingesetzt werden ergibt sich:

$$P = \epsilon_0 \chi_1 E + \frac{1}{2}\epsilon_0 \chi_2 E^2 + ... \tag{5}$$

Für die einfallende elektromagnetische Welle gilt $E(\omega) = E_0 cos(\omega t)$. Somit ergibt sich aus Gleichung 5:

$$\begin{aligned} P &= \epsilon_0 \chi_1 E_0 cos(\omega t) + \frac{1}{2}\epsilon_0 \chi_2 E_0^2 cos^2(\omega t) \\ &= \epsilon_0 \chi_1 E_0 cos(\omega t) + \frac{1}{2}\epsilon_0 \chi_2 E_0^2 \left(\frac{1 + cos(2\omega t)}{2}\right) \\ &= \underbrace{\epsilon_0 \chi_1 E_0 cos(\omega t)}_{1} + \underbrace{\frac{1}{4}\epsilon_0 \chi_2 E_0^2}_{2} + \underbrace{\frac{1}{2}\epsilon_0 \chi_2 E_0^2 cos(2\omega t)}_{3}. \end{aligned} \tag{6}$$

Die Formel besteht aus drei Termen. Die Ordnung des Effektes wird nach der Ordnung des Terms, welcher ihn beschreibt, definiert. Teil 1 beschreibt einen Effekt erster Ordnung. Dieser Term beschreibt die eingestrahlte Welle, welche wieder emittiert wird. Term 2 und 3 beschreiben nichtlineare Effekte zweiter Ordnung. Die genaue Betrachtung der Terme zweiter Ordnung und der aus ihnen folgenden Effekte erfolgt im nächsten Abschnitt.

2.3.1 Frequenzverdopplung

Die Terme 2 und 3 sind aufgrund von $\chi_2 \sim 10^{-10}\, cm/V$ [9] sehr klein. Term 2 beschreibt die optische Gleichrichtung und Term 3 die Frequenzverdopplung, auch Erzeugung der zweiten Harmonischen genannt. Die erzeugte Welle hat die Frequenz 2ω. Es ist zu erkennen, dass die Intensität der frequenzverdoppelten Welle quadratisch abhängig ist von der Intensität der eingestrahlten Welle. Es wird so ersichtlich, warum bei kleinen Feldstärken die Effekte höherer Ordnung vernachlässigt werden können und sie bei großen Feldstärken stärker sichtbar werden.

Für die Berechnung der Intensität der frequenzverdoppelten Welle muss die elektromagnetische Welle als komplexe Wellenfunktion betrachtet werden:

$$\begin{aligned} E(x) &= \frac{1}{2}\left(\epsilon_{2\omega}(x)e^{-i(2\omega-k_{2\omega}x)} + \epsilon_{2\omega}(x)e^{i(2\omega-k_{2\omega}x)}\right) \\ k_{2\omega} &= n(2\omega)\frac{2\omega}{c}. \end{aligned} \tag{7}$$

$\epsilon_{2\omega}$ ist hier die Einhüllende der Wellenfunktion mit der Frequenz 2ω und n der Brechungsindex des Materials. Es wird angenommen, dass sich $\epsilon_{2\omega}$ wesentlich langsamer ändert als die Schwingung selbst[10]. Die Näherung wird "slowly varying envelope approximation" genannt:

$$k_{2\omega}\frac{\partial\epsilon_{2\omega}}{\partial z} \gg \frac{\partial^2\epsilon_{2\omega}}{\partial z^2}.$$

Es wird nun Gleichung 7 in die Grundgleichung der nichtlinearen Optik(Gleichung 8):

$$\begin{aligned} \nabla^2 E - \frac{n_{2\omega}^2}{c^2}\frac{\partial^2 E}{\partial t^2} &= \mu_0\frac{\partial^2}{\partial t^2}P \\ ik_{2\omega}\frac{\partial\epsilon_{2\omega}}{\partial z}(e^{-i(2\omega-k_{2\omega}x)} + e^{i(2\omega-k_{2\omega}x)}) &= -2\omega^2\mu_0 P_{2\omega}(e^{-i(2\omega-k_{2\omega}x)} + e^{i(2\omega-k_{2\omega}x)}) \\ \frac{\partial\epsilon_{2\omega}}{\partial z} &= 2\frac{i\mu_0\omega^2}{k_{2\omega}}P_{2\omega}(e^{-i(2\omega-k_{2\omega}x)} + e^{i(2\omega-k_{2\omega}x)}), \end{aligned} \tag{8}$$

eingesetzt. Hier ist μ_0 die magnetische Feldkonstante. Nun wird die Änderung der Einhüllenden der frequenzverdoppelten Welle über die Länge des Kristalls betrachtet:

$$\frac{\partial\epsilon_{2\omega}}{\partial x} = \frac{2i\omega}{n(2\omega)}\sqrt{\frac{\mu_0}{\epsilon_0}}b\epsilon_\omega^2(x)e^{i\Delta kx} \tag{9}$$

$$b = \frac{n_e a e^3}{2m^2(\omega_0^2 - 4\omega^2)(\omega_0^2 - \omega^2)^2}.$$

Hier ist n_e die Elektronendichte, a die Beschleunigung, e die Elementarladung und m die effektive Masse des Elektrons. Es wird nun angenommen, dass die Erzeugung der 2. Harmonischen ein sehr ineffizienter Prozess ist. Es gilt dann $\epsilon_\omega^2(z) \approx \epsilon_\omega^2(0)$[10]. Durch Integration der Formel 9 über die Kristalllänge erhält man dann die Einhüllende der frequenzverdoppelten Welle:

$$\epsilon_{2\omega}(d) = \frac{i\omega}{n(\omega)}\sqrt{\frac{\mu_0}{\epsilon_0}}b\epsilon_\omega^2(0)de^{\left(\frac{i\Delta kd}{2}\right)} * \frac{sin\left(\frac{i\Delta kd}{2}\right)}{\frac{i\Delta kl}{2}}. \tag{10}$$

Die Intensität I der eingestrahlten und der frequenzverdoppelten Welle lässt sich durch folgende Formel bestimmen:

$$I_{\omega,2\omega} = \frac{n(\omega,2\omega)}{2}\sqrt{\frac{\epsilon_0}{\mu_0}}|\epsilon_{\omega,2\omega}|^2. \quad (11)$$

Durch einsetzen von Formel 10 in Formel 11 erhält man für die Intensität

$$I_{2\omega}(d) = 2\left(\frac{\epsilon_0}{\mu_0}\right)^{2/3}\frac{\omega^2 b^2}{n^2(\omega)n^2(2\omega)}I_\omega^2(0)d^2\frac{sin^2\left(\frac{\Delta kd}{2}\right)}{\left(\frac{\Delta kd}{2}\right)^2}.$$

Die Intensität der frequenzverdoppelten Welle hängt quadratisch von der Intensität der eingestrahlten Welle, sowie von der Länge des Kristalls ab. Zudem oszilliert die Intensität in Abhängigkeit von der Länge des Kristalls. Kleine Abweichungen in den Messwerten können zu großen Schwankungen zwischen errechneter und gemessener Intensität führen. Grund hierfür ist die quadratische Abhängigkeit. Die Länge des Kristalls sowie der Brechungsindex sind von der Raumtemperatur abhängig. Die Oberfläche des Kristalls ist nicht atomar glatt [9, 10].

2.3.2 Differenzfrequenzerzeugung

Die Differenzfrequenzerzeugung ist ein Effekt der Frequenzmischung. Für die Herleitung wird zunächst die Taylorreihenentwicklung der Polarisation (Formel 5) aus Abschnitt 2.3,

$$P = \epsilon_0\chi_1 E(t) + \epsilon_0\frac{1}{2}\chi_2 E^2(t), \quad (12)$$

sowie die Summe der Wellenfunktionen zweier einfallender elektromagnetischer Wellen (Formel 14) betrachtet:

$$\begin{aligned} E_1 &= E_{01}cos(\omega_1 t) \\ E_2 &= E_{02}cos(\omega_2 t+\phi) \end{aligned} \quad (13)$$

$$E_1 + E_2 = E_{01}cos(\omega_1 t) + E_{02}cos(\omega_2 t+\phi). \quad (14)$$

Im Spezialfall $\omega_1 = \omega_2$, $\phi = 0$ erhält man aus Formel 14 die gleiche Welle mit der doppelten Amplitude. Wenn die Phase der zweiten Welle bei gleicher Frequenz um den Wert Pi verschoben ist, werden sich beide Wellen hingegen auslöschen. Weitere Rechnungen würden in diesen beiden Fällen entfallen. Die Gleichung 14 wird in Gleichung 12 eingesetzt,

$$\begin{aligned} P = {} & \epsilon_0\chi_1(E_{01}cos(\omega_1 t) + E_{02}cos(\omega_2 t+\phi)) + \frac{1}{2}\epsilon_0\chi_2(E_{01}^2cos^2(\omega_1 t) \\ & + E_{02}^2cos^2(\omega_2 t+\phi) + 2E_{01}cos(\omega_1 t)E_{02}cos(\omega_2 t+\phi)). \end{aligned}$$

Da die Differenzfrequenzerzeugung ein Effekt zweiter Ordnung ist, kann man die Terme erster Ordnung vernachlässigen. Die Formel vereinfacht sich dann zu:

$$P = \frac{1}{2}\epsilon_0\chi_2(E_{01}^2 cos^2(\omega_1 t) + E_{02}^2 cos^2(\omega_2 t + \phi) + 2E_{01}cos(\omega_1 t)E_{02}cos(\omega_2 t + \phi) \tag{15}$$

$$= \epsilon_0\chi_2(\underbrace{\frac{1}{2}E_{01}^2 cos^2(\omega_1 t) + \frac{1}{2}E_{02}^2 cos^2(\omega_2 t + \phi)}_{1} + \underbrace{E_{01}E_{02}cos((\omega_1 - \omega_2)t + \phi)}_{2}$$
$$+\underbrace{E_{01}E_{02}cos((\omega_1 + \omega_2)t + \phi)}_{3}. \tag{16}$$

In Formel 16 sieht man einen Term, in welchem die transmittierten Wellen mit der ursprünglichen Frequenz enthalten sind (Term 1). Die anderen Terme beschreiben Wellenfunktionen mit der Frequenz $\omega_1 - \omega_2$ (Term 2) und $\omega_1 + \omega_2$ (Term 3). Diese Terme beschreiben die Summenfrequenzerzeugung und die Differenzfrequenzerzeugung. Da dies Effekte zweiter Ordnung sind, ist die Intensität dieser Effekte bei geringen Feldstärken klein. Der Grund liegt in der Größe der Suszeptibilität von $\chi_2 \sim 10^{-10}\, cm/V$ [9]. Die Intensität steigt erst bei großen Feldstärken an.

Durch eine Anpassung der Phase des Lichts lässt sich die Differenz- oder Summenfrequenz auswählen. Die Herleitung wird für die Auswahl der Summenfrequenz durchgeführt. Hierzu muss die Amplitude der Summenfrequenz maximal sein, wenn die Amplitude der Differenzfrequenz minimal ist. Somit muss $cos((\omega_1 - \omega_2)t + \phi) = 0$ und $cos((\omega_1 + \omega_2)t + \phi) = 1$ gelten. Da die Kosinusfunktion eine periodische Funktion ist, lassen sich diese Annahmen zu $\frac{(2n+1)}{2}\pi = (\omega_1 - \omega_2)t + \phi$, sowie $n\pi = (\omega_1 + \omega_2)t + \phi$ umschreiben. Diese beiden Funktionen werden nun gleichgesetzt und nach ϕ aufgelöst:

$$\begin{aligned} t &= \frac{\frac{2n+1}{2}\pi - \phi}{\omega_1 - \omega_2} = \frac{n\pi - \phi}{\omega_1 + \omega_2} \\ \phi &= \frac{-\frac{2n+1}{2}\pi + \frac{(\omega_1 - \omega_2)\, n\pi}{\omega_1 + \omega_2}}{1 - \frac{\omega_1 - \omega_2}{\omega_1 + \omega_2}}. \end{aligned} \tag{17}$$

Die Herleitung für die Auswahl der Differenzfrequenz erfolgt analog zur Herleitung von Formel 17. Deshalb wird die Formel 18 ohne Herleitung angegeben [9]:

$$\phi = \frac{-n\pi + \frac{(\omega_1 - \omega_2)\,\frac{2n+1}{2}\pi}{\omega_1 + \omega_2}}{1 - \frac{\omega_1 - \omega_2}{\omega_1 + \omega_2}}. \tag{18}$$

3 Aufbau des Lasersystems

Im Teilchenbeschleuniger der Gruppe PITZ wird eine Photokathode zur Erzeugung der Elektronenpakete genutzt. Hierbei werden mit einem Laserpuls Elektronen aus der Photokathode gelöst. Zur Erzeugung des Pulses wird ein, vom Max-Born-Institut(MBI) entwickeltes, Lasersystem verwendet. Im nächsten Abschnitt wird das Lasersystem zunächst allgemein beschrieben(Abschnitt 3.1). Im Anschluss wird der optional integrierbare Beamshaper (Abschnitt 3.2) und das Optical Sampling System (OSS) detailiert beschrieben (Abschnitt 3.3).

3.1 Beschreibung des MBI-Lasersystems

Das Lasersystem kann, wie in Abb.5 zu sehen, in mehrere Baugruppen unterteilt werden. Die Laserpulse werden im modengekoppelten Oszillator erzeugt. Das

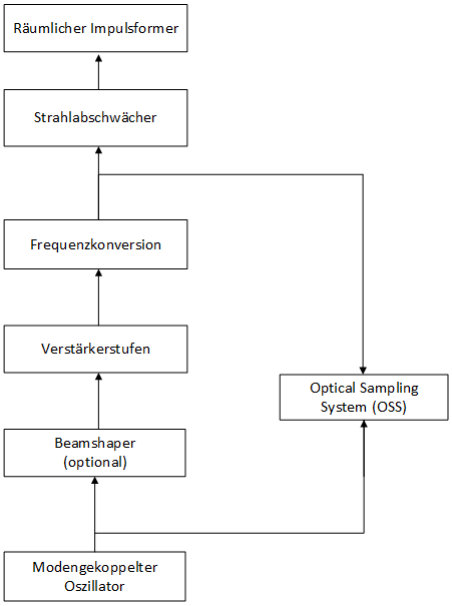


Abbildung 5: Schematische Darstellung des Lasersystems (Weg der Laserpulse durch Pfeile dargestellt)

Kernstück ist ein Ytterbium dotierter Yttrium-Aluminium-Granat-Kristall (Yb-YAG). Das emittierte Laserlicht hat eine Wellenlänge von 1030 nm. Im Oszillator werden die Laserpulse, mit einer Frequenz von 54 MHz und einer Pulslänge von 2 ps, generiert. Die Laserpulse haben die Form einer Gaußverteilung. Die Energie der Laserpulse beträgt ca. 1 nJ je Puls.

Nach dem Oszillator werden die Laserpulse in einen Pulsepicker gelenkt. Ein Teil der Laserpulse wird zur Vermessung des Laserpulses verwendet. Dieser Teil

wird direkt zum OSS übertragen.

Jeder 54. Puls wird vom Pulsepicker zum Beamshaper oder direkt zu den Verstärkerstufen geleitet. So wird die Frequenz auf 1 MHz reduziert. Der Beamshaper kann optional in den Aufbau eingefügt werden. Im Beamshaper werden aus den Pulsen mit gaußförmigem zeitlichem Profil, Laserpulse mit einem anderen definierten zeitlichen Profil geformt. Dies ist meist ein Flat-Top-Profil mit variabler Pulslänge. Der Aufbau und die Funktionsweise des Beamshapers wird in Abschnitt 3.2 beschrieben.

Die Intensität der Laserpulse wird in den Verstärkerstufen erhöht. Die Verstärkung ist notwendig, da die Formung des Intensitätsprofils die Intensität signifikant senkt. Zudem ist für die folgenden Frequenzkonversionen eine hohe Intensität notwendig. Die Verstärkerstufen bestehen aus drei Teilen. Einem regenerativen Verstärker, einem Doppel-Pass-Verstärker und einem Single-Pass-Verstärker. Hier finden nichtlineare Prozesse statt. Im regenerativen Verstärker wird die Energie der Laserpulse auf ca. 1 μJ je Puls erhöht. Nach dem regenerativen Verstärker wird der Laserpuls zum Doppel-Pass-Verstärker geleitet. Zwischen dem Doppel-Pass-Verstärker und dem Single-Pass-Verstärker befindet sich ein weiterer Pulspicker. Hier werden Pulszüge mit einer Frequenz von 10 Hz generiert. Die Pulszüge bestehen aus bis zu 600 Einzelpulsen. Nach dem Single-Pass-Verstärker beträgt die Energie je Puls ca. 10 μJ.

Die Frequenz der Laserpulse wird nun, mittels eines Lithiumtriboratkristalls (LBO) und eines Bariumboratkristalls (BBO), verändert. Hier findet der nichtlineare Prozess der Frequenzverdopplung statt. Dieser Prozess wurde in Abschnitt 2.3.1 beschrieben. Nach zwei Frequenzverdopplungen haben die Laserpulse eine Wellenlänge von 257 nm. Diese Wellenlänge ist notwendig, um Elektronen aus der Photokathode lösen zu können. Die Energie je Puls beträgt nun ca. 2,5 μJ.

Hiernach werden die Laserpulse entweder zum OSS ausgekoppelt oder weiter Richtung Beschleuniger gelenkt. Im OSS wird das zeitliche Intensitätsprofil vermessen. Das OSS wird in Abschnitt 3.3 genauer beschrieben.

Wenn die Pulse zum Beschleuniger auf die Photokathode gelenkt werden, können sie mittels einer $\lambda/2$-Platte und eines Glan-Thompson-Prismas abgeschwächt werden. In der $\lambda/2$-Platte wird die Polaristionsebene des Laserstrahls gedreht. Hiernach wird das Licht im Glan-Thompson-Prisma wieder in der ursprünglichen Schwingungsebene polarisiert und so die Intensität verringert.

Als letzter Schritt, wird das räumliche Profil geformt. Die eintreffenden Pulse haben ein gaußförmiges räumliches Intensitätsprofil. Sie werden durch eine Lochblende geformt. Hier wird der Rand des gaußförmigen, räumlichen Intensitätsprofils abgeschnitten. Es entsteht ein Laserpuls mit einer nahezu gleichmäßigen, räumlichen Intensitätsverteilung. Hiernach treffen die Laserpulse auf die Photokathode.

3.2 Beamshaper

Bei PITZ werden mit dem Beamshaper hauptsächlich Flat-Top-Profile erzeugt. Als Flat-Top-Profil wird ein Intensitätsprofil bezeichnet, welches eine konstante Intensität über die gesamte Pulsdauer aufweist. Der Bereich mit konstanter Intensität wird im Weiteren Flat-Top-Bereich genannt. Ein solches Intensitätsprofil ist nach dem Ausgangspolarisator in Abb.6 zu sehen. Der Aufbau des Beamshapers ist ebenfalls in Abb.6 zusehen. Er besteht aus 13 doppelbrechen-

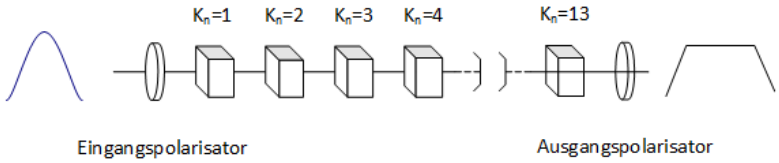


Abbildung 6: Schematische Darstellung des Beamshapers bei PITZ mit gaußförmigen Eingangslaserpuls und einem Laserpuls mit Flat-Top-Profil am Ausgang

den Kristallen. Die Kristalle bestehen aus Yttriumorthovanadat (YVO_4). Die Sellmeier-Gleichungen für die optischen Achsen dieses Kristalls lauten [11]:

$$n_o^2(\lambda) = 1 + \frac{2.7665\lambda^2}{\lambda^2 - 0.026884}, \tag{19}$$

$$n_{ao}^2(\lambda) = 1 + \frac{3.5930\lambda^2}{\lambda^2 - 0.032103}. \tag{20}$$

Die Kristalle sind in einer Reihe angeordnet. Sie werden durch Schrittmotoren gedreht. Diese können die Kristalle um maximal $\pm$ 1 Grad drehen. Die Temperatursteuerung der Kristalle erfolgt mit Peltier-Elementen. Dieser Aufbau ermöglicht die Einstellung der Länge und der Form des Intensitätsprofils. Die Länge und die Anzahl der Kristalle bestimmen die Länge des geformten Pulses. Durch die Kristallwinkel wird das Intensitätsprofil eingestellt. Der Flat-Top-Bereich wird durch die Regelung der Temperatur geglättet.

Der einfallende Laserpuls wird im doppelbrechenden Kristall auf die beiden optischen Achsen projiziert. Durch die unterschiedlichen Brechungsindizes der optischen Achsen und die Länge des Kristalls entsteht ein Laufzeitunterschied und die Kopien werden zeitlich separiert. Dieser Laufzeitunterschied bestimmt die Länge des Intensitätsprofils am Ende des Beamshapers. Aus den Sellmeier-Gleichungen (Formeln 19, 20) lassen sich die Brechungsindizes der Kristalle für die genutzte Wellenlänge berechnen. Die Brechungsindizes für eine Wellenlänge von 1030 nm lauten $n_o(1030\,\text{nm})$=1.959 und $n_{ao}(1030\,\text{nm})$=2.169 [12]. Der Brechungsindex von YVO_4 ist ebenfalls von der Temperatur abhängig. Die Abhängigkeit wird durch den thermo-optischen Koeffizienten beschrieben (siehe Tabelle 2). Der thermo-optische Koeffizient liegt im Bereich von ca. $10^{-6}\,\frac{1}{\text{K}}$ und

Tabelle 2: Kennzahlen von YVO_4

	ordentliche Achse	außerordentliche Achse
thermo-optischer Koeffizient(1030nm) $\frac{dn}{dT} = \left[\frac{10^{-6}}{\mathrm{K}}\right]$	3.2[13]-≈ 16[12]	0.6[13]- ≈ 8,7[12]
Ausdehnungskoeffizient $\alpha = \left[\frac{10^{-6}}{\mathrm{K}}\right]$[12]	1,76	8,24

ist um sechs Größenordnungen kleiner als der Brechungsindex. Der Einfluss auf die Zeitverzögerung ist deshalb vernachlässigbar gering. Aus den Kennzahlen für YVO_4 lässt sich nun ein Laufzeitunterschied pro Weglänge, von der ordentlichen zur außerordentlichen Welle, errechnen [5]:

$$\frac{\Delta t}{d} = \frac{n_o(1030\,\mathrm{nm}) - n_{ao}(1030\,\mathrm{nm})}{c} = 0.7\,\frac{\mathrm{ps}}{mm}. \tag{21}$$

In den Kristallen wird die Anzahl der einfallenden Pulse jeweils verdoppelt. Die Anzahl der Laserpulse m_{ges} am Ende der Reihe von K_{ges} Kristallen kann mit der Formel $m_{ges} = 2^{K_{ges}}$ berechnet werden. In jedem Kristall wird ein Teil der Pulse um die, durch die Länge der Kristalle definierte, Zeit verzögert. Die Gesamtlänge des Flat-Top-Bereichs t_{Puls} kann wie folgt geschätzt werden:

$$t_{Puls} = d \cdot 0.7\,\frac{\mathrm{ps}}{mm} \cdot K_{ges}. \tag{22}$$

Hierbei ist jedoch zu beachten, dass die Kopien einzeln sichtbar sind, wenn die zeitliche Verzögerung durch die Kristalle wesentlich größer ist als die Breite des Laserpulses.

Durch die Kristallwinkel werden die Intensitäten der Kopien eingestellt. So ist es möglich, die Intensitäten der Kopien so zu wählen, dass man am Ende einen neuen Gesamtpuls mit einem variablen Profil erhält. In Abschnitt 2.2.1 wurde erklärt, wie sich durch Änderung des Drehwinkels eines doppelbrechenden Kristalls die Intensitätsverteilung des Pulses nach dem Kristall beeinflussen lässt. Der Berechnungsalgorithmus lässt sich auf eine Reihe von doppelbrechenden Kristallen übertragen, indem er auf jede Kopie angewendet wird. Der Flat-Top-Bereich des Pulses besitzt nach seiner Erzeugung noch Unebenheiten. Da die Laserpulse elektromagnetische Wellen sind, interferieren die Kopien miteinander. Durch die Veränderung der Phasenverschiebung ist es möglich, das Profil zu glätten. Für die Berechnung des Intensitätsprofils darf der thermo-optische Koeffizient, im Gegensatz zur Berechnung der zeitlichen Verzögerung, deshalb nicht vernachlässigt werden.

Die Feineinstellung der Phasenverschiebung wird durch die Temperatur der Kristalle geregelt. Die temperaturabhängige Änderung der Phasenverschiebung lässt sich theoretisch berechnen. Hierzu sind der thermo-optische Koeffizient $\frac{\mathrm{dn}}{dT}$ und der Ausdehnungskoeffizient α notwendig. Die Werte für YVO_4-Kristalle sind in Tabelle 2 zu finden. Die Phasenverschiebung von ordentlicher zu außerordentlicher Achse berechnet sich nach der Formel:

$$\phi = 2\pi \cdot d(T) \frac{n_o(T,\lambda) - n_{ao}(T,\lambda)}{\lambda} \tag{23}$$

$$\phi = 2\pi \cdot d(T_0) \cdot \alpha \left(T - T_0\right) \cdot \frac{n_o(\lambda) + (T - T_0)\,\frac{dn_o(\lambda)}{dT} - n_{ao}(\lambda) + (T - T_0)\,\frac{dn_{ao}(\lambda)}{dT}}{\lambda}. \tag{24}$$

Da der thermo-optische Koeffizient vom Kristall abhängt, ist die Berechnung der Phasenverschiebung für die Kristalle im realen Beamshaper nicht sinnvoll. Die durch die Temperaturregelung erzeugte Phasenverschiebung muss deshalb, für jeden im Beamshaper befindlichen Kristall gemessen werden.

3.3 Optical Sampling System

Für den definierten Betrieb des Beschleunigers ist es notwendig, das geformte Intensitätsprofil des Laserpulses zu kennen. So können Elektronenpakete mit definierten Eigenschaften erzeugt werden. Bei PITZ wird zur Messung des Intensitätsprofils ein Optical Sampling System (OSS) verwendet.

Wie bereits in Abschnitt 3.1 erwähnt, werden im modengekoppelten Oszillator Laserpulse mit einer Frequenz von 54 MHz erzeugt. Hiervon wird jeder 54. Laserpuls zur Generierung der Elektronenpakete geformt. Die restlichen Laserpulse werden verwendet, um das Intensitätsprofil der geformten Pulse zu messen. Sie werden nicht geformt. Im Weiteren werden die Laserpulse, die zur Bestimmung des Intensitätsprofils verwendet werden, als Abtastpulse bezeichnet. Die geformten Laserpulse haben im OSS eine Wellenlänge von 257 nm. Die Abtastpulse haben eine Wellenlänge von 1030 nm.

Der schematische Aufbau des OSS ist in Abb.7 zu sehen. Die Abtastpulse durchlaufen zunächst einen regenerativen Verstärker. Durch die Verstärkung der Abtastpulse kann, im nichtlinearen Kristall, die Differenzfrequenz mit einer höheren Intensität erzeugt und ein größeres Signal bei der Messung erzielt werden. Die Abtastpulse werden nun auf einen Spiegel geleitet. Der Spiegel wird mit einer Spule, senkrecht zur Spiegelebene, in Schwingung versetzt. Durch die Schwingung wird die Weglänge, welche die Abtastpulse zurücklegen, variiert. Die Laufzeit der Abtastpulse wird so periodisch geändert. Die Abtastpulse (rot Strich-Punkt) werden zusammen mit dem geformten Puls (schwarz durchgehend) in einen nichtlinearen Kristall gelenkt (siehe Abb.8). Hier überlagern sie sich. Dabei wird die Differenzfrequenz gebildet (grün gestrichelt). Der Prozess

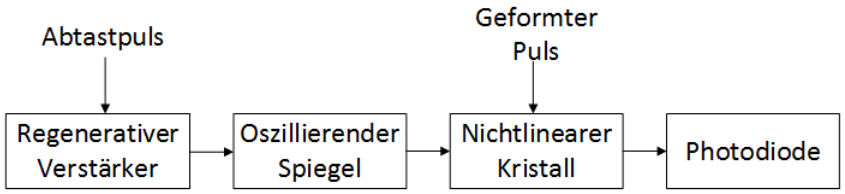


Abbildung 7: schematischer Aufbau des OSS. DerWeg der Laserpulse ist durch Pfeile dargestellt

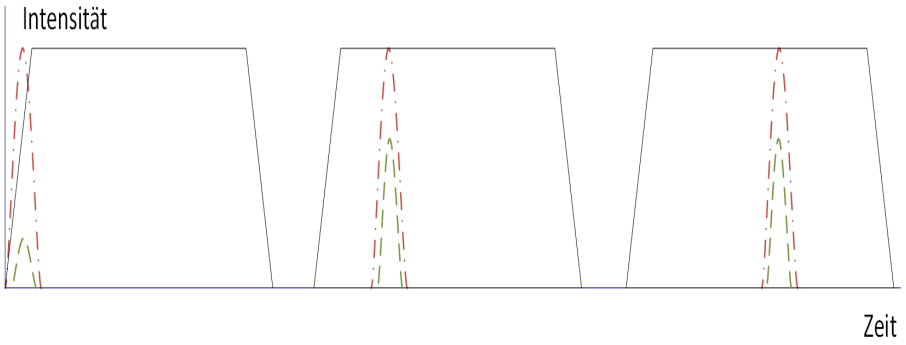


Abbildung 8: Überlagerung von Abtastpuls (rot, Strich-Punkt) mit dem geformten Laserpuls (schwarz, durchgehend) bei verschiedenen Stellungen des oszillierenden Spiegels und der daraus resultierende Differenzfrequenzpuls (grün, gestrichelt)

der Differenzfrequenzerzeugung wurde in Abschnitt 2.3.2 erklärt. Da die Laufzeit der Abtastpulse durch den Spiegel variiert wird, überlagern sich die Abtastpulse zu verschiedenen Zeitpunkten mit dem geformten Laserpuls. Die Intensität der erzeugten Differenzfrequenzpulse wird mit einer Photodiode gemessen. Die gesamte Pulsform wird aus den Messungen im Computer rekonstruiert. Der Abtastpuls hat eine Breite von ca. 2 ps und der geformte Laserpuls für die Photokathode eine Breite von ca. 22 ps. Der Signalpuls hätte durch die Faltung der Funktionen die Breite $\sqrt{2}\cdot 2\,\mathrm{ps}=2{,}8\mathrm{ps}$. Die zeitliche Auflösung der Messung wäre dadurch sehr gering. Es ist jedoch zu beachten, dass der Signalpuls für die Messung in einem nichtlinearen Kristall erzeugt wird. Wie in Abschnitt 2.3.2 bereits erwähnt, ist die Intensität des Lichtes für nichtlineare Prozesse von großer Bedeutung. Somit sind die Flanken des Abtastpulses für die Messung nicht von Bedeutung, da hier die Intensität zu gering ist. Der gemessene Bereich ist deshalb kleiner als 2 ps. Die Zeitauflösung der Messung steigt dadurch. Die hier durchgeführten Messungen zeigen das Intensitätsprofil des geformten Pulses im ultravioletten Wellenlängenbereich. Dieses weicht vom simulierten Profil ab, da hier das Intensitätsprofil im infraroten Bereich errechnet wird.

4 Simulationsprogramm PulSi

Im Rahmen dieser Arbeit wurde ein Programm zur Simulation von Intensitätsprofilen erstellt. Die Formung der Profile findet mit Beamshapern, der in Abschnitt 3.2, Abb.6 beschriebenen Bauweise, statt. Das Programm wird PulSi genannt. Der Name PulSi steht für *Pul*seshaper *Si*mulation. Durch PulSi können die Auswirkungen von Veränderungen am Beamshaper auf die Pulsform simuliert werden. Somit kann der Nutzer testen, ob die Änderung von Einstellungen am Beamshaper die von ihm gewünschte Veränderung der Pulsform verursacht. Durch die Simulation kann dies während des laufenden Betriebs stattfinden, ohne den Betrieb des Beschleunigers zu beeinflussen.

PulSi ist in der Programmiersprache C++ geschrieben. Die Programmiersprache wurde gewählt, da sie die Möglichkeit zur Erstellung der grafischen Oberfläche bietet. Die Benutzeroberfläche wurde mit der Software QT Creator Version 2.4.1 erstellt. Für das Darstellen der Diagramme wurde die Bibliothek QCustomPlot verwendet. Das Programm wurde unter Ubuntu 14.04 getestet.

Das Programm enthält bereits Strukturen, welche, bei einer Weiterentwicklung von PulSi, die direkte Steuerung des Beamshapers durch PulSi ermöglicht. Diese Strukturen führen in der aktuellen Version noch keine Aktionen aus.

In den nächsten Abschnitten wird zunächst das grafische Benutzerinterface erklärt. Danach wird die Implementierung des Berechnungsalgorithmus erklärt und auf seine Genauigkeit eingegangen. Zudem wird der Suchalgorithmus zum Finden von definierten Intensitätsprofilen beschrieben.

4.1 Benutzeroberfläche

Für PulSi wurde eine grafische Bedienungsoberfläche erstellt. Falls PulSi über die Konsole gestartet wurde, können hier während des automatischen Suchlaufs zusätzliche Informationen über dessen Status abgelesen werden. Die grafische Oberfläche besteht aus mehreren Interfacefenstern, sodass die Oberfläche übersichtlich bleibt. Die Einteilung nach Fenstern entspricht verschiedenen Programmfunktionen.

Im Hauptfenster können die Eigenschaften des Beamshaper definiert und manipuliert werden. In den "Erweiterten Optionen" können Einstellungen für den automatischen Suchlauf, sowie spezielle Einstellungen des Beamshapers, definiert werden. Es gibt außerdem Fenster zum Laden und Speichern von Daten sowie zur Darstellung des Status des automatischen Suchlaufs. Die einzelnen Fenster werden in den nächsten Abschnitten genauer beschrieben.

4.1.1 Hauptfenster

In diesem Abschnitt wird das Hauptfenster von PulSi beschrieben. Hier werden die Eigenschaften des Beamshapers definiert und manipuliert. Zudem wird hier das Intensitätsprofil dargestellt und der automatische Suchlauf gestartet.

Im Hauptfenster (Abb.9) von PulSi werden in Bereich 1 die Eigenschaften des Beamshapers definiert. Die Einstellungen der Winkel und der Phasenverschiebungen der Kristalle des Beamshapers sind in Bereich 2 zu sehen. Diese können direkt dort oder mit den Schaltflächen im Bereich 3 verändert werden. In Bereich 4 wird das Intensitätsprofil des Beamshapers mit den aktuellen Einstellungen in einem Diagramm dargestellt. In Bereich 5 sind Optionen zur Manipulation des Diagramms zu sehen. In Bereich 6 können verschiedene Eigenschaften, wie z.B. die Breite des Intensitätsprofils, bestimmt werden. Im Hauptfenster wird zudem der automatische Suchlauf gestartet. Zusätzlich sind Strukturen implementiert, welche in späteren Versionen das direkte Steuern des Beamshapers mit PulSi ermöglichen sollen. Die Funktionen der einzelnen Bereiche werden im folgenden näher erklärt.

Für die Nutzung von PulSi muss zunächst ein Beamshaper erstellt werden. Dies wird in Bereich 1 durchgeführt (Abb.10). Hier werden die Eigenschaften des Beamshapers und des eintreffenden Laserpulses definiert. Dazu gehören die Anzahl der Kristalle im Beamshaper (Feld "Number of crystals"), die durch sie verursachte zeitliche Verzögerung (Feld "Time delay of ordinary to extraordinary wave") und die Phasenverschiebung (Feld "Phase shift [Pi]"). Die Phasenverschiebung wird bei der Erstellung des Beamshapers für alle Kristalle mit dem gleichen Wert definiert. Sie kann nach der Erstellung des Beamshapers geändert werden. Außerdem muss die Halbwertsbreite des gaußförmigen Laserpulses am Eingang des Beamshapers definiert werden (Feld "FWHM gaussian pulse [ps]").

Zum Betrieb des automatischen Suchlaufs muss die maximale Abweichung vom vorgegebenen Zielprofil zum berechnetem Profil definiert werden. Diese Abweichung wird im Feld "max. deviation to the pulse shape [%]" definiert. Die definierten Werte werden durch Drücken der Schaltfläche "apply" in Bereich 1 unten links bestätigt. Soll nur die Genauigkeit des Suchlaufs geändert werden, wird die Änderung durch Drücken der Schaltfläche "change accuracy" bestätigt. Die definierten Eigenschaften des Beamshapers und des Laserpulses bleiben dann unverändert. Durch die Bestätigung der Eingaben mittels einer Schaltfläche, wird ein versehentliches Ändern der definierten Werte verhindert.

Nach Drücken der Schaltfläche "apply" werden intern die Winkel der Kristalle und des Polarisators am Ausgang des Beamshapers definiert. Alle Winkel werden relativ zum Winkel des Eingangspolarisators (0°) angegeben. Die Winkelverteilung der Kristalle folgt der Formel

$$\psi_n = \frac{45^\circ}{K_{ges}}(2K_n - 1) \tag{25}$$

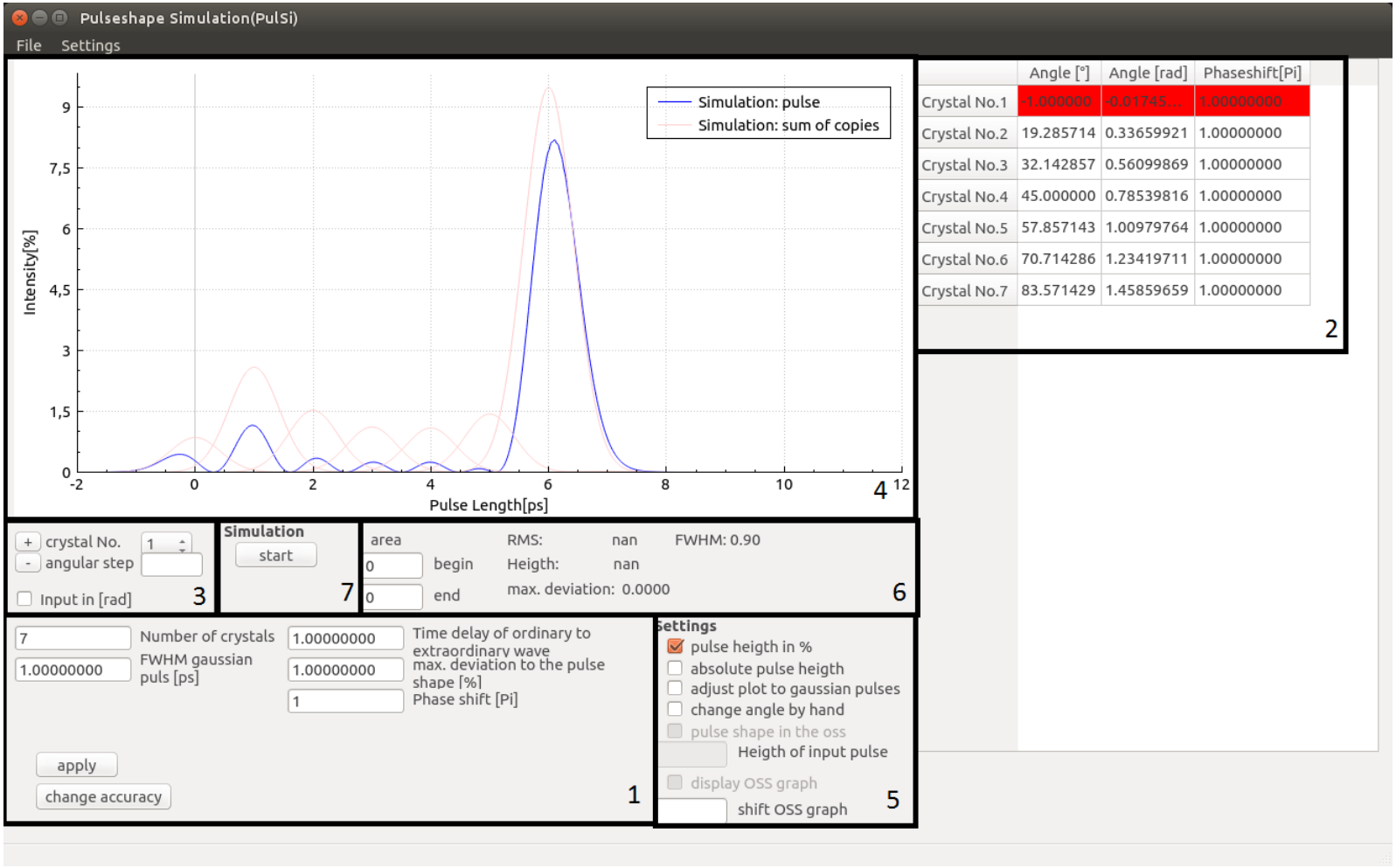


Abbildung 9: Hauptfenster des Programms PulSi mit der Unterteilung in 7 Bereiche (schwarze Rahmen)

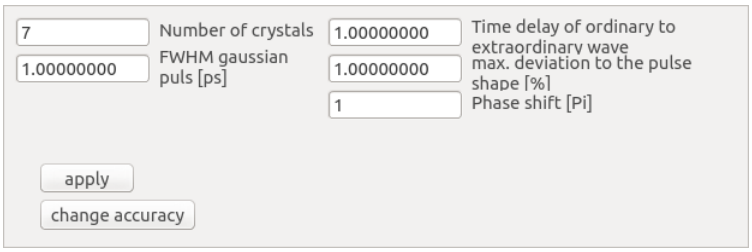


Abbildung 10: Interface zur Definition der Eigenschaften des Beamshapers und des einfallenden Laserpulses (Abb.9 Bereich 1)

[5]. Hierbei ist ψ_n der Winkel des Kristalls K_n und K_{ges} die Kristallanzahl im Beamshaper. Durch diese Winkelverteilung erhält man ein relativ gleichmäßiges Intensitätsprofil im gesamten Laserpuls. Der Winkel des Ausgangspolarisators wird mit 0° definiert. In den erweiterten Optionen (settings→advanced options) kann der Drehwinkel des Ausgangspolarisators geändert werden (siehe Abschnitt 4.1.2).

Die Winkel der Kristalle und deren Phasenverschiebung werden in einer Tabelle ausgegeben. Die Tabelle ist in Bereich 2 zu sehen (Abb.11). Wenn die Einstellungen der Kristalle des Beamshapers manuell verändert werden sollen, können die Werte in dieser Tabelle verändert werden. Hierzu muss die entsprechende Zelle

	Angle [°]	Angle [rad]	Phaseshift[Pi]
Crystal No.1	-1.000000	-0.01745...	1.00000000
Crystal No.2	19.285714	0.33659921	1.00000000
Crystal No.3	32.142857	0.56099869	1.00000000
Crystal No.4	45.000000	0.78539816	1.00000000
Crystal No.5	57.857143	1.00979764	1.00000000
Crystal No.6	70.714286	1.23419711	1.00000000
Crystal No.7	83.571429	1.45859659	1.00000000

Abbildung 11: Tabelle mit den Kristallwinkeln und Phasenverschiebungen für das zu simulierende Intensitätsprofil (Abb.9 Bereich 2), Zeilen werden rot dargestellt falls Eingabe des Winkels außerhalb der definierten Winkelgrenzen der Kristalle des Beamshapers liegt

ausgewählt und geändert werden. Der neue Wert der Zelle wird durch Drücken der Enter-Taste bestätigt. Im realen Beamshaper können die Kristalle nur in einem bestimmten Bereich gedreht werden. Für die Drehwinkel der Kristalle können deshalb in den erweiterten Einstellungen Grenzen festgelegt werden. Sollten diese Grenzen durch manuelle Eingaben überschritten werden, wird in der Tabelle die Zeile des entsprechenden Kristalls rot unterlegt dargestellt. Dies ist exemplarisch in der Zeile “crystall No.1” zu sehen.

Die Winkel der Kristalle können auch schrittweise geändert werden. Hierzu wird in Bereich 3 (Abb.12) die Schrittweite (Feld “angular step”) und der zu drehende Kristall (Feld “crystall No.”) festgelegt. Die Schrittweite kann in Bogenmaß

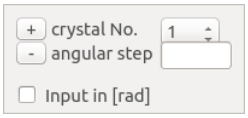


Abbildung 12: Interface zur schrittweisen Änderung der Kristallwinkel (Abb.9 Bereich 3)

eingegeben werden. Hierzu muss die Option “input in [rad]” aktiviert werden. Ist diese Option nicht aktiviert, erfolgt die Eingabe in Grad. Durch Drücken der Schaltflächen “+” und “-” in Bereich 3 wird der Winkel des ausgewählten Kristalls um die festgelegte Schrittweite verändert.

Das Intensitätsprofil des geformten Laserpulses, als Funktion der Pulslänge, wird in Bereich 4 dargestellt (Abb.13). Die blaue Kurve beschreibt das Intensitätspro-

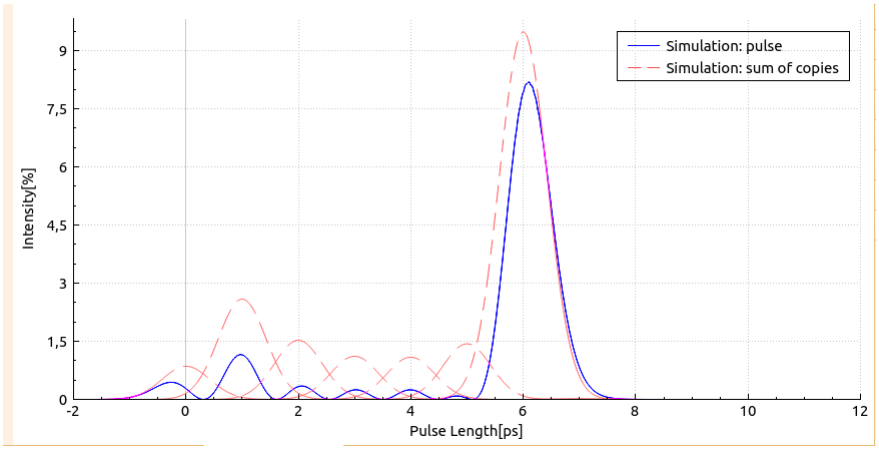


Abbildung 13: Ausgabe des simulierten Intensitätsprofils im Hauptfenster von PulSi (Abb.9 Bereich 4)

fil des geformten Laserpulses am Ende des Beamshapers. Die orangenen gestrichelten Gaußkurven stellen die Summation der Kopien des Eingangslaserpulses an dieser Position dar. Die Phasenverschiebung der Kopien wird bei der Berechnung der Gaußkurven nicht berücksichtigt. In der Realität entsteht die zeitliche Verschiebung der Kopien durch eine Phasenverschiebung. Da in der Berechnung jedoch die Kopie durch eine Faltung von elektromagnetischer Welle und Gaußfunktion beschrieben wird, kann die Position der Gaußkurve unabhängig von der Phasenverschiebung der Welle betrachtet werden. Die Berechnung der Kurven wird in Abschnitt 4.2 beschrieben. Der Nutzer kann die Darstellung im Diagramm verändern.

In Bereich 5 (Abb.14) sind verschiedene Optionen verfügbar, mit welchen das in Abb. 13 dargestellte Diagramm angepasst werden kann. Es ist möglich, die

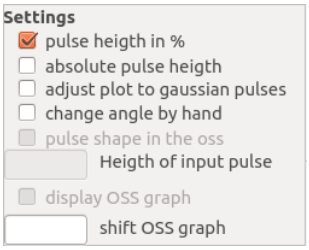
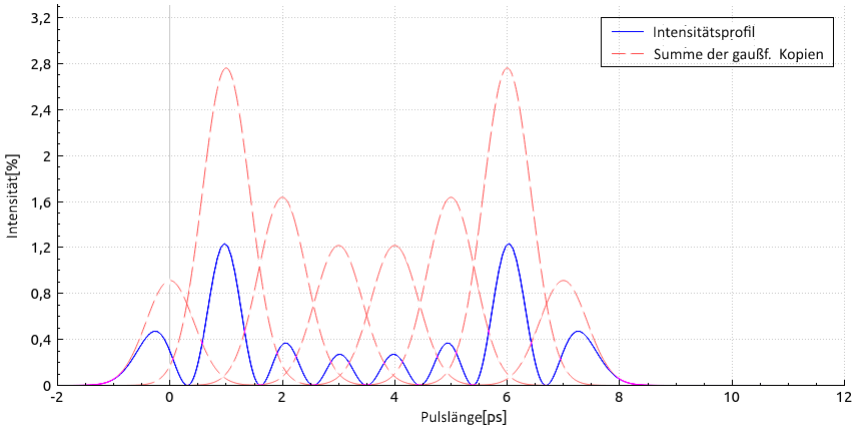


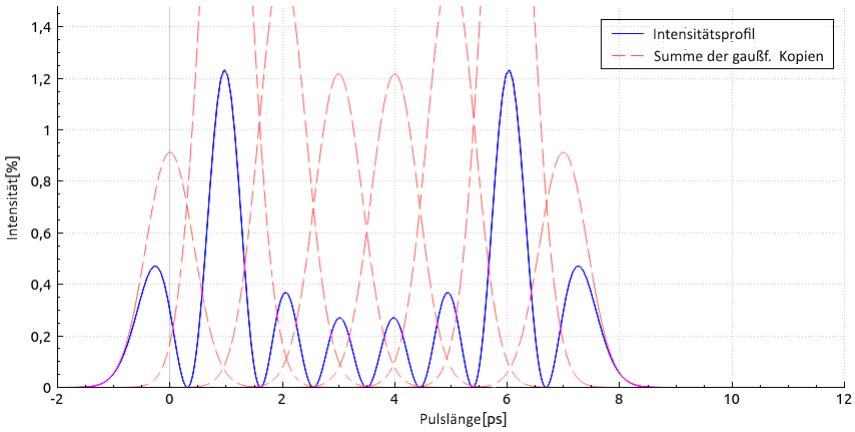
Abbildung 14: Optionen zur Darstellung des Intensitätsprofils und der OSS-Messungen (Abb.9 Bereich 5). Nicht aktivierte Optionen sind grau dargestellt.

Skalierung der y-Achse zu ändern. Man kann mit der Option "pulse heigth in %" die Höhe in Prozent vom Eingangspuls anzeigen lassen. Mit der Option "absolute pulse heigth" wird die absolute Höhe des Pulses angezeigt. Wird keine der beiden Optionen ausgewählt, wird die y-Achse ohne Skalierung generiert. In der aktuellen Version des Programms ist die Höhe des Laserpulses am Eingang des Beamshapers auf 1 normiert. Mit der Option "adjust plot to Gaussian pulses" wird bestimmt, ob der Graph an der höchsten Gaußkurve (Abb.15a) oder der blauen Kurve (Abb.15b) ausgerichtet wird. Wenn die Option aktiviert ist, wird das Diagramm an der höchsten Gaußkurve ausgerichtet (Abb.15a). Mit der Option "change angle by hand" werden nur ausgewählte Punkte der blauen Kurve berechnet. Sie befinden sich an der zeitlichen Position der Maxima der jeweiligen Gaußkurve. Durch die Darstellung weniger Punkte wird Rechenzeit zur Berechnung des Profils gespart. Die Punkte entsprechen den Testpunkten des automatischen Suchlaufs. Der automatische Suchlauf und die Auswahl der Testpunkte wird in Abschnitt 4.3 erklärt. Diese Option sollte gewählt werden, wenn der Beamshaper aus vielen Kristallen besteht und manuell ein Profil erstellt werden soll. Hierbei ist zu erwähnen, dass die Phasenverschiebung der Kristalle nicht angepasst werden sollte. Um ihren vollständigen Einfluss sehen zu können, müssten mehr Punkte berechnet werden.

Der Nutzer kann sich zudem, die im OSS gemessenen, Kurven anzeigen lassen. Sobald eine Kurve geladen wurde, kann sie mit der Option "display OSS graph" im Diagramm angezeigt werden. Die Option ist nur auswählbar, wenn OSS-Daten geladen wurden. Im Feld "shift OSS graph" kann die Messkurve des OSS zeitlich verschoben werden. Der Nutzer kann so das gemessene Profil über das simulierte Profil schieben. Dies ist notwendig, da die Zeitindices der Simulation und des gemessenen Profils zueinander verschoben sind. Die Option "pulse shape in the OSS" ist momentan nicht verfügbar. Sie wurde implementiert, da PulSi im nächsten Entwicklungsschritt das Intensitätsprofil im OSS simulieren soll. Dann können die Ergebnisse der Simulation, mit den gemessenen Profilen, verglichen werden. Dadurch können ebenfalls die Feineinstellungen mit PulSi durchgeführt werden. Hierzu ist es nötig, die Verstärkung und die Intensitätsabhängigkeit der



(a) Diagramm am höchsten Gaußpuls ausgerichtet



(b) Diagramm am Intensitätsprofil ausgerichtet

Abbildung 15: Verschiedene Möglichkeiten der Ausrichtung des Diagramms in PulSi

Frequenzkonversion des Laserpulses zu messen. Die theoretische Berechnung ist aufgrund des Messfehlers der Kennzahlen der optischen Bauelemente, in welchen die nichtlinearen Prozesse stattfinden, zu ungenau (siehe Abschnitt 2.3.1). Die hierzu notwendigen Messungen werden zu einem späteren Zeitpunkt durchgeführt.

Zur Charakterisierung des erstellten Intensitätsprofils ist es notwendig, verschiedene Eigenschaften der Kurve zu kennen. In Bereich 6 (Abb.16) wird die Halbwertsbreite des Intensitätsprofils ("FWHM") ausgegeben. Hierzu wird die maximale Höhe des Profils gesucht. Es wird dann der erste Punkt gesucht, dessen Höhe größer ist als die Hälfte der maximalen Höhe sowie bei welchem Punkt

sie wieder unter die Hälfte des Maximalwertes fällt. Die zeitliche Differenz der Werte wird dann ausgegeben. In einem vom Nutzer bestimmten Bereich wird

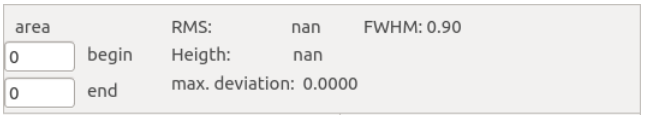


Abbildung 16: Kenngrößen des Intensitätprofils (Abb.9 Bereich 6)

die durchschnittliche Höhe ("heigth"), die maximale Abweichung von ihr ("max. deviation") und der RMS Wert ("RMS") berechnet. Der Nutzer definiert hierzu den Messbereich in den Feldern "begin" und "end" in Bereich 6. Die im Feld "begin" eingetragene Zahl muss kleiner als die Zahl im Feld "end" sein, damit Ergebnisse ausgegeben werden.

Im Hauptfenster wird zudem der automatische Suchlauf gestartet. Dies wird mit der Schaltfläche "start" in Bereich 7 (Abb.17) durchgeführt. Für den au-

Abbildung 17: Schaltfläche zum Starten des automatische Suchlaufs (Bereich 7 Abb.9)

tomatischen Suchlauf können verschiedene Einstellungen im Fenster "erweiterte Einstellungen" vorgenommen werden. Diese Einstellungen werden im Abschnitt 4.1.2 erklärt.

4.1.2 Erweiterte Einstellungen

In den erweiterten Einstellungen (Abb.18) können Einstellungen des automatischen Suchlaufs angepasst werden. Außerdem kann hier der Aufbau des Beamshapers durch das Einfügen von Polarisatoren verändert werden. Es ist außerdem möglich, die Methode zur Definition der Eigenschaften des Beamshapers zu ändern. Zudem gibt es eine Option zur Auswahl verschiedener Berechnungsmethoden. Im folgenden Abschnitt werden die Optionen und Eingabemöglichkeiten genauer beschrieben.

In Bereich 1 können Einstellungen an den Polarisatoren des Beamshapers durchgeführt werden. Es wurde in Bereich 1 die Option "Polarizer after each crystal" implementiert. Hiermit wird nach jedem Kristall ein Polarisator, mit dem im Feld "Angle of the last Polarizer[°]" definierten Winkel, platziert. Zur Simulation eines Intensitätsprofils des Beamshaper des MBI-Lasersystems muss die Option "Polarizer at the end" in Bereich 1 aktiviert sein. Es können durch diese beiden

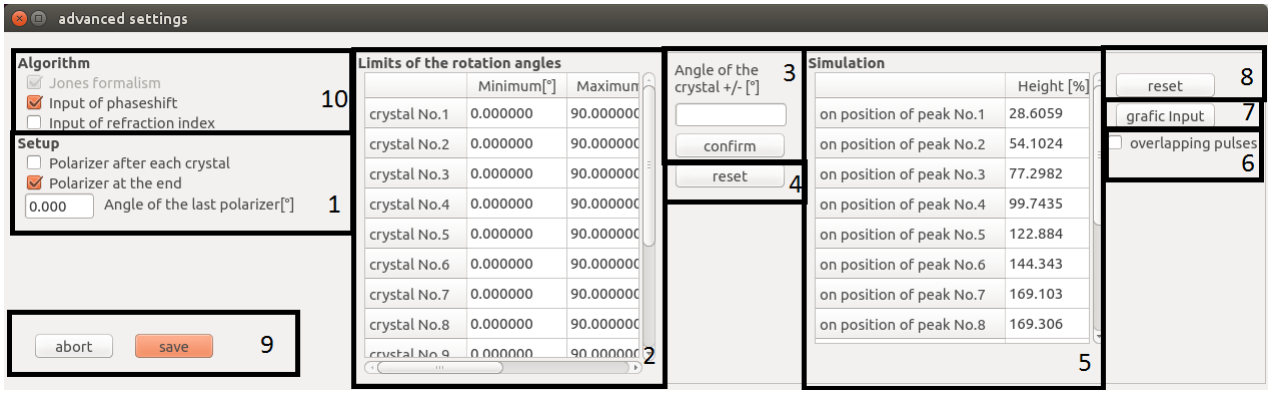


Abbildung 18: Erweiterte Einstellungen in PulSi, entsprechend der Funktionen unterteilt in 10 Bereiche

Optionen verschiedene Bauformen des Beamshapers simuliert werden. In Bereich 1 kann im Feld “Angle of the last polarizer[°]” der Winkel des Polarisators am Ausgang des Beamshapers definiert werden.

Bei einem realen Beamshaper kann nicht automatisch von einem Drehbereich der Kristalle von 360° ausgegangen werden. Dies muss bei der Simulation berücksichtigt werden. In der Tabelle in Abb.19 können die Grenzen der Drehwinkel der Kristalle definiert werden. Der automatische Suchlauf sucht dann innerhalb

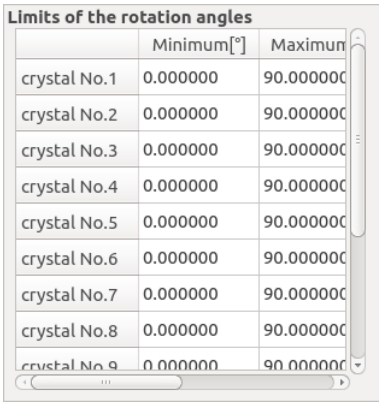


Abbildung 19: Tabelle zur Festlegung der minimalen und maximalen Drehwinkel der Kristalle im Beamshaper (Abb.18 Bereich 2)

dieser Grenzen nach einer Kristallkonfiguration, welche das geforderte Intensitätsprofil erzeugt. Wenn die Grenzen durch manuelle Eingaben überschritten werden, werden die entsprechenden Drehwinkel, in der Tabelle (siehe Abschnitt 4.1.1 Abb.9 Bereich 7) rot unterlegt, dargestellt. Die Grenzen sind standardmäßig auf eine untere Grenze von 0° und eine obere Grenze von 90° festgelegt. Die Winkel können verändert werden, indem der Nutzer einen anderen Wert in die Zelle der Tabelle schreibt.

Wenn der Nutzer möchte, dass alle Kristalle, ausgehend von ihrer aktuellen Position, nur um einen bestimmten Wert in beide Richtungen gedreht werden können, kann er Bereich 3 (Abb.20) nutzen. Im Eingabefeld in Bereich 3 kann der

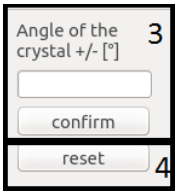


Abbildung 20: Optionen zur Bearbeitung der Tabelle mit den maximalen Drehwinkeln der Kristalle (Abb.18 Bereiche 3 und 4)

Nutzer einen Wert definieren z.B. ± 4°. Mit der Schaltfläche “confirm” bestätigt

er die Eingabe. PulSi setzt dann die Grenzen auf $\psi \pm 4°$. Mit der Schaltfläche "reset" (Abb.20, Bereich 4) werden die Werte, auf die Werte bei Öffnen, der erweiterten Optionen zurückgesetzt.

In den erweiterten Optionen wird die zu suchende Pulsform definiert. Dies kann in der Tabelle aus Abb.21 durchgeführt werden. Hier wird die Pulshöhe am

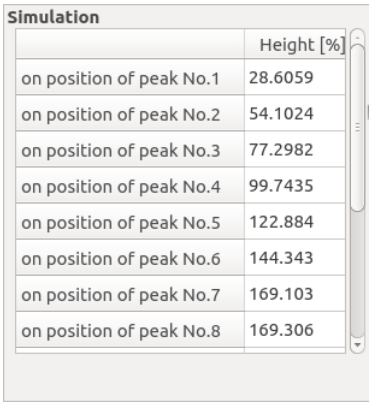


Abbildung 21: Tabelle zur Definition der vom automatischen Suchlauf zu suchenden Pulsform (Abb.18 Bereich 5)

Messpunkt in Prozent von der durchschnittlichen Pulshöhe des gesamten Pulses angegeben. PulSi überprüft automatisch die eingegebenen Pulshöhen. Es wird überprüft, dass die durchschnittliche Pulshöhe 100 % ist. Ist diese Bedingung nicht erfüllt, wird die Eingabe nicht akzeptiert. Die Überprüfung der Eingabe kann, durch Aktivieren der Option "overlapping pulses" (Abb.22 Bereich 6), deaktiviert werden. Dies kann jedoch dazu führen, dass das Intensitätprofil im

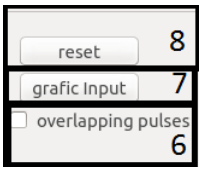


Abbildung 22: Optionen und Schaltflächen zur Einstellung der zu suchenden Pulsform (Abb.18 Bereiche 8, 9, 10)

automatischen Suchlauf nicht gefunden werden kann. Der Suchlauf fällt dann in eine Endlosschleife.

Deshalb gibt es die Möglichkeit, das Intensitätsprofil in einer Grafik einzustellen. Die bereits erwähnte grafische Eingabe der Pulsform ist die empfohlene Eingabeform. Hierzu muss in den erweiterten Optionen die Schaltfläche "grafic input" (Abb.22 Bereich 7) betätigt werden. Es öffnet sich das in Abb.23 gezeigte Fenster. Im Bereich A (Abb.23) werden die Messpunkte dargestellt. Hier kann

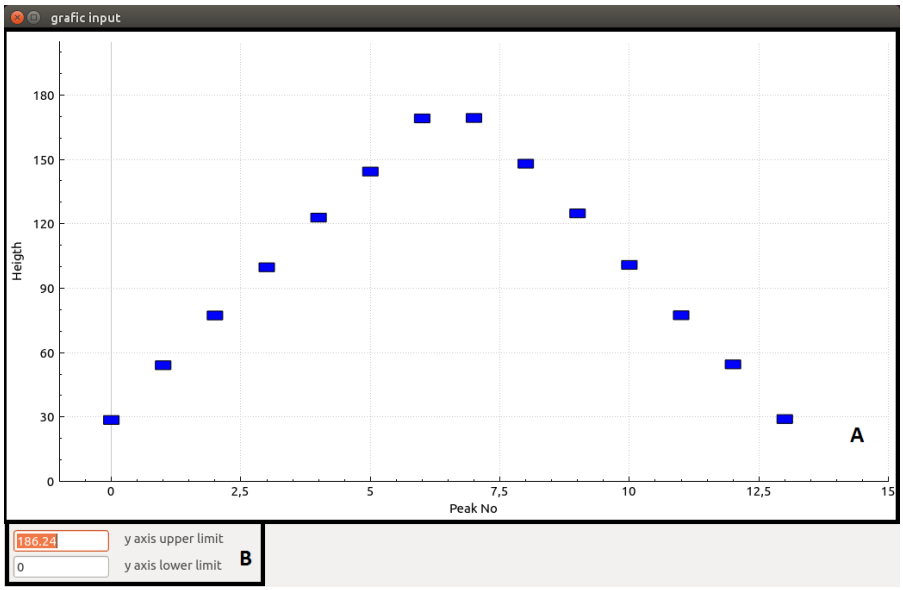


Abbildung 23: Grafisches Eingabeinterface des geforderten Intensitätsprofils in PulSi

einer der Messpunkte durch anklicken ausgewählt werden. Dieser wird nach der Auswahl nur als blauer Rahmen dargestellt und an den Mauszeiger geheftet. Dies ist am achten Messpunkt in Abb.24 zu sehen. Der Messpunkt kann nur entlang der y-Achse bewegt werden. Wenn die Option "overlapping pulses" in den erweiterten Einstellungen deaktiviert ist, passt PulSi die restlichen Messpunkte entsprechend an. PulSi passt die Höhe der Messpunkte so an, dass die durchschnittliche Höhe der Messpunkte 100% beträgt. Der Bereich der y-Achse, welcher dargestellt wird, kann im Bereich B (Abb.23) ausgewählt werden. Im Feld "y axis lower limit" wird die untere Grenze der y-Achse definiert. Im Feld "y axis upper limit" wird die obere Grenze definiert. Der Nutzer kann alle Einstellungen des zu suchenden Intensitätsprofils auf den Stand bei Öffnen der erweiterten Optionen mit der Schaltfläche "reset" (Abb.22 Bereich 8), in den erweiterten Optionen zurücksetzen.

Alle im Fenster "erweiterte Einstellungen" durchgeführten Veränderungen werden durch Benutzung der Schaltfläche "save" im Bereich 9 des Fensters "advanced settings" (Abb.25) gespeichert. Die Fenster "advanced settings" und "grafic input" werden nach Benutzung der Schaltfläche geschlossen. Mit der Schaltfläche "abort" in Abb.25 werden alle Änderungen verworfen und die Fenster geschlossen.

Für die Weiterentwicklung von PulSi wurden in den erweiterten Optionen Auswahlmöglichkeiten für unterschiedliche Eingabemethoden der Kristalleigenschaften implementiert. Diese Optionen sind in Abb. 26 zu sehen. In PulSi gibt es

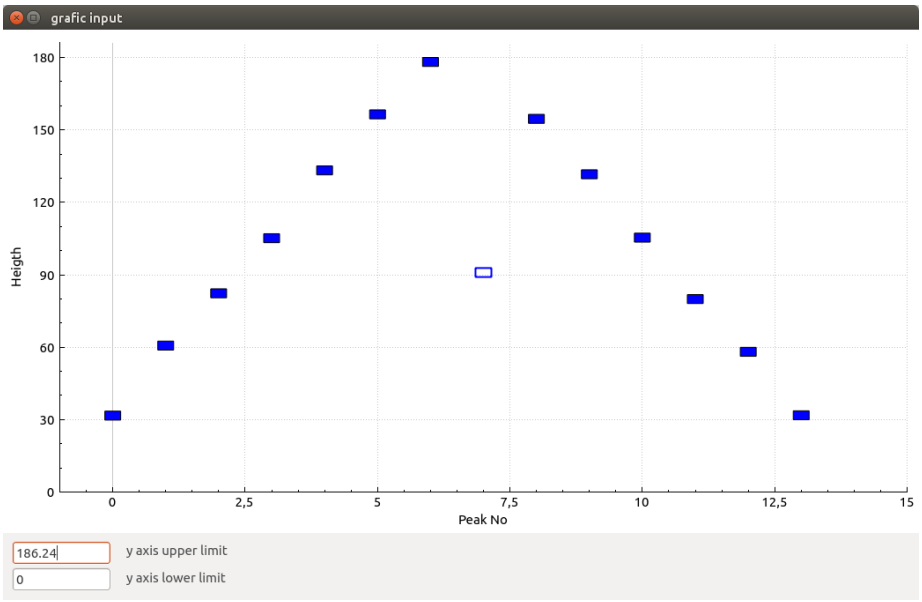


Abbildung 24: Grafisches Eingabeinterface des geforderten Intensitätsprofils in PulSi während des Verschiebens eines Messpunktes (Messpunkt 8)

Abbildung 25: Schaltflächen zum Speichern oder Verwerfen der in den erweiterten Optionen gemachten Einstellungen (Abb.18 Bereich 9)

die Möglichkeit zwischen der Eingabe der Phasenverschiebung ("Input of phaseshift") und der Eingabe des Brechungsindex der Kristalle("Input of refraction index") zu wählen. Die Eingabe des Brechungsindex erfordert eine präzise Mes-

Abbildung 26: Auswahloptionen für die Eingabemethode und den Berechnungsalgorithmus (Abb.18 Bereich 10)

sung der Brechungsindizes der Kristalle und ihrer Temperaturabhängigkeit. Diese ist momentan nicht geplant. In der momentanen Version von PulSi wird der selbe Brechungsindex für alle Kristalle definiert. Es ist somit nicht möglich, die Brechungsindizes für jeden Kristall einzeln zu ändern und so das Intensitätsprofil zu glätten. Die Option zur Eingabe des Brechungsindex wurde implementiert

für den Fall, dass diese Eingabemethode in späteren Versionen gefordert werden sollte. Die aktuelle Implementierung der Eingabe des Brechungsindex erlaubt nicht den Zugriff auf alle Funktionen von PulSi.

4.1.3 Der automatische Suchlauf

Pulsi besitzt für den geplanten automatischen Betrieb des Beamshapers einen Algorithmus, welcher es ihm ermöglicht, die Kristallkonfigurationen zur Generierung definierter Intensitätsprofile zu finden. Dieser Algorithmus wird automatischer Suchlauf genannt. In diesem Abschnitt werden die Bedienung sowie die Ausgaben des Suchlaufs thematisiert. Der Algorithmus hinter dem automatischen Suchlauf wird in Abschnitt 4.3 genau beschrieben.

Der automatische Suchlauf kann selbstständig Kristallkonfigurationen finden, welche ein definiertes Intensitätsprofil erzeugen. Vor dem Start des automatischen Suchlaufs muss das Intensitätsprofil, welches der Laserpuls aufweisen soll, sowie die Grenzen der Drehwinkel der Kristalle definiert werden. Dies wird in den erweiterten Optionen durchgeführt (Abschnitt 4.1.2). Der automatische Suchlauf wird durch das Betätigen der Schaltfläche "start" im Hauptfenster gestartet (Abschnitt 4.1.1 Abb.9 Bereich 7).

Zunächst überprüft PulSi die definierten Eigenschaften des Beamshapers. Dies ist notwendig, da bei der Definition der Eigenschaften nicht berücksichtigt wird, ob die Eingaben physikalisch sinnvoll oder vollständig sind. Bei Eingaben, welche nicht sinnvoll oder unvollständig sind, tritt ein unvorhersehbares Verhalten des Suchlaufs auf. Dies führt im schlimmsten Fall zum Absturz von PulSi. Wird von PulSi ein Fehler entdeckt, wird in roter Schrift, unter der Schaltfläche "start", eine Fehlermeldung ausgegeben. Wird kein Fehler gefunden, startet der Suchalgorithmus des Suchlaufs und "Simulation running" erscheint unter der Schaltfläche "start" (Abschnitt 4.1.1, Abb.9, Bereich 7). Die Schaltfläche "start" wird deaktiviert, bis der Suchalgorithmus des automatischen Suchlaufs beendet ist. Durch das Starten mehrerer Instanzen von PulSi ist es möglich, mehrere Suchläufe parallel auszuführen. Das parallele Ausführen von Suchläufen in einer Instanz von PulSi ist nicht möglich.

Nach dem Start des automatischen Suchlaufs wird das Fenster "current status" geöffnet (Abb.27). In diesem Fenster kann der Nutzer die Simulation abbrechen und den aktuellen Status der Simulation sehen. Im oberen Bereich ist die Schaltfläche "stop" zu sehen. Mit ihr kann der Suchlauf bei Bedarf abgebrochen werden. Der Abbruch erfolgt, wenn der Suchalgorithmus den nächsten Kristall auswählt(siehe Erklärung des Suchalgorithmus in Abschnitt 4.3). Es können deshalb, je nach Anzahl der Kristalle, einige Sekunden bis zu einer Minute zwischen Betätigung der Schaltfläche "stop" und der tatsächlichen Beendigung des Suchlaufs vergehen. Die Zeit steigt mit der Anzahl der Kristalle im Beamshaper. Über der Schaltfläche ist ablesbar, ob der Suchlauf arbeitet ("Simulation runnig") oder ob er bereits abgeschlossen wurde("Simulation finished").

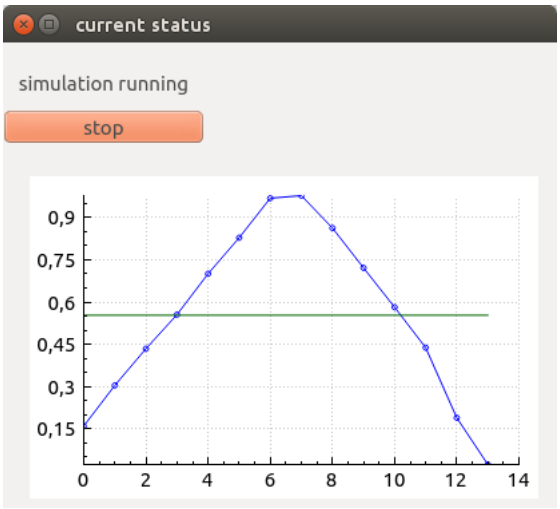


Abbildung 27: Fenster “current status” mit dem aktuellen Status des automatischen Suchlaufs in PulSi(Testpunkte: blaue Punkte; durchschnittliche Höhe der Testpunkte: grüne Linie)

Im unteren Bereich ist ein Diagramm zu sehen. Hier kann der Nutzer den aktuellen Stand der Simulation sehen. Die Messpunkte sind die blauen Kreise. Die grüne Linie zeigt die durchschnittliche Höhe der Messpunkte. Da PulSi hauptsächlich die Aufgabe hat, Flat-Top-Intensitätsprofile zu berechnen, lag diese Darstellung nahe. Die obere Grenze des Diagramms wird durch den größten Messpunkt und die untere Grenze durch den kleinsten Messpunkt definiert. Bei der Suche nach einem Flat-Top-Profil nähern sich die Messpunkte der grünen Linie an. Das Diagramm wird jedes mal, wenn der Suchalgorithmus einen neuen Kristall ausgewählt, aktualisiert.

4.1.4 Speichern und Laden von Daten

In PulSi können die simulierten Profile gespeichert werden. Zum Speichern der Daten des Diagramms muss man im Hauptfenster “File”→”Save plot data” auswählen. Zum Speichern der definierten Eigenschaften und der Kristallkonfiguration des Beamshapers muss “File”→”save data” ausgewählt werden. Man kann hier sowohl das Diagramm als auch die Kristallkonfiguration speichern (Abb.28). Das Diagramm wird als .png Datei und die Rohdaten als .txt Datei gespeichert. Die Kristallkonfiguration wird ebenfalls als .txt Datei gespeichert(Anhang A). In den .txt Dateien werden die Daten in einer Form gespeichert, dass man sie, ohne PulSi zu starten, auslesen kann. Alle Dateien werden im Ordner der Anwendungsdatei von PulSi gespeichert.

Die Kristallkonfigurationen können über “File”→”load data” geladen werden. Es öffnet sich ein Fenster, in welchem das Dateisystem des Computers zu sehen ist.

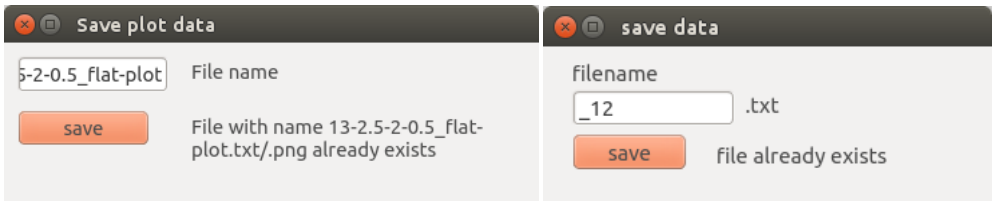


Abbildung 28: Fenster zum Speichern des Diagramm(links) und der Kristallkonfiguration(rechts)

Der Nutzer kann jede Datei laden. Es wird keine Fehlermeldung ausgegeben, wenn die Datei eine unerwartete Struktur hat. Bei Dateien, welche nicht die erwartete Struktur aufweisen, stürzt PulSi ab.

PulSi kann ebenfalls OSS-Daten laden (Abb.29). Hierzu muss der Nutzer das

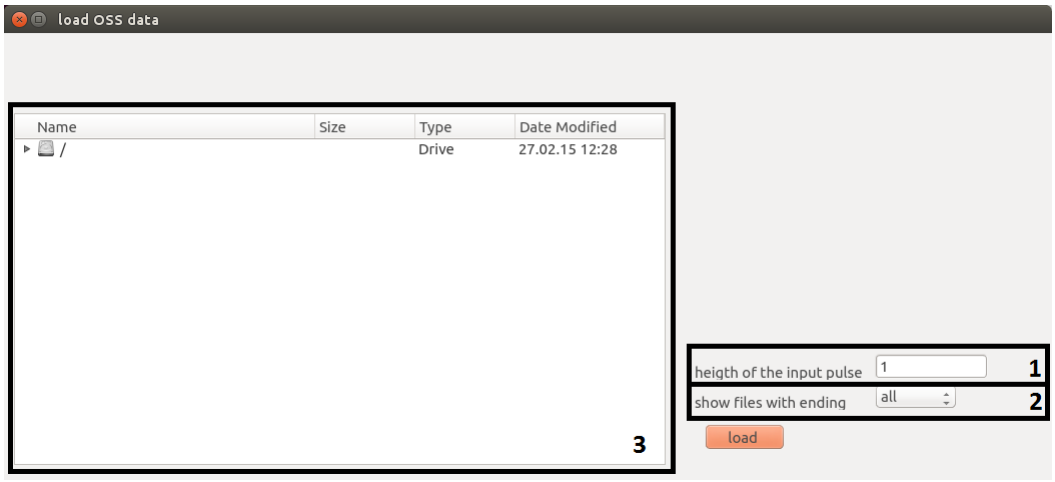


Abbildung 29: Interface zum Laden von OSS-Daten(Verschiedene Funktionen durch schwarze Rahmen gekennzeichnet)

Fenster aus Abb.29 über "File"→"load OSS data" öffnen. In Bereich 3 (Abb.29) sieht der Nutzer das Dateisystem seines Rechners. Zur besseren Navigation kann der Nutzer in Bereich 2 (Abb.29) wählen, ob er alle Dateien sehen möchte oder nur Dateien mit der Endung .oss. Dieses Dateiformat besitzen die Dateien mit Messungen des OSS normalerweise. Wenn der Nutzer eine Datei ausgewählt hat, muss er die Höhe des Laserpulses am Eingang des Beamshapers in Bereich 1 definieren. Die Höhe ist standardmäßig auf 1 festgelegt. Der Wert hängt von den Einstellungen im Lasersystem ab und sollte bei jeder Messung im OSS zusätzlich gemessen werden. Dies ist notwendig, um die OSS Daten auf die Höhe der von PulSi berechneten Kurve zu skalieren. Zum Abschluss betätigt der Nutzer die Schaltfläche "load". Danach wird die Kurve im Hauptfenster, nach Auswahl der

entsprechenden Option, von PulSi angezeigt werden.

4.2 Implementierung der Berechnung der Pulsform

Für die Simulation des Intensitätsprofils ist es notwendig, die Pulsform zu berechnen. Hierzu wurde der für die Berechnung der Intensitätsverteilung in doppelbrechenden Kristallen entwickelte Algorithmus aus Abschnitt 2.2.1 verwendet. Zunächst soll der Berechnungsalgorithmus für ein Intensitätsprofil eines Beamshaper vorgestellt werden. Danach wird die Genauigkeit der Berechnung betrachtet.

Die gesamte Pulsform setzt sich aus allen Kopien des Ausgangspulses, welche im Beamshaper entstanden sind, zusammen. Zur Berechnung des Intensitätsprofils muss bekannt sein, welche Intensitäten die einzelnen Kopien haben, sowie um welche Zeit sie verzögert werden. Der Beamshaper des MBI Lasersystems besteht aus 13 doppelbrechenden Kristallen. In jedem Kristall wird die Anzahl der eintreffenden Pulse verdoppelt. Somit besteht der Laserpuls am Ende aus einer Überlagerung von $2^{13} = 8192$ Kopien des gaußförmigen Laserpulses am Eingang des Beamshapers. Aus der Anzahl der Verzögerungen und der Größe der zeitlichen Verzögerung lässt sich der Zeitindex der Kopie errechnen, an welchem sich die Kopie im Intensitätprofil befindet. In der Berechnung wird angenommen, dass alle Kristalle des Beamshapers die gleiche Dicke aufweisen und somit die gleiche Verzögerung erzeugen. Für die Übertragung der Simulation auf den realen Beamshaper reicht es aus, wenn diese Forderung nur annähernd erfüllt ist. In der Realität verursachen nicht alle Kristalle exakt die gleiche Verzögerung der Laserpulse. Die Annahme der Verzögerung vereinfacht jedoch die Berechnung. Es lassen sich so die Zeitindices der Kopien des Laserpulses sehr einfach beschreiben. Durch die Annahme gibt es $K_{ges} + 1$ Zeitindices, an denen sich die Kopien befinden können.

Als Beispiel für die Berechnung der Zeitindices der Kopien wird ein Beamshaper mit 2 Kristallen verwendet. Man erhält am Ende einen Laserpuls, welcher aus der Überlagerung von 4 Kopien besteht. Diese Kopien können sich an 3 Zeitindices befinden. In Tabelle 3 ist dargestellt, wie sich die einzelnen Kristalle auf die Verzögerung der Kopien auswirken. Die Mitte der Pulsform setzt sich aus 2 Kopien zusammen (Kopie 1;2). Die Verteilung der Kopien auf die einzelnen Zeitindices folgt einer Binomialverteilung. Für die Berechnung des Intensitätsprofils muss die Intensität jeder Kopie bekannt sein.

Nach der Definition der Eigenschaften des Beamshapers werden alle daraus folgenden Variablen im Programm in der Funktion "Definition()" definiert (siehe Quelltext Abschnitt A.3, Zeile 14). Das Programm legt in der Funktion "Definition()" das Array "A[Kopie][Kristall]" mit sämtlichen möglichen Kombinationen von Verzögerungen durch die Kristalle an (siehe Tabelle 3). Die Daten werden in der gleichen Struktur wie in Tabelle 3 gespeichert. Der erste Index beschreibt die Kopie, welche man betrachtet. Der zweite Index beschreibt die Wirkung des Kristalls mit dem selben Index auf die Kopie. Da die Verzögerung durch

Tabelle 3: Verzögerung der Kopien des einfallenden Laserpulses am Beispiel eines, aus 2 Kristallen bestehenden, Beamshapers. Jeder Kristall kann die Kopien um 0 oder 1 Zeiteinheiten verzögern.

Kopie	Verzögerung der Kopie durch Kristall Nr.1 (Array A[0][n])	Verzögerung der Kopie durch Kristall Nr.2 (Array A[1][n])	gesamte Verzögerung der Kopie (Array tk[n])
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	2

den Brechungsindex entsteht, wird so auch die optische Achse des Kristalls ausgewählt. Die gesamte Verzögerung, welche eine Kopie erfährt, wird im Array tk[Kopie] mit dem Datentyp double[1] gespeichert. Da sich die Verzögerung der Kopien im Beamshaper nicht ändert, wird diese einmal bei der Erstellung des Beamshapers berechnet.

Nach der Definition der Variablen kann die Berechnung des Intensitätsprofils beginnen. Die Intensitäten der einzelnen Kopien berechnen sich, wie in Abschnitt 2.2.1 beschrieben. In PulSi werden die Intensitäten der Kopien in der Funktion "Inten()" berechnet(siehe Quelltext Abschnitt A.3, Zeile 163). Für die Berechnung der Intensitäten der einzelnen Kopien ist die Phasenverschiebung der Kopien zueinander vorerst nicht von Bedeutung. Die Kopien werden deshalb vorerst nur als Gaußfunktionen betrachtet. Das Programm berechnet sequentiell, für alle möglichen Kombinationen, die Intensität. Hierbei wird betrachtet, bei welchem Kristall die Kopie auf welche optische Achse projiziert wird. Für die Intensität, der in einem Kristall verzögerten Kopie, gilt:

$$\vec{I}(K_N) = \begin{pmatrix} I_{x,K_{N-1}} sin^2(\psi) - I_{y,K_{N-1}} cos(\psi) \cdot sin(\psi) \\ I_{y,K_{N-1}} cos^2(\psi) - I_{x,K_{N-1}} cos(\psi) \cdot sin(\psi) \end{pmatrix}.$$

Für die nicht verzögerte Kopie gilt:

$$\vec{I}(K_N) = \begin{pmatrix} I_{x,K_{N-1}} cos^2(\psi) + I_{y,K_{N-1}} cos(\psi) \cdot sin(\psi) \\ I_{y,K_{N-1}} sin^2(\psi) + I_{x,K_{N-1}} cos(\psi) \cdot sin(\psi) \end{pmatrix}.$$

Um Speicherplatz zu sparen, wird sofort die Stellung des Polarisators am Ende des Beamshapers bei der Intensitätsberechnung berücksichtigt. Somit wird die Intensität nicht als zweidimensionaler Vektor, sondern nur der Absolutwert gespeichert.

$$I_m = I_{x,K_{ges}} cos(\psi_{Polarisator}) - I_{y,K_{ges}} sin(\psi_{Polarisator})$$

[1] Der Datentyp double besitzt eine Genauigkeit von 16 Nachkommastellen.

Die berechneten Werte werden im Array “Intensitaet[Kopie]” gespeichert. Die Werte werden als Datentyp double gespeichert. Durch die Genauigkeit der double Variablen ist der Fehler, welcher durch die numerische Ungenauigkeit entsteht, vernachlässigbar klein.

Als nächster Schritt wird das gesamte Intensitätsprofil errechnet. Hierzu wird über alle Kopien summiert. In Abb.30 ist das Intensitätsprofil, welches durch den als Beispiel genutzten Beamshaper mit den Eigenschaften aus Tabelle 4 erzeugt wurde, zu sehen. Die orangenen Gaußkurven stellen die Summation der

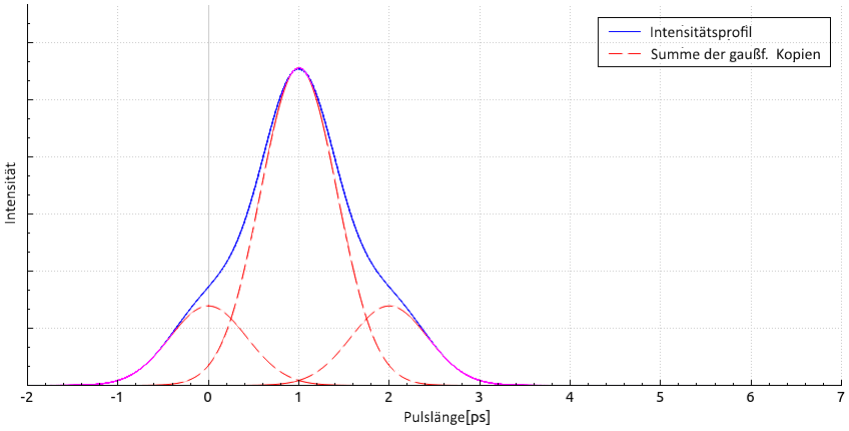


Abbildung 30: Intensitätsprofil in einem aus 2 Kristallen bestehenden Beamshapers geformt. Die Spezifikationen des Beamshapers und des Eingangslaserpulses sind in Tabelle 4 zu finden.

Tabelle 4: Spezifikationen des Beispielbeamshapers

Breite des Eingangslaserpuls	$1\,ps$
Anzahl der Kristalle	2
zeitliche Verzögerung durch die Kristalle	$1\,ps$
Phasenverschiebung	
Kristall Nr.1	$0,5\pi$
Kristall Nr.2	$0,5\pi$
Position des Ausgangspolarisators	$0°$

Kopien, welche sich an dieser Stelle befinden, dar. Aus Tabelle 3 ist zu erkennen, dass 2 Kopien um eine Picosekunde verzögert sind. Somit setzt sich die mittlere Gaußkurve, welche um eine Picosekunde verzögert ist, aus 2 Kopien zusammen. Die Gaußkurven werden berechnet, indem die Höhe der einzelnen Kopien (Array “Intensitaet[Kopie]”) an diesem Zeitindex summiert wird. Für eine präzise Abbildung müsste die Phasenverschiebung jeder einzelnen Kopie berücksichtigt werden. Die Gaußkurven dienen nur der Orientierung. Um Rechenzeit zu sparen,

wird deshalb die Phasenverschiebung nicht in die Berechnung der Gaußkurven einbezogen. Daraus folgt, dass die Berechnung der Höhe der Gaußkurven ungenau wird, sobald der Beamshaper aus mehreren Kristallen besteht. Durch die gewählten Eigenschaften des Beispielbeamshapers ist die Berechnung in diesem Sonderfall exakt. Die Summation der Höhe der Kopien an diesen Zeitindices ist nur möglich aufgrund der Annahme, dass jeder Kristall die selbe Verzögerung verursacht.

Die blaue Kurve in Abb.30 stellt die Einhüllende bzw. das Intensitätsprofil des Laserpulses nach dem Beamshaper dar. Für die Berechnung des Intensitätsprofils wird das Laserlicht als elektromagnetische Welle betrachtet. Die Welle hat die Form $E(t) = \cos(\omega t + \phi)$. Die Phasenverschiebung der Kopien zueinander muss somit ebenfalls beachtet werden. Im Programm wird definiert, dass die Kopie, welche verzögert wird, ebenfalls um die gesamte Phasenverschiebung von ordentlicher zu außerordentlicher Achse verschoben wird. Die unverzögerte Kopie wird nicht in ihrer Phase verschoben. Somit kann für die Berechnung der Phasenverschiebung das Array die gleiche Struktur besitzen, wie für die Berechnung der Verzögerung der Kopien. Es wird somit wieder das Array "A[Kopie][Kristall]" genutzt. Es ist zu beachten, dass jeder Kristall eine andere Phasenverschiebung verursachen kann. Die Phasenverschiebung ϕ_m muss somit für jede Kopie einzeln berechnet werden. Die Formel hierfür lautet:

$$\phi_m \quad = \quad \sum_{K_n=1}^{K_{ges}} A(m)(K_n) \cdot \phi_{K_n}.$$

Die Phasenverschiebung kann sich auch durch Änderung der Einstellungen der Kristalle ändern. Deshalb muss diese Rechnung, bei jeder Eingabe des Nutzers, neu durchgeführt werden. Die Phasenverschiebung wird im Array "Phasenverschiebung[Kopie]" als Datentyp double gespeichert. Die Wellenfunktion einer Kopie $E_m(t)$ des Laserpulses ist eine Faltung der Funktion der elektromagnetischen Welle und einer Gaußkurve:

$$E_m(t) \quad = \quad I_m \cdot e^{-\frac{(t-t_0)^2}{2 \cdot \tau^2 \cdot \ln 2}} \cdot \cos(\omega \cdot t + \phi_m). \tag{26}$$

Hier ist I_m die Intensität der m-ten Kopie, t_0 der Erwartungswert der Gaußkurve und τ ihre Halbwertsbreite. Für die Berechnung des Intensitätsprofils werden 1000 Stützstellen verwendet. An jedem Punkt muss über die Wellenfunktionen aller Kopien $E_m(t)$ summiert werden. Die Funktion zur Berechnung eines Punktes lautet somit (siehe siehe Quelltext Abschnitt A.3 Zeile 237):

$$\begin{aligned} E(t) &= \sum_{m=1}^{m_{ges}} E_m(t), \\ I(t) &= \frac{c}{2\epsilon_0} E^2(t). \end{aligned}$$

Da es sich um eine elektromagnetische Welle mit variabler Phasenverschiebung handelt, wird bei einer Verteilung der 1000 Stützstellen mit konstantem Abstand

nicht davon ausgegangen, dass sich der berechnete Punkt, im folgenden Messpunkt genannt, immer am Maximum der Welle befindet. Der Grund hierfür liegt in den verschiedenen Phasenverschiebungen der Kopien. Es ist nur mit großen Rechenaufwand möglich, die Phase der Welle am Messpunkt zu berechnen. Dies wäre erforderlich, um das Maximum der Welle zu ermitteln. Für ein präzises Intensitätsprofil ist es notwendig einen Punkt, welcher sehr nah am Maximum liegt, zu berechnen.

Aufgrund der Erfahrungen mit dem aktuell im Betrieb befindlichen Beamshaper kann davon ausgegangen werden, dass die zu berechnenden Intensitätsprofile Längen zwischen ca. 3 ps und ca. 25 ps aufweisen werden. Diese Werte werden angenommen, da der kleinste geformte Puls eine Breite von ca. 3 ps (Beamshaper mit einem Kristall) und der, laut Spezifikationen des Beamshapers längste Flat-Top-Puls, eine Länge von 25 ps aufweist[14]. Die Anstiegszeit der Flanken der Flat-Top-Pulse beträgt ca. 2 ps. Das Diagramm wird eine Zeitspanne von ca. 6 ps bis ca. 30 ps abdecken, um Pulse inklusive der Flanken vollständig darstellen zu können.

Für die Berechnung der elektromagnetischen Welle muss eine Referenzwellenlänge ω gewählt werden. Im Folgenden wird die Wahl dieses Parameters der Funktion begründet und sein Einfluss auf die Genauigkeit der Berechnung des Intensitätsprofils dargestellt.

Bei der kürzesten angenommenen Zeitspanne, welche das Diagramm abdeckt, muss circa alle 6 ps/1000=6 fs eine Stützstelle liegen. Der Zeitabstand der Messpunkte entspricht einer Referenzwellenlänge von 1,8 μm. Die Referenzwellenlänge in der Rechnung muss deshalb kleiner als 1,8 μm sein. Der gleiche Algorithmus wird zur Berechnung der Höhe der Messpunkte im automatischen Suchlauf verwendet. Es muss deshalb gewährleistet werden, dass das Maximum nah am erwarteten Testpunkt des automatischen Suchlaufs liegt. Eine zu große Abweichung vom Maximum der Welle, zu dem vom automatischen Suchlauf angenommenen Testpunkt, verringert die Genauigkeit des Suchlaufs. Das ist besonders problematisch, wenn die Einhüllende des Intensitätsprofils, im Bereich des Testpunktes, einen starken Anstieg aufweist. Der automatische Suchlauf wird in Abschnitt 4.3 genauer erklärt.

Bei der Definition der Eigenschaften des Beamshapers, durch Eingabe der Brechungsindizes der Kristalle und ihrer Temperatur, muss ebenfalls die im MBI-Lasersystem verwendete Wellenlänge des Lichts, als Referenzwellenlänge, definiert werden. Die Phasenverschiebung wird dann aus dem Brechungsindex und der Temperatur des Kristalls berechnet. Da im Programm die Phasenverschiebung direkt eingegeben wird, ist die Referenzwellenlänge nicht für die Berechnung physikalischer Eigenschaften notwendig. Die Wahl der Referenzwellenlänge wird deshalb nur von der geforderten Genauigkeit des automatischen Suchlaufs bestimmt.

Die Referenzwellenlänge wurde auf 1 nm festgelegt. Dies entspricht einer Periodendauer von 3,3 as. Da für die Berechnung der Intensität nur der Betrag der Wellenfunktion von Bedeutung ist, kann sowohl das Minimum, als auch das

Maximum für die Berechnung des Intensitätsprofils genutzt werden. Somit kann angenommen werden, dass der Betrag der Wellenfunktion, innerhalb einer halben Periodenlänge, ein Maximum aufweist. Daraus folgt, dass innerhalb eines Bereiches von $\pm$ 0,83 as um den Messpunkt, ein Maximum der Wellenfunktion zu finden ist. Wie in Abb.31 zu sehen, werden pro halbe Periode 20 Messpunkte

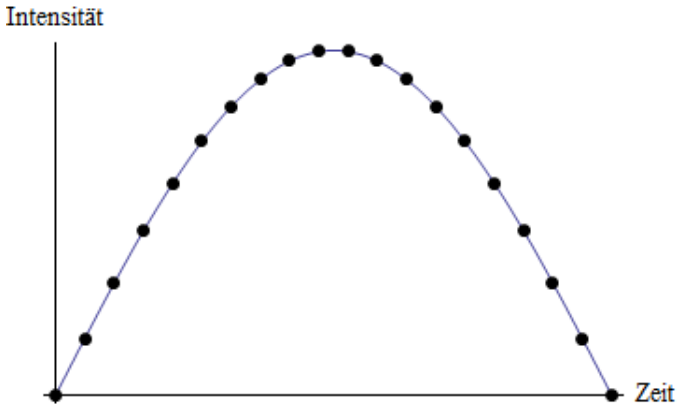


Abbildung 31: Messpunkte auf der sinusförmigen Wellenfunktion zur Berechnung einer Stützstelle des Intensitätsprofils (schwarze Punkte: Messpunkte, blaue Kurve: Wellenfunktion)

berechnet. Um auch den Sonderfall, dass das Maximum direkt am Rand liegt, zu berücksichtigen, wird auf beiden Seiten jeweils ein Punkt außerhalb des Intervalls berechnet. Es werden somit insgesamt 22 Messpunkte berechnet. Die Wahl von 22 Messpunkten ist ein Kompromiss zwischen Rechenzeit und Genauigkeit der Berechnung. Für eine größere Genauigkeit wären mehr Messpunkte nötig. Aus diesen 22 Messpunkten wird der größte Wert ausgewählt. Dieser Wert wird im Diagramm, als Stützstelle, am dem entsprechenden Zeitindex eingetragen.

Man kann mit den 20 Messpunkten pro Periode die maximale Abweichung vom tatsächlichen Maximum der elektromagnetischen Welle berechnen. Der Abstand der Messpunkte zueinander beträgt $\frac{1}{20}\pi$. Die größte Entfernung zu einem Messpunkt weist das Maximum der Welle auf, wenn es sich mittig zwischen zwei Messpunkten befindet. So ist das Maximum $\frac{1}{40}\pi$ vom nächsten Messpunkt entfernt. Die maximale Abweichung beträgt somit $1 - cos\left(\frac{1}{40}\pi\right) \approx 3 \cdot 10^{-3}$. Das entspricht einem Fehler von 0.3 %. In Abb.32 ist eine von PulSi berechnete Pulsform zu sehen. Im Flat-Top-Bereich (0 ps bis 26 ps) des Intensitätsprofils erkennt man eine wellenförmige Struktur. Die grobe Wellenstruktur entsteht durch die Überlagerung der Gaußkurven aufgrund der Einstellungen am Beamshaper. Die feine Struktur entsteht durch die Ungenauigkeit der Berechnung. Hier wird die feine Wellenstruktur betrachtet. Die größte Schwankung der Wellenstruktur beträgt im Beispiel von Abb.32 $1,3 \cdot 10^{-5}/4,4 \cdot 10^{-3} \approx$0,3 % (siehe Abb.33). Dieser Wert stimmt mit dem berechneten Fehler überein. Es ist zu vermuten, dass diese Struktur auch an den Flanken des Intensitätsprofils auftritt. Hier ist die Struk-

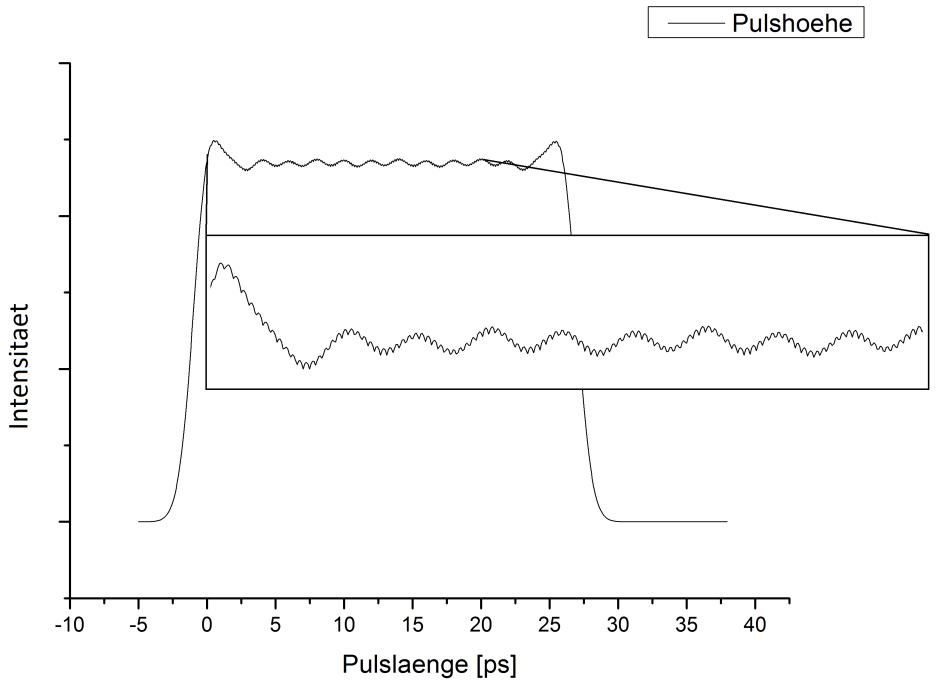


Abbildung 32: Vom Programm PulSi berechnete Pulsform mit einer Wellenstruktur im Flat-Top-Bereich.

tur jedoch, aufgrund des großen Anstiegs, der daraus resultierenden Verzerrung und der geringeren Intensität, geringer ausgeprägt. Die Genauigkeit wird erhöht, indem mehr Messpunkte berechnet werden. In Abb.33 ist der Unterschied der berechneten Kurven bei einer unterschiedlichen Anzahl von Messpunkten zu sehen. Hier wurden einmal 20 Messpunkte je halbe Periode (rote durchgehende Kurve) und einmal 40 Messpunkte je halbe Periode (schwarze Punkt-Strich Kurve) berechnet. Es ist zu erkennen, dass die wellenförmige Struktur bei der Verwendung von 40 Messpunkten kleiner wird. Hier reduziert sich der Fehler auf 0,07%. Die Genauigkeit der Berechnung, bei der Verwendung von 20 Messpunkten, ist ausreichend für die Anwendung und den Vergleich mit dem Beamshaper, da hier die Messungen im OSS, aufgrund der Auflösung, einen größeren Messfehler aufweisen. Außerdem werden die idealisierten Annahmen, welche für die Berechnung des Intensitätsprofils getroffen wurden, in der Realität nicht erfüllt. Hierzu wären perfekte Kristalle notwendig. Zur Berechnung der Testpunkte für den Suchalgorithmus wird das gleiche Verfahren angewendet. Es werden jedoch nur $K_{ges} + 1$ Stützstellen berechnet. Die Stützstellen befinden sich so nahe wie möglich am Zeitindex der Kopien des Eingangslaserpulses. Die Darstellungsform ist in Abb.34 in Abschnitt 4.3 gezeigt.

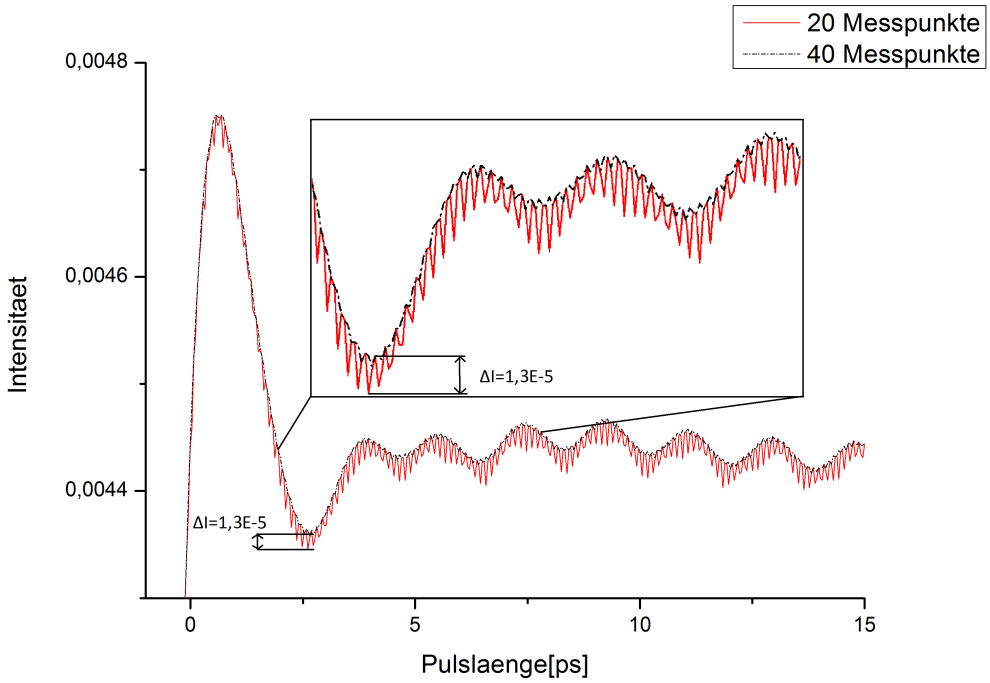


Abbildung 33: Zwei Intensitätsprofile bei einer unterschiedlichen Anzahl an berechneten Messpunkten je Stützstelle (rote Kurve(Linie): 20 Messpunkte je halbe Periode, schwarze Kurve(Strich-Punkt): 40 Messpunkte je halbe Periode)

4.3 Suchalgorithmus

Für PulSi wurde ein automatischer Suchlauf entwickelt. Durch diesen kann PulSi, von beliebigen Kristallwinkeln und dem daraus resultierenden Profil ausgehend, vom Nutzer definierte, Intensitätsprofile selbstständig finden. Die notwendigen Kristallwinkel, welche dieses Profil generieren, werden von PulSi ausgegeben. Bei der automatischen Suche gibt es Einschränkungen, auf die später genauer eingegangen wird. Im folgenden Abschnitt wird dieser automatische Suchlauf vorgestellt.

Bei PITZ werden momentan Schrittmotoren mit einer Schrittweite von 17,5 μrad und einem Drehbereich von $\pm$0,01745 rad verwendet. Für den Fall der Weiterentwicklung des Lasersystems soll PulSi auch mit kleineren Schrittweiten und größeren Drehbereichen arbeiten können. Deshalb wurde der minimale Schrittwinkel mit 0.1 μrad und definiert. Der Drehbereich kann vom Nutzer in den erweiterten Einstellungen (siehe Abschnitt 4.1.2) begrenzt werden. Man kann sich für den Suchlauf verschiedene Lösungsansätze überlegen. Der einfachste An-

satz ist, systematisch jede mögliche Stellung der Kristalle durchzugehen. Dieser Ansatz ist jedoch zeitaufwendig und somit nicht optimal. Aus dem Drehbereich des Motors, seiner minimalen Schrittweite und der Anzahl der Kristalle im Beamshaper lassen sich die Anzahl an möglichen Positionen errechnen. In einem Drehbereich von 360° ergeben sich dann ca. 63 Millionen Möglichkeiten der Positionierung. Da der Beamshaper aus 13 Kristallen mit jeweils dieser Anzahl an Positionierungsmöglichkeiten aufgebaut ist, gibt es insgesamt $2.5{\cdot}10^{101}$Möglichkeiten der Positionierung. Aus der Arbeit mit PulSi ist bekannt, dass die Berechnung des gesamten Intensitätsprofils, einer dieser Möglichkeiten, ca. 1 Min dauert. Die Gesamtlaufzeit des Suchalgorithmus würde somit bis zu $4.8 \cdot 10^{95}$ Jahre betragen.

Der Suchalgorithmus muss, wie aus der eben gemachten Rechnung hervorgeht, optimiert werden. Zur Einsparung von Zeit startet der Suchalgorithmus die Suche nicht mit der minimalen Winkelschrittweite, sondern mit einer Schrittweite von 1°. Es muss nun nicht nur überprüft werden, ob die aktuelle Kristallkonfiguration das geforderte Intensitätsprofil erzeugt, es muss zudem eine interne Genauigkeit definiert werden und diese, genau wie der Winkelschritt, angepasst werden. Zur weiteren Einsparung von Rechenzeit wird im Suchalgorithmus nicht die gesamte Pulsform berechnet. Der Simulation liegt die Annahme zugrunde, dass alle Kristalle die gleiche Verzögerung der Laserpulse bewirken. Hier soll nun der automatische Suchlauf zur einfacheren Beschreibung an einem Beispielbeamshaper mit 2 Kristallen erklärt werden. Der Berechnungsalgorithmus der Testpunkte ist der gleiche Algorithmus wie zur Berechnung des Intensitätsprofils, welcher in Abschnitt 4.2 am selben Beispiel erklärt wurde. Hier werden jedoch nur $K_{ges} + 1$ Punkte berechnet. Die Testpunkte der Simulation (blaue Kreuze) befinden sich am Zeitindex der gaußförmigen Kopien (orange Kurve) (siehe Abb.34). Das komplette Profil (blaue Kurve) ist in Abb.35 zu sehen. Der Raum zwischen den 3 Testpunkten in Abb.34 ist von Bedeutung, wenn die Phasenverschiebung durch die Temperatur ebenfalls optimiert werden soll. Der Suchalgorithmus soll jedoch nur die Kristallwinkel bestimmen. Deshalb sind nur die 3 Testpunkte, an welchen die gaußförmigen Kopien liegen, von Bedeutung. Da nur diese 3 Punkte betrachtet werden, wird Rechenzeit bei der Berechnung der Testpunkte gespart. Weitere Rechenzeit kann gespart werden, wenn man effektiv bestimmen kann, welcher Kristall, welchen Testpunkt beeinflusst. Man kann annehmen, dass jeder Testpunkt M von 2 Kristallen beeinflusst wird. Die Auswahlregel lautet[5]:

$$M = K_{ges} - K_n + 1, \tag{27}$$

$$M = K_{ges} - K_n + 2. \tag{28}$$

Das Ziel des Suchalgorithmus ist es, die Abweichung der Testpunkte zum geforderten Wert zu minimieren. Die Genauigkeit des Suchlaufs wird durch die Genauigkeit der im Programm genutzten Variablen und die minimale Schrittweite der Schrittmotoren des Beamshapers beschränkt. Den größten beschränkenden Einfluss hat die minimale Schrittweite der Schrittmotoren. Die minimale Schrittweite wurde, wie bereits erwähnt, mit 0,1 μrad definiert. Der Einfluss der

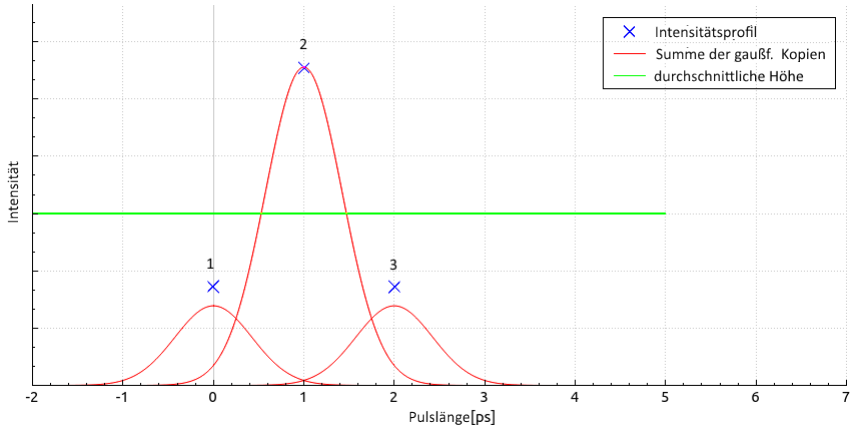


Abbildung 34: Darstellung der Pulsform aus PulSi, wie sie in der Simulation verwendet wird. Die blauen Kreuze stellen die Testpunkte und die grüne waagerechte Linie ihre durchschnitlliche Höhe dar. Die gaußförmigen Pulse stellen die Summe der Kopien des Laserpulses an dieser Stelle dar.

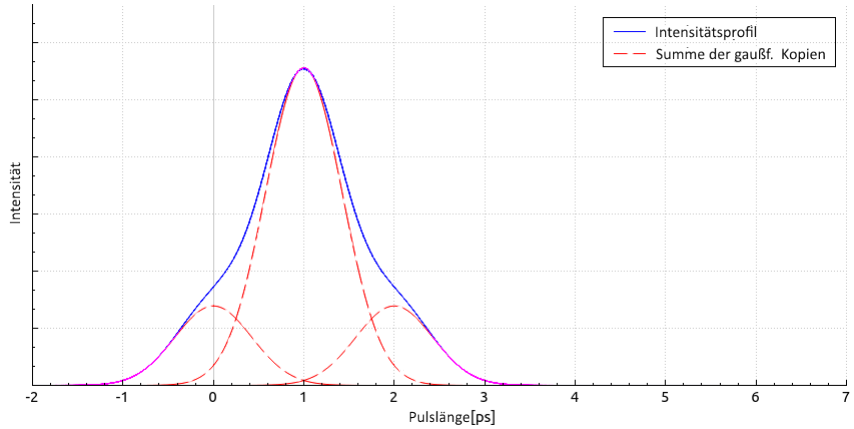


Abbildung 35: Vollständige Darstellung der Pulsform aus PulSi mit 1000 Stützstellen. Die gaußförmigen Pulse stellen die Summe der Kopien des Laserpulses an dieser Stelle dar.

Variablendeklaration wurde durch die Deklaration der Intensität als double minimiert. Damit wird erreicht, dass auch bei Erreichen der minimalen Schrittweite das Profil korrekt dargestellt wird. Der Nutzer muss die Genauigkeit, bis zu welcher das simulierte Profil der Vorgabe angenähert werden soll, festlegen. Es gibt keine Fehlermeldung, falls der Suchalgorithmus, aufgrund des Erreichens der minimalen Schrittweite, kein Ergebnis mit der geforderten Genauigkeit finden kann.

Die zu erreichende Höhe der Testpunkte wird relativ zur durchschnittlichen Höhe definiert. Die durchschnittliche Höhe aller Zielwerte muss immer ca. 100% betragen, damit eine Lösung gefunden werden kann. Die maximale Abweichung von diesem Wert, bei welchem noch eine Lösung gefunden werden kann, wird von der maximalen Abweichung, welche das simulierte Profil zur Vorgabe haben darf, bestimmt. Diese wird durch den Nutzer bestimmt. Durch numerische Ungenauigkeiten kann ein Wert von exakt 100% nicht erreicht werden. Wenn man einen Absolutwert als Zielwert angibt und dieser nicht erreicht wird, findet der Suchlauf keine Kristallkonfiguration zur Generierung des Intensiätsprofils. Der Suchlauf fällt in eine Endlosschleife.

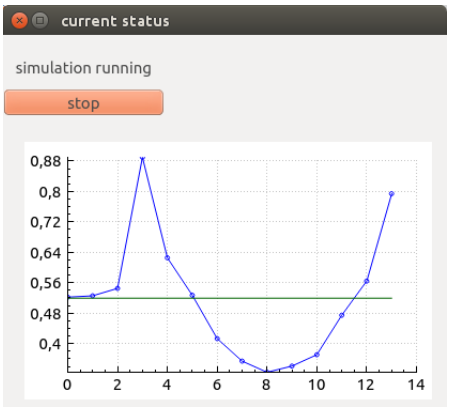
PulSi versucht das geforderte Profil von der linken Seite anzunähern(von Testpunkt 1). PulSi ändert die Intensität der Testpunkte 1 und 2 im Intensitätsprofil des Bespielbeamshapers aktiv. In einem Beamshaper aus einer beliebigen Anzahl von Kristallen (K_{ges}), werden die Testpunkte 1 bis K_{ges} aktiv geändert. Die zu drehenden Kristalle werden mit der Formel 27 für die Testpunkte 1 bis $K_{ges} - 1$ ausgewählt. Für den Testpunkt K_{ges} wird der Kristall mit der Formel 28 ausgewählt. Der letzte Testpunkt wird nicht aktiv verändert. Für den Beispielbeamshaper ist die Auswahl der Kristalle in der Tabelle 5 zu finden. Durch

Tabelle 5: Zielwerte der Testpunkte in Prozent der durchschnittlichen Intensität und Auswahl der zu drehenden Kristalle bei einem Beispielbeamshaper aus 2 Kristallen

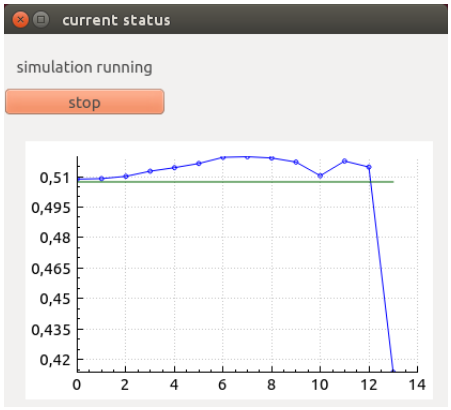
Testpunkt	Zielwert [% der durchschnittlichen Höhe]	zu drehender Kristall
1	90	1
2	120	2
3	90	-
Durchschnitt	100	

diese Auswahl wird die Abweichung des n-ten Testpunktes auf den n+1-ten Testpunkt übertragen. Dies ist in den Abb.36a bis 36d zu sehen.

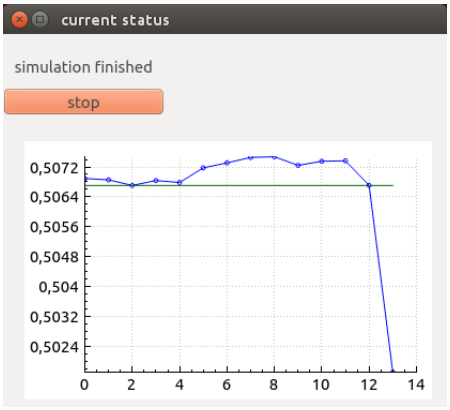
(a) Intensitätsprofil beim Start des Suchlaufs



(b) Intensitätsprofil nach ca. 5min



(c) Intensitätsprofil nach ca. 40min



(d) Intensitätsprofil nach Beendigung des Suchlaufs

Abbildung 36: Intensitätsprofil im automatischen Suchlauf zu verschiedenen Zeitpunkten bei der Suche nach einem Flat-Top-Profil. Die Skalierung der y-Achse wird von PulSi automatisch angepasst. Zur besseren Darstellung wurde ein Intensitätsprofil eines Beamshaper aus 13 Kristallen dargestellt.

Am Beispiel aus Tabelle 5 lässt sich erkennen, dass das Intensitätsprofil erreicht wird, wenn der Testpunkt 1 die Höhe von 90% und der Testpunkt 2 die Höhe von 120% erreicht haben. Durch die Angabe der Höhe in relativen Werten muss der Testpunkt 3 nun nämlich die geforderte Höhe von 90% aufweisen. Der Grund ist, dass alle Messpunkte zusammen immer eine durchschnittliche Höhe von 100% haben. Der Vorteil dieser Herangehensweise ist, dass man kein Wissen braucht, wie groß die Intensität des Pulses am Ende des Beamshapers ist. Dieses Schema ist auf verschiedene Intensitätsprofile anwendbar.

Das Schema des Suchalgorithmus ist in Abb. 37 grafisch dargestellt. Nach dem Start des Suchalgorithmus werden zunächst die Intensitäten der 3 Testpunkte

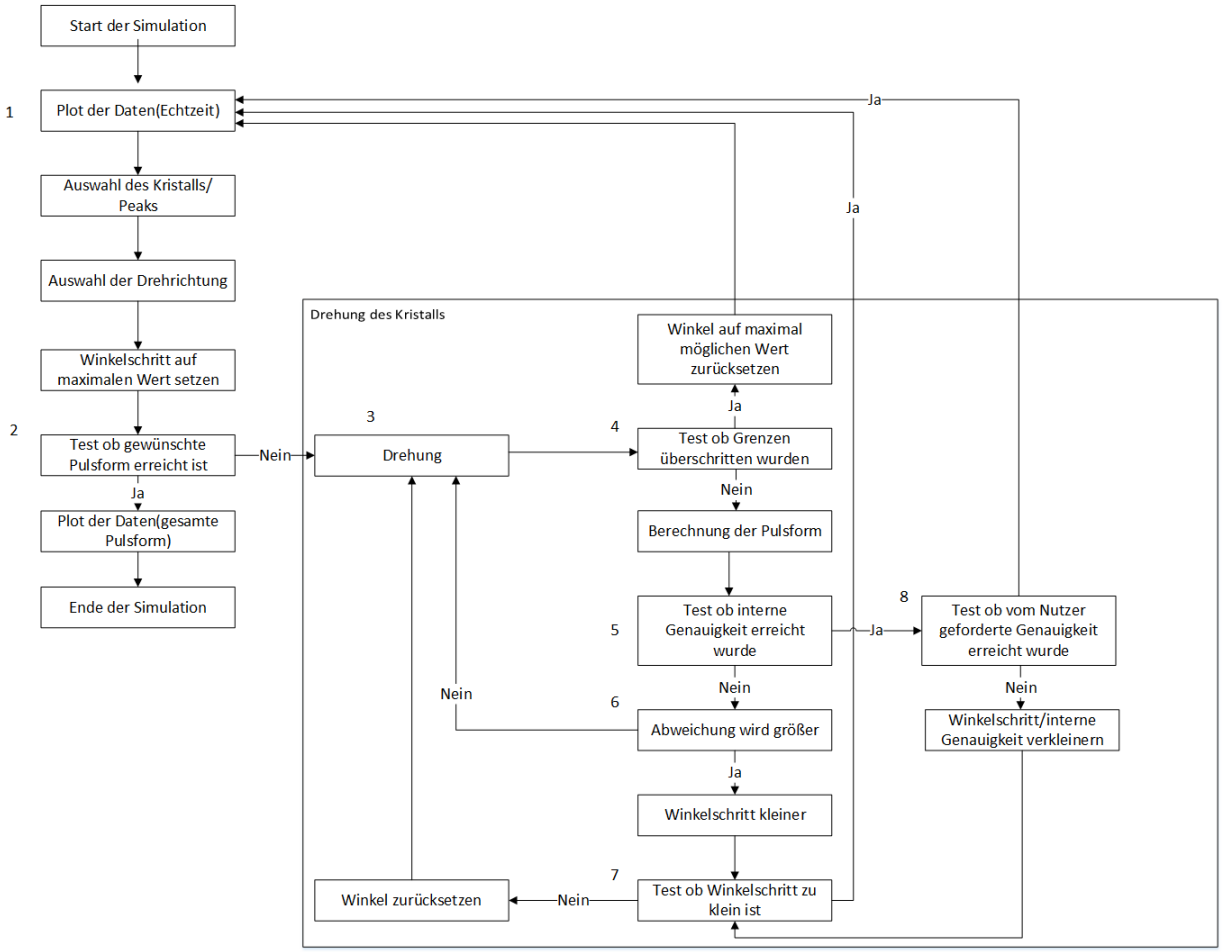


Abbildung 37: Funktionsweise des Suchalgorithmus inklusive der Kommunikation mit anderen Programmabschnitten

berechnet. Die errechneten Werte werden an das Fenster “current status” gesendet (Punkt 1). In diesem Fenster sieht der Nutzer in einem Diagramm den aktuellen Stand der Simulation (Abb.36). Danach wählt der Suchlauf den ersten Testpunkt aus. Er berechnet mit den Formeln 27 und 28 welche Kristalle diesen Testpunkt beeinflussen. Um die Intensität des Testpunktes anzupassen, muss der Suchalgorithmus nun bestimmen, in welche Richtung der Kristall gedreht werden muss. Hierzu wird der Kristall um die minimale Winkelschrittweite gedreht. Es wird überprüft, ob die Abweichung zum gewünschten Intensitätsprofil größer oder kleiner wird. Wenn die Abweichung kleiner wird, wird die verwendete Richtung als die Drehrichtung definiert. Wenn die Abweichung größer wird, wird die entgegengesetzte Richtung als Drehrichtung definiert. Der Winkel wird nach diesem Vorgang wieder auf den ursprünglichen Wert zurückgesetzt.

Nun wird geprüft, ob die Höhe der Testpunkte mit der geforderten Pulsform übereinstimmt (Punkt 2). Es werden sequentiell alle Testpunkte überprüft, ob sie die geforderte Intensität haben. Sollte einer der Testpunkte nicht mit der geforderten Pulsform übereinstimmen, wird die Überprüfung abgebrochen. Es wird dann mit Punkt 3 fortgefahren. Wenn alle Testpunkte mit der geforderten Pulsform übereinstimmen, wird die gesamte Pulsform berechnet und im Hauptfenster dargestellt. Der automatische Suchlauf ist an diesem Punkt beendet.

Beim Übergang von Punkt 2 zu 3 wird die interne Genauigkeit auf 50 % gesetzt. An Punkt 3 wird der ausgewählte Kristall gedreht. Nach der Drehung müssen verschiedene Bedingungen überprüft werden (Abb.38).

Zunächst wird geprüft, ob durch die Drehung die Grenzen für den Drehwinkel des Kristalls überschritten werden (Punkt 4). Wenn die Grenzen nicht überschritten werden, erfolgt nun die Berechnung der Pulsform an den Testpunkten. Der Suchlauf wird mit Punkt 5 fortgesetzt. Wenn die Grenzen überschritten sind, wird der Drehwinkel auf den, vom Nutzer definierten, maximal möglichen Wert zurückgesetzt. Es erfolgt der Plot der Daten (Punkt 1). Die Drehrichtung wird nicht geändert, da die Abweichung vom geforderten Wert wieder größer werden würde.

Nun muss geprüft werden, ob die geforderte Genauigkeit erreicht wurde. Da die Schrittweite 1° beträgt, kann nicht sofort, mit der vom Nutzer geforderten Genauigkeit, gearbeitet werden. Diese wird durch die großen Winkelschritte nicht sofort erreicht. Deshalb wird zu Anfang eine interne Genauigkeit definiert. Durch die interne Genauigkeit von 50% ist gewährleistet, dass der Suchlauf trotz des großen Winkelschritts ein Ergebnis, in der Nähe des Zielwertes, findet. Es erfolgt nun eine Überprüfung, ob die interne Genauigkeit erreicht ist (Punkt 5). Wird die interne Genauigkeit an Punkt 5 nicht erreicht, fährt der Suchlauf mit Punkt 6 fort. Wenn die interne Genauigkeit erreicht ist, wird der Suchlauf bei Punkt 8 fortgesetzt.

Wenn der Suchlauf mit Punkt 6 fortfährt, erfolgt die Überprüfung, ob die Abweichung größer wird. Ist das nicht der Fall, wird mit der Drehung an Punkt 3 weitergemacht. Wenn die Abweichung vom Zielwert steigt, folgt eine Halbierung des Winkelschrittes. Hiernach wird mit Punkt 7 fortgefahren.

An Punkt 7 erfolgt die Überprüfung der minimalen Schrittweite. Ist der Winkelschritt größer als die minimale Schrittweite wird der Winkel zurückgesetzt und die Drehung fortgesetzt (Punkt 3). Durch das Zurücksetzen des Winkels wird verhindert, dass der Zielwert schon überschritten wurde und trotz der Verkleinerung des Winkelschritts nicht mehr erreichbar ist. Bei Erreichen der minimalen Schrittweite erfolgt als nächster Schritt der Plot der Daten (Punkt 1).

Wenn die interne Genauigkeit (Punkt 5) erreicht ist, erfolgt die Überprüfung der vom Nutzer geforderten Genauigkeit(Punkt 8). Ist die, vom Nutzer geforderte, Genauigkeit nicht erreicht, erfolgt die Halbierung des Winkelschritts und der internen Genauigkeit. Danach wird mit Punkt 7 fortgefahren. Ist die vom Nutzer definierte Genauigkeit erreicht, wird zum Plot der Daten in Echtzeit gesprungen (Punkt 1). Dieser Algorithmus wird solange durchgeführt, bis die definierte

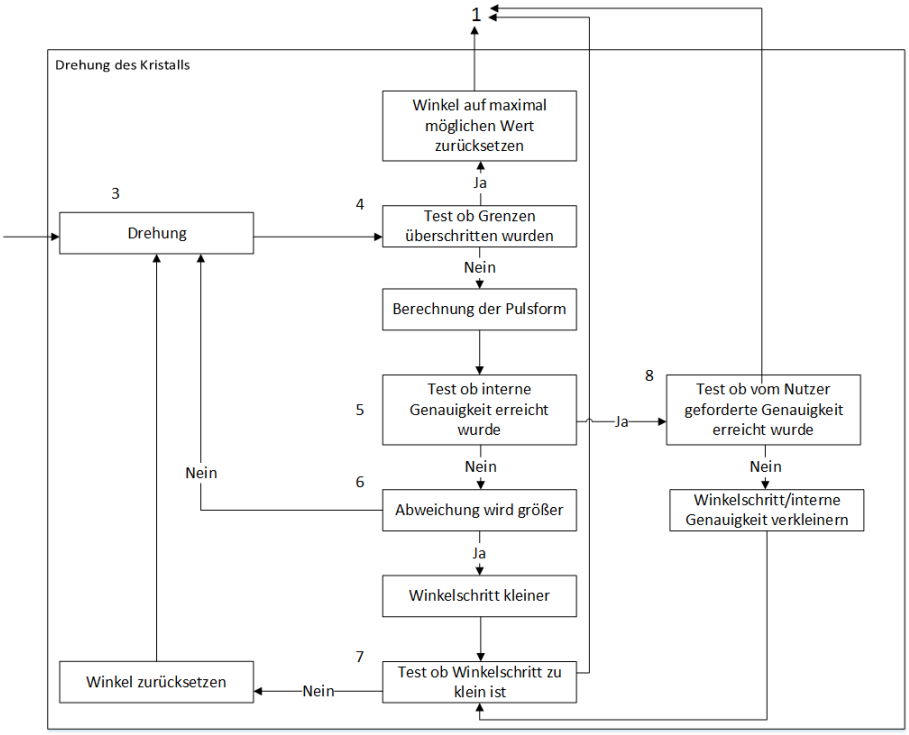


Abbildung 38: Funktionsweise des Suchlaufs zur Drehung des Kristalls und Überprüfung der Bedingungen

Pulsform mit der geforderten Genauigkeit erreicht wird (Punkt 2). Sollte die Pulsform nicht erreicht werden können, fällt der Suchlauf in eine Endlosschleife.

5 Planung von Messaufbauten zur Charakterisierung der Kristalle im Beamshaper des MBI-Lasersystems

Zur Simulation des Intensitätsprofils des Laserpulses im MBI-Lasersystem ist es notwendig, verschiedene Eigenschaften zu kennen. Manche dieser Eigenschaften sind bekannt, wie die Breite des Laserpulses am Eingang des Beamshapers. Andere Eigenschaften, wie die Zeitverzögerung oder die temperaturinduzierte Phasenverschiebung der Laserpulse, welche durch die Kristalle verursacht wird, sind nicht bekannt. Diese Eigenschaften müssen gemessen werden. Der Messaufbau und die geplanten Messungen werden in den folgenden Abschnitten beschrieben.

5.1 Planung des Messaufbaus

Im Rahmen dieser Arbeit wurde ein Messaufbau geplant, mit welchem die Eigenschaften der doppelbrechenden Kristalle bestimmt werden können. Der Messaufbau kann durch Modifikationen am Beamshaper des MBI-Lasersystem aufgebaut werden. In Abb.39 ist der zur Messung der Eigenschaften der Kristalle, geplante Messaufbau dargestellt. Zur Durchführung der Messung werden die

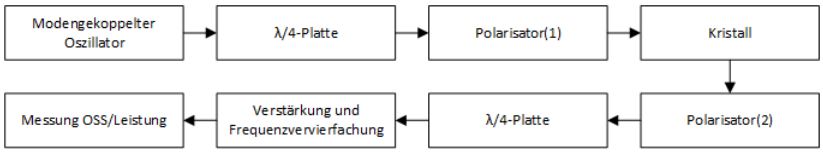


Abbildung 39: schematische Darstellung des Messaufbaus zur Bestimmung der Eigenschaften der Kristalle

Laserpulse aus dem modengekoppelten Oszillator durch eine $\lambda/4$-Platte geleitet. Durch die $\lambda/4$-Platte werden aus den linear polarisierten Laserpulsen zirkular polarisierte Laserpulse erzeugt. Die Pulse haben die gleiche Intensität für jeden Polarisationswinkel. Hiernach werden sie im Polarisator 1 linear polarisiert und so eine Polarisationsebene gewählt. Zur Messung der Kristalleigenschaften ist es notwendig, die Polarisationsebene der Pulse frei wählen zu können, da die Kristalle nur um $\pm 1^{\circ}$ drehbar sind. Die linear polarisierten Pulse werden durch den zu vermessenden doppelbrechenden Kristall des Beamshapers gelenkt und wieder linear polarisiert. Dies entspricht dem Aufbau eines Beamshaper aus einem Kristall. Nun ist das Intensitätsprofil der Laserpulse zur Bestimmung der Eigenschaften geformt. Hiernach folgt eine $\lambda/4$-Platte. Die Pulse werden zirkular polarisiert. Dies wird durchgeführt, da die Polarisationsebene nach der Formung der Pulse nicht der optimalen Polarisationsebene für die folgende Verstärkung in den Kristallen entspricht. Die Polarisationsebene muss hier nicht mit einem Polarisator ausgewählt werden, da in den Kristallen nur Licht einer bestimmten

Polarisationsebene verstärkt wird. Es wird dann die Leistung des Pulses oder das Intensitätsprofil im OSS gemessen. Zur Bestimmung der Kristallwinkel und der verursachten zeitlichen Verzögerung durch die Kristalle muss im OSS das Intensitätprofil gemessen werden. Bei der Messung der Phasenverschiebung, welche die Kristalle verursachen (Abschnitt 5.3), kann das Intensitätsprofil oder die transmittierte Leistung bestimmt werden. Die Messungen konnten noch nicht durchgeführt werden, da hierfür der Aufbau des Beamshaper verändert werden muss. Dies war während des momentanen Betriebs von PITZ nicht möglich.

5.2 Bestimmung der Kristallwinkel und der Verzögerung

Für die Simulation des Intensitätsprofils nach dem Beamshaper mit PulSi ist es notwendig, die Positionierung der Kristalle im Beamshaper und die durch sie erzeugte Phasenverschiebung zu kennen. Die Kristalle des Beamshapers sind mit Hilfe der bereits vorhandenen Software zur Steuerung der Kristalle um $\pm 1^\circ$ drehbar. Der Grund für die Begrenzung ist, dass die Schrittmotoren, welche die Kristalle drehen, nur einen Drehbereich von $\pm 1^\circ$ abdecken. Die minimale Schrittweite beträgt 0,001°. Im Steuerungsinterface wird nur die Verdrehung um die Ausgangsposition der Kristalle angegeben. Diese Werte können in PulSi nicht verwendet werden. Es ist deshalb notwendig, die Ausgangsposition der Kristalle im Beamshaper zu messen.

Hierzu wird der in Abschnitt 5.1 Abb.39 gezeigte Messaufbau verwendet. Der Polarisator 2 steht im 90° Winkel zum Polarisator 1. Werden nun beide Polarisatoren, unter Beibehaltung des 90° Winkels zueinander, gedreht, verändert sich die Intensität des geformten Pulses. Wenn die Intensität minimal wird, entsprechen die Winkel der Polarisatoren den Winkeln der optischen Achsen der Kristalle. Der Winkel des Polaristors 1 wird als 0° definiert. Nun werden beide Polaristoren auf einen Winkel von 45°, relativ zum eben definierten Winkel, eingestellt. Mit dem OSS kann nun eine Messung durchgeführt werden. In Abb.40 ist die Intensität des einfallenden Laserpulses auf beide Kopien (orange Kurven) gleichmäßig verteilt. Grund hierfür ist die Stellung der Polarisatoren, relativ zum Kristall. Als nächster Schritt wird der Polaristor 2 auf 0° oder 90° eingestellt. Eine der gaußförmigen Kopien wird so ausgewählt (Abb.41). Für die Verwendung der Messwerte in PulSi, muss der Winkel der schnellen Achse des Kristalls bestimmt werden. Der Winkel muss relativ zum Winkel des Eingangspolarisators im Betriebszustand des Beamshapers angegeben werden. Durch die Bestimmung des Abstands der beiden Gaußkurven, wird die durch den Kristall verursachte zeitliche Verzögerung der Kopien Δt berechnet(siehe Abb.40).

Da die beschriebenen Messungen nicht möglich waren, konnte nur der Winkel anhand der Fassungen der Kristalle bestimmt werden. Diese waren an den Kristallhalterungen ablesbar. Durch sie kann die Verdrehung, relativ zum Eingangspolarisator, bestimmt werden. Der Messfehler beträgt $\pm 1^\circ$, da die Skalierung auf den Halterungen in Schritten von 1° aufgetragen war. Man muss jedoch davon

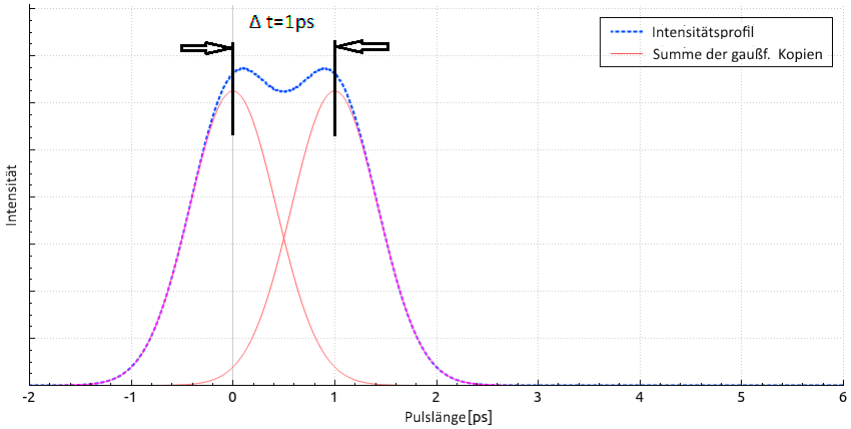


Abbildung 40: Grafische Darstellung des mit PulSi simulierten Intensitätsprofils für einen dopppelbrechenden Kristall zwischen 2 Polaristoren welche um 45° zu einer der optischen Achsen verdreht wurden

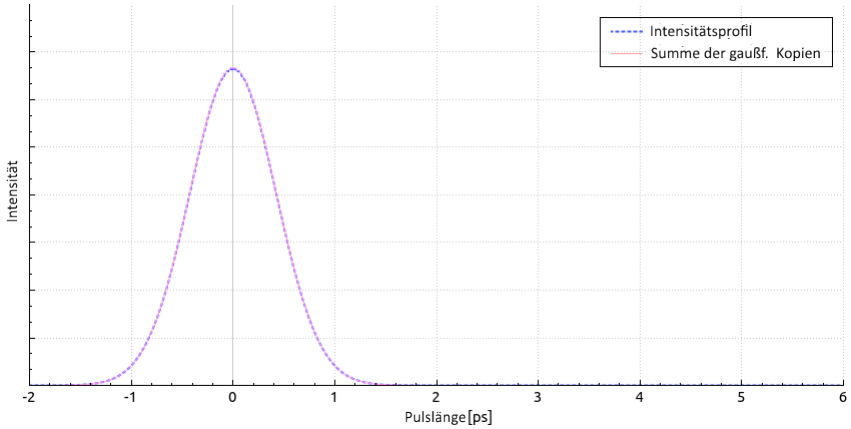


Abbildung 41: Grafische Darstellung des mit PulSi simulierten Ergebnisses für einen Kristall zwischen 2 Polarisatoren (Auswahl der schnellen optischen Achse des Kristalls)

ausgehen, dass die Kristalle, alle in der gleichen Weise, in die Fassung eingesetzt wurden. Die Ergebnisse der Messung sind in Tabelle 6 zu sehen.

Tabelle 6: Drehwinkel der Kristallfassungen

Kristall Nr.	Winkel [°]	Verdrehung relativ zum vorherigen Kristall[°]
1	4±1	–
2	6±1	2±1
3	13±1	7±1
4	24±1	11±1
5	34±1	10±1
6	48±1	14±1
7	61±1	13±1
8	76±1	15±1
9	86±1	10±1
10	96±1	10±1
11	102±1	6±1
12	104±1	2±1
13	101±1	-3±1

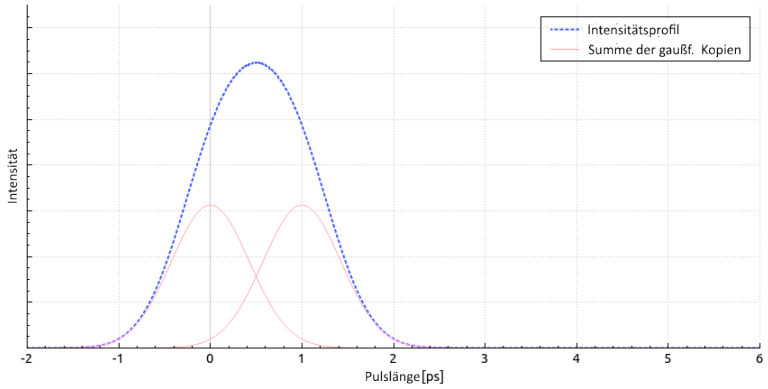
5.3 Messung der Phasenverschiebung

Die Phasenverschiebung wird im Beamshaper des MBI-Lasersystem durch die Regelung der Temperatur der Kristalle eingestellt. In PulSi wird jedoch die Phasenverschiebung ausgegeben. Wie in Abschnitt 3.2 beschrieben, ist es nicht sinnvoll, für einen realen doppelbrechenden Kristall, die aus der Temperatur resultierende Phasenverschiebung, zu berechnen. Zur Übertragung der Ergebnisse der Simulation auf den realen Beamshaper ist es jedoch notwendig, den Zusammenhang von Phasenverschiebung und Temperatur zu kennen. Es soll deshalb der Zusammenhang von Temperatur und Phasenverschiebung gemessen werden.

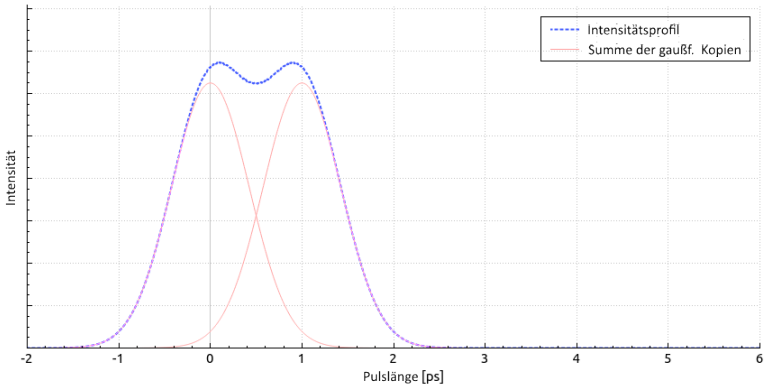
Um Schäden an der Steuerelektronik oder den Schrittmotoren zu verhindern, sollte die Messung nicht bei Kristalltemperaturen unterhalb der Zimmertemperatur durchgeführt werden. Es könnte sich Kondenswasser bilden. Zudem kommt es durch Kondenswasser auf den Kristallen zur Brechung oder Reflexion des Laserstahls. Das verfälscht die Messung. Das Ziel der Messung ist die Bestimmung der Höhe des Intensitätsprofils zwischen den gaußförmigen Kopien des Laserpulses.

Für die Messung wird der Messaufbau aus Abb.39 verwendet. Zur Durchführung der Messung werden die Polaristoren 1 und 2 auf einen Winkel von 0° gestellt. Der Kristall wird um 45° zum Polarisator am Eingang verdreht. Dadurch weisen beide gaußförmigen Kopien die gleiche Intensität auf. So kann die Höhe des Intensitätsprofils zwischen beiden Kopien gemessen werden. Durch die Änderung der Temperatur ändert sich der Brechungsindex und somit die Phasenverschiebung der Kopien zueinander. Als Folge ändert sich das Intensitätsprofil zwischen den Kopien. Das Intensitätsprofil kann im OSS gemessen werden. Da sich die Gesamtintensität des Laserpulses ändert kann auch die Leistung des Pulses gemessen werden. Für jede Temperatur wird das Intensitätprofil gemessen. Mit

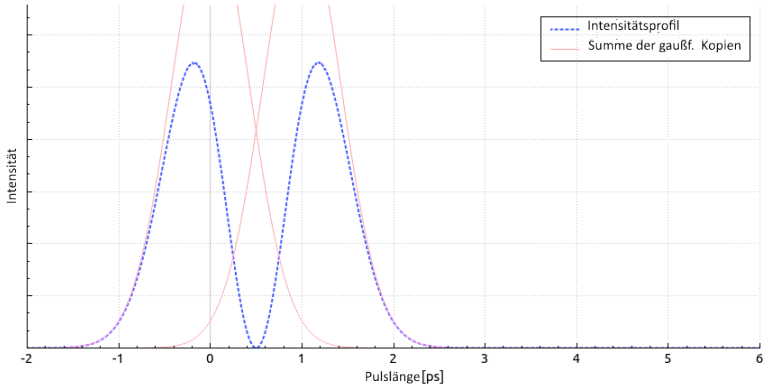
PulSi wurden die zu erwartenden Intensitätsprofile bei verschiedenen Phasenverschiebungen berechnet (Abb.42a-42c). Vergleicht man die gemessenen und die simulierten Profile miteinander, so kann ein Zusammenhang zwischen Phasenverschiebung der Kopien und Temperatur der Kristalle hergestellt werden.



(a) Phasenverschiebung von 0π



(b) Phasenverschiebung von $0{,}5\pi$



(c) Phasenverschiebung von 1π

Abbildung 42: Darstellung der Ergebnisse der von PulSi berechneten Pulsformen eines Kristalls zwischen 2 Polarisatoren im Winkel von 45° zu einer Kristallachse bei verschiedenen Phasenverschiebungen (zeitliche Verzögerung durch die Kristalle 1 ps)

6 Simulationen und Ergebnisse

Im Rahmen dieser Masterarbeit werden verschiedene Simulationen mit PulSi durchgeführt. Im folgenden Abschnitt werden die Laufzeitmessungen des automatischen Suchlaufs beschrieben. Außerdem werden die Ergebnisse von Suchläufen zum Finden definierter Intensitätsprofile gezeigt. Anhand von Flat-Top-Pofilen werden die Möglichkeiten der Optimierung der Pulsintensität und Pulsstruktur vorgestellt. Es wird zudem die Möglichkeit gezeigt, mit PulSi simulierte Intensitätsprofile, an vom OSS gemessene Profile, anzupassen. Hierbei werden die nichtlinearen Prozesse, welche den Laserpuls in seiner Form verändern, in der aktuellen Version von PulSi noch nicht simuliert.

6.1 Laufzeitmessungen

Bei PulSi handelt es sich um ein sehr rechenaufwendiges Programm. Es ist deshalb notwendig, die benötigte Zeit der Prozessschritte aller Programmteile zu minimieren. Dadurch können im Suchlauf, in der gleichen Zeit, weitere Parameter (z.B. Phasenverschiebung) automatisch optimiert werden. Zur Minimierung der Rechenzeiten ist es notwendig, die benötigte Zeit der einzelnen Programmfunktionen zu kennen. Durch die Verringerung der benötigten Zeit der zeitintensivsten Prozesse, kann die Rechenzeit des Programms effektiv reduziert werden. Hierzu werden die Messungen der Rechenzeit zum einen anhand der benötigten Zeit vom Start bis Ende des Suchlaufs, zum anderen anhand der benötigten CPU Takte gemessen. Zudem wurden die benötigten Iterationen zum Finden eines Intensitätsprofils gezählt. Die Auswahl eines Kristalls und dessen Drehung bis zur Auswahl des nächsten Kristalls wird als eine Iteration definiert.

PulSi kann aufgrund seiner Programmierung nur einen Kern des Prozessors nutzen. Dieser wird jedoch vollständig von PulSi genutzt. Somit kann die Anzahl der Clock per Second mit 1.000.000[2] angenommen werden.

Die Messungen werden mit Hilfe des automatischen Suchlaufs durchgeführt. Hierbei wurde von einem definierten Startprofil (Kristallwinkel nach Formel 25, Phasenverschiebung $0{,}5\pi$) aus versucht, die Kristallwinkel eines Flat-Top-Profils zu bestimmen. Es wird zum einen die benötigte Zeit gemessen. Hierzu wird von PulSi die Differenz des Zeitindex des Startzeitpunktes und des Endzeitpunktes des automatischen Suchlaufs berechnet. Die Zeitindizes werden mit der Funktion "gettimeofday" bestimmt.

$$Zeitindex\,Start - Zeitindex\,Ende = benötigte\,Zeit$$

Zum anderen wird die benötigte Rechenzeit berechnet. Hierzu werden die benötigten CPU Takte von Programm gezählt. Die benötigten CPU Takte können in die benötigte Zeit, mit der Formel:

$$\frac{Benötigte\,CPU\,Takte}{Clocks\,per\,Second} = benötigte\,Zeit,$$

[2] Maschinenbefehle pro Sekunde (Wert wird durch Bibliothek "time.h" ausgegeben)

umgerechnet werden. Die gesamte Zeit, welche der Suchlauf benötigt, kann durch die Formel:

$$t_{Suchlauf} = (t_{Iteration} + t_{Profil}) \cdot Iterationen \tag{29}$$

berechnet werden. Hierbei ist $t_{Suchlauf}$ die vom Suchlauf benötigte Zeit, $t_{Iteration}$ die für eine Iteration benötigte Zeit, t_{Profil} die zur Berechnung der Testpunkte benötigte Zeit und Iterationen sind die Anzahl der benötigten Iterationen. Unter der Voraussetzung, dass $t_{Profil} \gg t_{Iteration}$ gilt, lässt sich t_{Profil} durch die Formel $t_{Profil} = t_{Suchlauf}/Iterationen$ berechnen. In Tabelle 7 sind die Parameter

Tabelle 7: Parameter der Simulation zur Bestimmung der Rechenzeit.

Laserpulsbreite	1ps
Zeitverzögerung durch die Kristalle	1ps
Phasenverschiebung durch die Kristalle	$0,5\pi$
Pulsform	Flat-Top-Profil
Genauigkeit der Suche	1%

der Simulation, zur Suche nach einem Flat-Top-Profil, zu sehen. Zur Messung der Rechenzeit wurde die Anzahl der Kristalle zwischen einem und 13 Kristallen variiert. Die benötigte Zeit und die benötigten Takte werden im Programm gemessen und am Ende der Suche ausgegeben. Die Ergebnisse der Laufzeitmessungen des Suchlaufs sind in Abb.43 zu sehen. Man erkennt, dass die Rechenzeit exponentiell ansteigt. Die Messung der CPU-Takte und der Zeitdifferenz von Start-

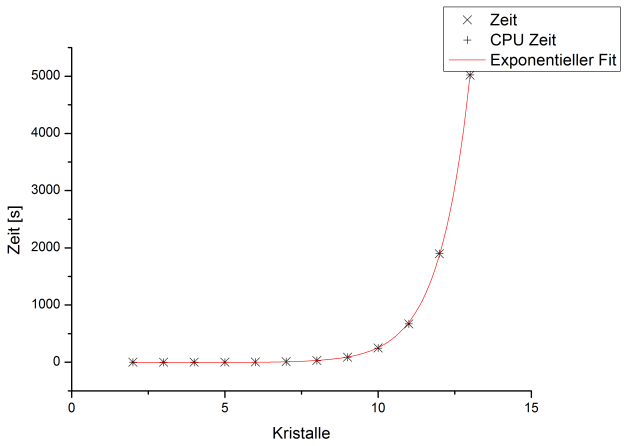


Abbildung 43: Vom automatischen Suchlauf benötigte Zeit zur Bestimmung der Kristallwinkel eines Flat-Top-Profils. Berechnet anhand der Zeitindizes (Kreuz) und der CPU-Takte (Plus), in Abhängigkeit der Anzahl der Kristalle und der exponentielle Fit der Daten(rote Kurve)

zu Endzeitpunkt des Suchlaufs ergeben das gleiche Ergebnis. Der exponentielle Fit ergibt die Funktion $t_{Suchlauf}(K_{ges}) = 0,014 \cdot \exp(-K_{ges}/1,02) - 5,16$. Die Anpassung stimmt sehr gut mit der Messung überein.

Da eine Iteration des Suchlaufs aus Abschnitt 4.3 wesentlich weniger Schritte benötigt (ca.10 Schritte), als die Berechnung der Testpunkte (Abschnitt 4.2) ab einer Kristallanzahl von 5 Kristallen, kann $t_{Profil} \gg t_{Iteration}$ angenommen werden. Durch die Summation über alle Kopien steigt der Rechenaufwand exponentiell an ($m_{ges} = 2^{K_{ges}}$). Nach Formel 29 kann nun die benötigte Zeit für die Berechnung der Testpunkte, in Abhängigkeit der Kristallanzahl, berechnet werden (Abb.43). Da die Kurve aus der in Abb.43 dargestellten Messung nach

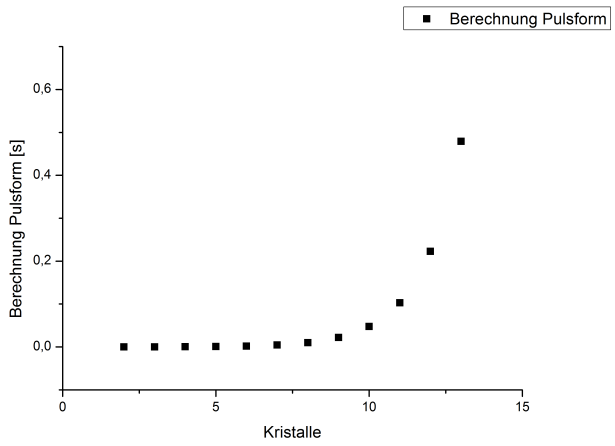


Abbildung 44: Benötigte Rechenzeit zur Berechnung der Messpunkte im Intensitätsprofil berechnet aus Abb.43

der Formel $t_{Profil} = t_{Suchlauf}/Iterationen$ berechnet wird, weist die Kurve die selbe Form auf, wie die Kurve der für die Simulation benötigten Gesamtzeit. Aus dieser Kurve kann die zur Berechnung der Testpunkte benötigte Zeit abgelesen werden.

In Abb. 45 sind die benötigten Iterationen über die Kristallanzahl aufgetragen und der Fit der Daten zu sehen. Die Funktion des Fits lautet: $Iteration(K_{Ges}) = 98,25 \cdot \exp(-K_{ges}/4,79) - 180,86$. Es ist zu sehen, dass der Fit nicht perfekt mit den Messwerten übereinstimmt.

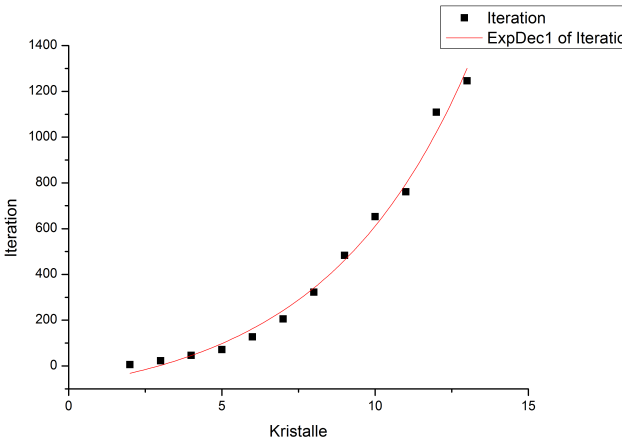
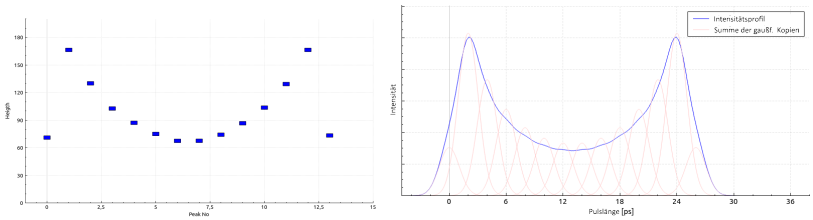


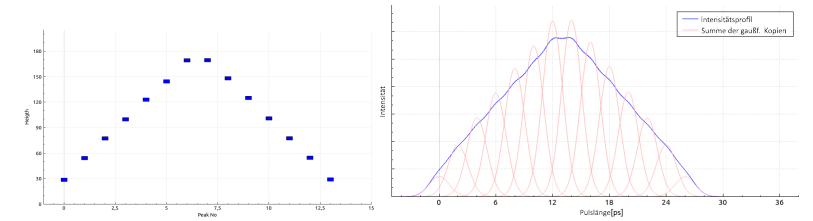
Abbildung 45: Die benötigten Iterationen des Suchlaufs zur Bestimmung der Kristallpositionen eines Intensitätsprofils (Quadrat) in Abhängigkeit von der Kristallanzahl und der exponentielle Fit der Daten(rote Kurve).

6.2 Simulation verschiedener Intensitätsprofile

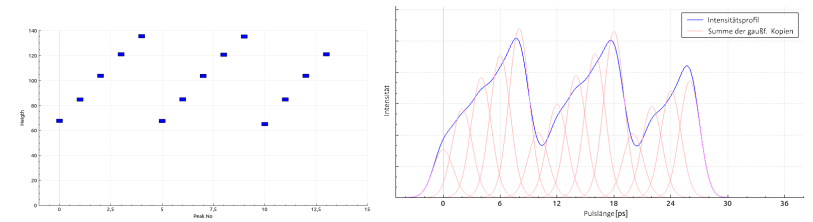
Mit PulSi ist es möglich, verschiedene Intensitätsprofile zu berechnen. Hierzu muss das Profil zunächst definiert werden. Die in den Abb. 46a, 46c, 46e, 46g zu sehenden Diagramme, sind die Vorgaben für den automatisch Suchlauf in PulSi. Diese Vorgaben wurden direkt aus dem Eingabefenster von PulSi kopiert und haben deshalb eine andere Skalierung der Achsen als die Ergebnisse der Simulation in Abb.46b, 46d, 46f, 46h auf der rechten Seite. PulSi kann somit über seine eigentliche Aufgabe hinaus auch andere Profile selbstständig simulieren.



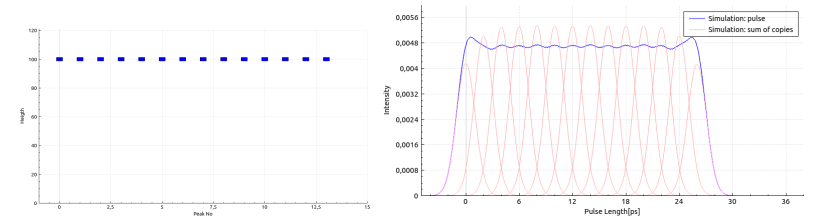
(a) Vorgabe Sattelform/SOLC-Filter (b) Sattelform/SOLC-Filter



(c) Vorgabe Pyramide (d) Pyramiden-Profil



(e) Vorgabe Sägezahn (f) Sägezahn-Profil



(g) Vorgabe Flat-Top-Profil (h) Flat-Top-Profil

Abbildung 46: Darstellung der Definition verschiedener zu suchender Pulsformen (Abb. 46a, 46c, 46e, 46g) und die mit PulSi daraus simulierten Profile (Abb. 46b, 46d, 46f, 46h).

6.3 Optimierung der Einstellungen eines simulierten Intensitätsprofils

Mit PulSi werden Simulationen zur Optimierung der transmittierten Intensität und der Struktur des Flat-Top-Bereichs eines Intensitätsprofils durchgeführt.

Hierzu wird zunächst die Abhängigkeit der Intensität von den Kristallwinkeln und der Phasenverschiebung betrachtet. Ein ausgewähltes Flat-Top-Profil wird dann, durch Anpassung der Phasenverschiebung, geglättet.

Zunächst wird ein Intensitätsprofil einer Gesamtpulslänge von 13 ps simuliert. Der Eingangslaserpuls hat eine Breite von 1 ps. Hier wird untersucht, wie sich der Winkel des Ausgangspolarisators auf die Intensität des Laserpulses auswirkt. Hierzu werden Flat-Top-Profile bei verschiedenen Winkeln des Ausgangspolarisators, mit dem automatischen Suchlauf, simuliert. Sobald ein Flat-Top-Profil gefunden wird, wird der Polarisator leicht gedreht und die gefundenen Kristallwinkel als neue Startwinkel für den neuen Suchlauf definiert. Die gefundenen Kristallwinkel sind in den Abb.47 und 48 zu sehen. Bei beiden Diagrammen

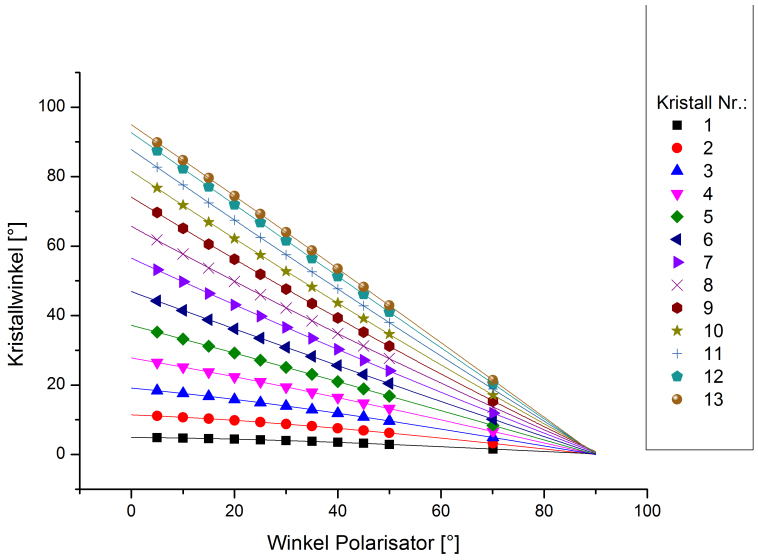


Abbildung 47: Flat-Top-Profil (Startkonfiguration 1): Winkel der Kristalle in Abhängigkeit zum Winkel des Ausgangspolarisators zur Formung eines Flat-Top-Profils

wurden unterschiedliche, zufällige Startwinkel verwendet. So wurde herausgefunden, dass es bei einem festen Winkel des Ausgangspolarisators mehrere Kristallstellungen gibt, welche ein Flat-Top-Profil erzeugen können. Es kann nicht ausgeschlossen werden, dass noch weitere Konfigurationen zur Erzeugung von Flat-Top-Profilen existieren. Es ist in Abb.47 und 48 zu erkennen, dass die Kristallwinkel immer näher beieinander liegen, je näher man einem Polarisatorwinkel von 90° kommt. Wenn 2 aufeinanderfolgende Kristalle den gleichen Winkel aufweisen, ist jedoch kein Flat-Top-Profil mit der vorgegebenen Länge mehr möglich. Es ist nicht möglich, manuell oder automatisch, bei einem Winkel des Ausgangspolarisators von 90°, ein Flat-Top-Profil zu finden. Es scheint

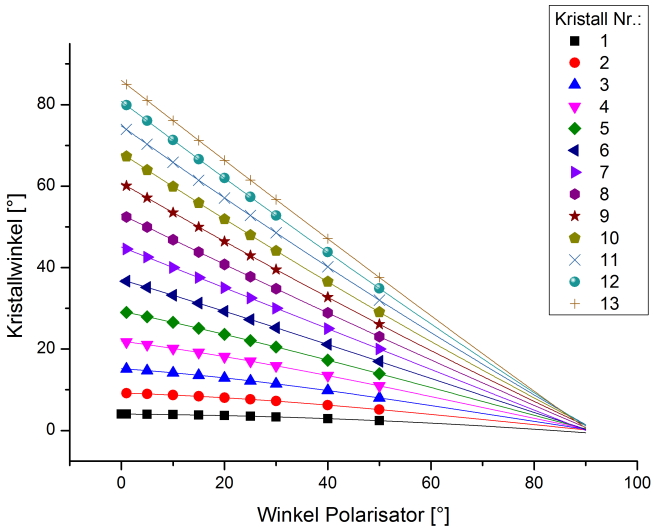


Abbildung 48: Flat-Top-Profil (Startkonfiguration 2): Winkel der Kristalle in Abhängigkeit zum Winkel des Ausgangspolarisators zur Formung eines Flat-Top-Profils

somit bei diesem Winkel kein Flat-Top-Profil möglich zu sein. Der Unterschied zwischen beiden Kristallkonfigurationen in Abb.47 und 48 ist in der Intensität des geformten Laserpulses zu finden(Abb.49). Es ist zu erkennen, dass die Intensität bei einem Winkel von 0° des Ausgangspolarisator ihr Maximum erreicht. Die Kristallkonfigurationen aus Abb.47 weisen, verglichen mit Abb.48, eine höhere Intensität auf. Es ist mit PulSi leider nicht auswählbar, welche dieser Konfigurationen man am Ende des Suchlaufs erhält, wenn man mit beliebigen Startwinkeln die Simulation startet. Hierzu müsste eine zu erreichende Intensität vorgegeben werden. Dies ist jedoch nicht vorgesehen, da es zu Fehlern im automatischen Suchlauf führt.

Es wurde außerdem der Einfluss der Phasenverschiebung auf die Intensität untersucht. Im hier definierten Beamshaper erzeugen die Kristalle eine Zeitverzögerung von 2 ps. Der Flat-Top-Bereich der geformten Laserpulse weist eine Länge von ca 2 ps*13=26 ps auf, was der maximalen Pulslänge, welche mit dem Beamshapers im MBI-Lasers geformt werden kann, entspricht[14]. Die Breite des gaußförmigen Laserpulses vor der Formung wird mit 2 ps definiert. In dieser Simulation werden Flat-Top-Profile simuliert. Die Phasenverschiebung wird für alle Kristalle auf den selben Wert gesetzt. Dies wird für verschiedene Phasenverschiebungen wiederholt.

Durch die Phasenverschiebung interferrieren die Kopien miteinander. Dies hat Einfluss auf die Höhe des geformten Intensitätsprofils. Es ist in Abb.50 zu se-

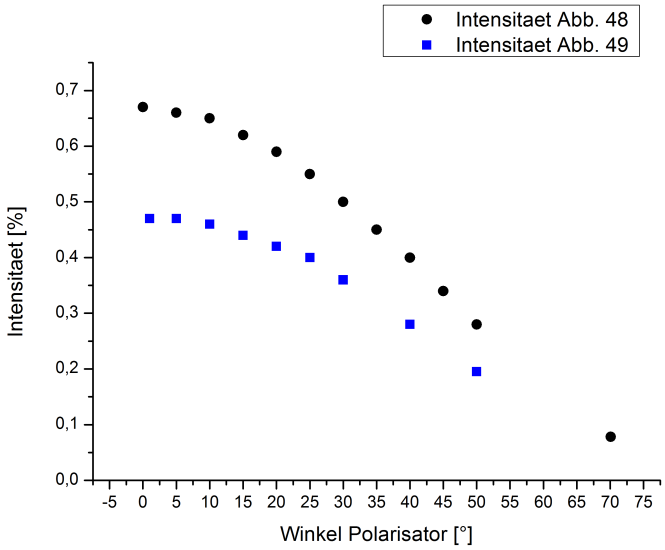


Abbildung 49: Intensitäten der unterschiedlichen Kristallkonfigurationen in Abb.47 (Kreis) und 48 (Quadrat) in Abhängigkeit zum Winkel des Ausgangspolarisators

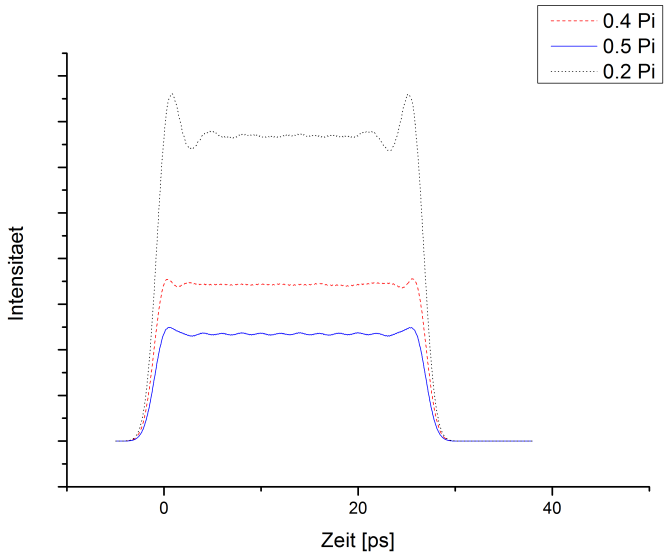


Abbildung 50: Flat-Top-Profile bei unterschiedlichen Phasenverschiebungen

hen, dass bei einer Phasenverschiebung von $0{,}2\pi$, die Intensität des Pulses am höchsten ist. Bei einer Phasenverschiebung von $0{,}2\pi$ ist die Mitte des Flat-Top-Bereichs sehr glatt, jedoch sind am Rand des Flat-Top-Bereichs große Maxima zu sehen. Diese können minimiert werden. Die Intensität des Flat-Top-Profils ist bei einer Phasenverschiebung von $0{,}5\pi$ geringer als bei $0{,}2\pi$. Der Grund ist, dass bei Vielfachen von $x = 2\pi n$ die Interferenz konstruktiv ist. Bei Vielfachen von $x = 2\pi n + 1$ ist sie destruktiv. Bei einer Phasenverschiebung von $0{,}5\pi$ ist die Interferenz mit den benachbarten Kopien minimal. Hier können die Unebenheiten nicht weiter optimiert werden, ohne neue an anderer Stelle zu erzeugen. Jedoch ist die Oberfläche sehr wellig.

Aufgrund der großen Intensität soll das Profil mit der Phasenverschiebung von $0{,}2\pi$ optimiert werden. Hierzu wird die Phasenverschiebung per Hand angepasst. Nach jeder Anpassung werden mit dem automatischen Suchlauf die Kristallwinkel korrigiert, sodass wieder ein Flat-Top-Profil entsteht. Dieser Ablauf wird solang durchgeführt, bis keine Verbesserung des Intensitätsprofils mehr feststellbar ist. Die Phasenverschiebungen und Kristallwinkel vor und nach der Optimierung sind in Tabelle 8 zu sehen. Nach der Optimierung der Ränder des

Tabelle 8: Kristallkonfigurationen vor und nach der Optimierung der Phasenverschiebung des Flat-Top-Profils aus Abb.51

Kristall Nr.	Kristallwinkel[°] vor Optimierung	Phasenverschiebung [π] vor Optimierung	Kristallwinkel[°] nach Optimierung	Phasenverschiebung [π] nach Optimierung
1	6,154	0,2	7,182	0,52
2	10,641	0,2	15,45	0,42
3	16,774	0,2	22,344	0,3
4	23,12	0,2	28,011	0,22
5	30,134	0,2	33,520	0,2
6	37,455	0,2	39,203	0,2
7	44,993	0,2	44,998	0,2
8	52,533	0,2	50,794	0,2
9	59,867	0,2	56,475	0,2
10	66,885	0,2	61,985	0,22
11	73,246	0,2	67,651	0,3
12	79,359	0,2	74,56	0,42
13	83,865	0,2	82,814	0,52

Intensitätsprofils (Abb.51) weist das Intensitätsprofil nahezu den selben RMS Wert (0,0093) auf, wie das Profil mit einer Phasenverschiebung von $0{,}5\pi$ (0,0078) in Abb.50 (siehe Tabelle 9). Die Intensität ist jedoch mehr als doppelt so groß. Im Vergleich zum Profil vor der Optimierung ist der RMS Wert (0,0509) stark gesunken . In Abb.51 kann man die Unterschiede, welche durch die Optimierung

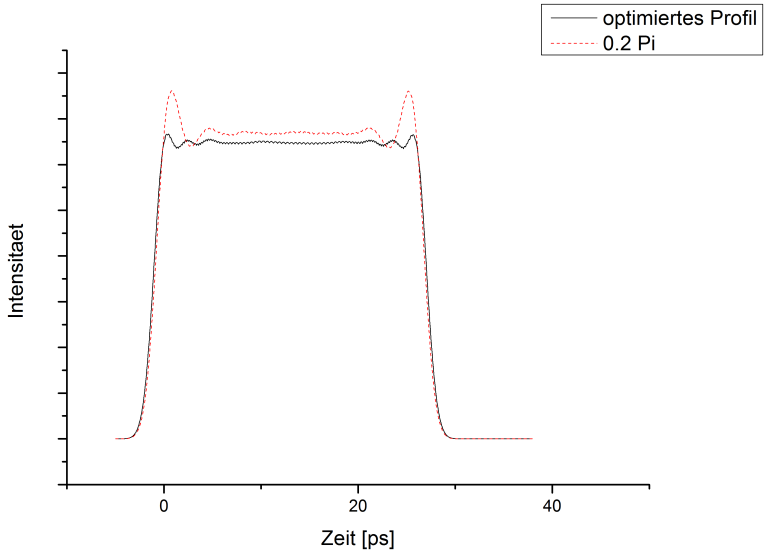


Abbildung 51: nicht optimiertes Flat-Top-Profil und optimiertes Flat-Top-Profil ausgehend vom Profil mit einer Phasenverschiebung von $0,2\pi$ in Abb.49

Tabelle 9: Charakterisierung der Intensitätprofile aus Abb.51

	Profil bei $0,5\pi$	nicht optimiertes Profil $(0,2\pi)$	optimiertes Profil(ausgehend von $0,2\pi$)
Höhe[% des Eingangspulses]	0,47	1.35	1.29
FWHM[ps]	28.15	27.69	28.25
RMS	0.0078	0.0509	0.0093

entstanden sind, sehen. Die Maxima am Rand wurden wesentlich verkleinert. Dafür ist jedoch auch die Gesamtintensität des Pulses um 0,06 % der transmittierten Intensität des Eingangspulses kleiner als vor der Optimierung. Da ein größerer Teil der Gesamtintensität in die Kopien am Rand verschoben wurde, wird der Laserpuls um ca. 0,5 ps länger. Relativ zur Gesamtlänge des Laserpulses ist diese Veränderung aber gering(ca. 2 %).

Durch die Annahme, die Kristalle würden jeweils eine Verzögerung von 2 ps verursachen und die Verlagerung einer höheren Intensität auf die Kopien am Rand des Profils, ist der Laserpuls ca. 3 ps länger als die theoretisch maximale Pulslänge des realen Beamshapers(25 ps FWHM). Durch die nachfolgenden nichtlinearen Kristalle werden die Flanken des Pulses weniger stark verstärkt.

Der Puls wird somit kürzer, wodurch eine Pulslänge von 26 ps erreicht wird. Für andere Pulslängen sind die selben Ergebnisse, mit geringen Anpassungen der Phasenverschiebung und der Kristallwinkel, erreichbar.

6.4 Anpassung von simulierten Intensitätsprofilen an OSS-Messungen

Mit PulSi wurde versucht, verschiedene Pulsformen anhand von früheren Messungen des OSS zu rekonstruieren. Es ist zu erwähnen, dass dies nur der Überprüfung der prinzipiellen Fähigkeit von PulSi dient, OSS-Daten zu laden und mit ihnen zu arbeiten. In der aktuellen Version kann nur das Profil im infraroten Bereich direkt nach Formung im Beamshaper simuliert und mit der Messung des Intensitätsprofils im OSS verglichen werden. Aufgrund der nichtlinearen Kristalle, zwischen dem Beamshaper und dem OSS, sind simuliertes und gemessenes Profil nur bedingt vergleichbar.

Für die Anpassung des simulierten an ein gemessenes Profil, werden Profile mit einem Flat-Top-Profil gewählt. Man muss die Struktur im Flat-Top-Bereich betrachten, da hier die Intensitätsabhängigkeit der nichtlinearen Effekte am wenigsten Auswirkungen hat. Grund hierfür ist die gleichmäßige Intensitätsverteilung. Diese bewirkt die gleichmäßige Verstärkung, welche das Profil kaum verändert. Die Profile wurden den Datenarchiven von PITZ entnommen. Die Daten werden geladen und es wird manuell versucht, das Profil zu rekonstruieren.

Tabelle 10: Einstellungen der Kristallwinkel und der Temperaturen der Kristalle des Beamshapers aus dem Logbuch

Kristall	Winkel[°]	Temperatur[°C]
1	0,524	25,4
2	0,11	35,123
3	-0,1	37,789
4	-0,067	23,405
5	-0,005	38,293
6	-0,129	38,624
7	-0,173	26,371
8	-0,307	41,646
9	-0,304	38,045
10	-0,142	36,63
11	-0,113	40,8
12	-0,231	40,211
13	-0,225	28,406

In PulSi wurde für die Simulation die Breite des Eingangslaserpulses zunächst mit 2,5 ps angenommen. Die zeitliche Verzögerung von 26 ps/13=2 ps je Kris-

tall, ausgehend von einer maximalen Pulslänge von 26 ps [14], ergibt ein Intensitätsprofil, welches wesentlich zu lang ist. Die zeitliche Verzögerung ist somit nicht bekannt. Sie wird aus den Messungen des OSS geschätzt. Während der Simulation wurde festgestellt, dass mit einer Verzögerung zwischen 1,5 und 1,6 die besten Ergebnisse erzielt wurden. In PulSi wurde deshalb eine Verzögerung von 1,5ps bzw. 1,6ps angenommen. Dies entspricht ebenfalls nicht den Spezifikationen des gaußförmigen Laserpulses im MBI-Lasersystem. In Abb.52 ist die Anpassung des Intensitätsprofils, an ein, mittels OSS gemessenes, Intensitätsprofil, zu sehen. Die Abweichung des simulierten Profils vom gemessenen Profil

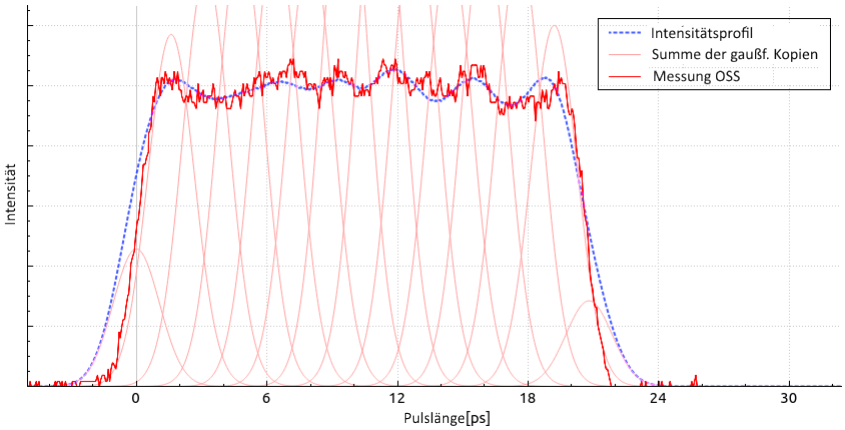


Abbildung 52: Anpassung an das gemessene Intensitätsprofil ausgehend von den gemessenen Kristallwinkeln in Tabelle 6 mit einer Breite des einfallenden Laserpulses von 1,6 ps

ist an den Flanken besonders groß. Im Flat-Top-Bereich stimmen beide Kurven gut überein. In Tabelle 11 sind die Drehwinkel und Phasenverschiebungen der Kristalle zu sehen, welche die Anpassung aus Abb.52 generieren. Da für die Anpassung Eigenschaften angenommen werden mussten, welche nicht den Spezifikationen des MBI-Lasersystems entsprechen, entfällt eine Vergleich der simulierten Drehwinkel der Kristalle mit den in Abschnitt 5.2 gemessenen Winkeln. Die Rekonstruktion des Intensitätsprofils mit einer Breite des einfallenden Laserpulses von 1,5ps (Abb.53) ergibt eine Anpassung, welche geringfügig besser auf das gemessene Profil passt.

Mit diesen Simulationen kann veranschaulicht werden, wie PulSi im MBI-Lasersystem eingesetzt werden kann. Es wurde gezeigt, dass das simulierte Intensitätsprofil an, im OSS gemessene, Intensitätsprofile angepasst werden kann. Die Abweichung der gemessenen Profile von den simulierten Profilen ist vermutlich auf die fehlende Berücksichtigung der nichtlinearen Prozesse zurückzuführen. Wenn die nichtlinearen Bauelemente von PulSi ebenfalls berücksichtigt werden, ist es somit möglich, aus Messungen des OSS die Einstellungen des Beamshapers zu rekonsturieren. So können Optimierungen der Einstellungen des Beamsha-

Tabelle 11: Winkel des berechneten Intensitätsprofils in Abb.52

Kristall Nr.	Winkel[°]	Phasenverschiebung[π]	Verdrehung relativ zum vorherigen Kristall[°]
1	2,13	0,51	
2	6,9	0,51	4,77
3	12,9	0,51	6
4	19,85	0,49	6.95
5	27,43	0,5	7,58
6	35,75	0,5	8,32
7	44,7	0,52	8,95
8	53,65	0,5	8,95
9	61,84	0,49	8,19
10	69,53	0,5	7,69
11	76,5	0,5	6,97
12	82,48	0,5	5,98
13	87,3	0,49	4,82

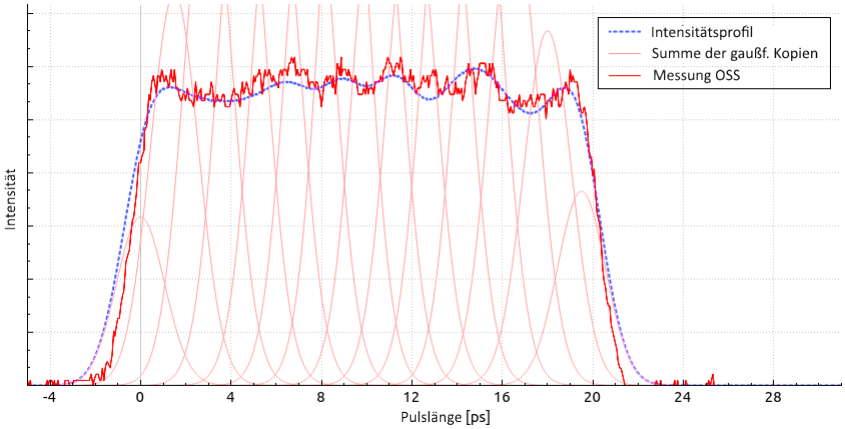


Abbildung 53: Anpassung an das gemessene Intensitätsprofil ausgehend von den gemessenen Kristallwinkeln in Tabelle 6 mit einer Breite des einfallenden Laserpulses von 1,5 ps

pers ohne Eingriff in den laufenden Betrieb durchgeführt werden.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde das Programm PulSi (Pulseshaper Simulation) geschrieben. Mit PulSi kann das Intensitätsprofil simuliert werden, welches in einem Beamshaper geformt wird. Das Design der Beamshaper muss dem Beamshaper im MBI-Lasersystem, in der Gruppe PITZ am DESY Zeuthen, entsprechen. Für das Programm wurde eine grafische Benutzeroberfläche entwickelt. Über diese können die Einstellungen des Beamshapers geändert und der Suchlauf gesteuert werden. Es wurde zudem ein Berechnungsalgorithmus zur Berechnung des Intensitätsprofils und ein Suchalgorithmus entwickelt. Mit dem Suchalgorithmus können, vom Nutzer definierte, Intensitätsprofile automatisch gesucht werden. Für diese Profile gibt PulSi die erforderlichen Einstellungen am Beamshaper an. Der Nutzer kann zudem mit PulSi manuell das simulierte Intensitätsprofil verändern. Die erzeugten Intensitätsprofile werden in einem Diagramm dargestellt.

Es wurden Planungen zur genauen Charakterisierung der Kristalle im Beamshaper durchgeführt. Hierzu wurde ein Messaufbau entwickelt. Es wurde die Messung der Kristallwinkel und die Messung, der durch die Kristalle verursachten, zeitlichen Verzögerung der Laserpulse geplant. Durch die Verwendung dieser Messungen können die Ergebnisse von PulSi auf einen realen Beamshaper übertragen werden. Für die Übertragung, der in der Simulation verwendeten Phasenverschiebung auf den Beamshaper, soll der Zusammenhang zwischen Phasenverschiebung und Temperatur der Kristalle ebenfalls gemessen werden.

Es wurden verschiedene Simulationen mit PulSi durchgeführt. Hierbei wurden die Laufzeiten unterschiedlicher Programmteile bestimmt. Zudem wurde versucht, die Kristallstellungen zur Erzeugung verschiedener Intenstitätsprofile zu simulieren. Es wurde untersucht, wie die transmittierte Intensität und die Struktur der Flat-Top-Profile optimiert werden können. Mit diesen Simulationen wurden gleichzeitig die Funktionen von PulSi getestet.

Bei den durchgeführten Laufzeitmessungen wurde versucht, die zeitintensivsten Programmteile zu bestimmen. Es stellte sich heraus, dass die Berechnung der Testpunkte die meiste Rechenzeit benötigt. Außerdem wurde festgestellt, dass die Anzahl der notwendigen Iterationen, zur Bestimmung der Kristallwinkel und die benötigte Rechenzeit, exponentiell mit der Kristallanzahl steigt.

Es wurde zudem gezeigt, dass PulSi mit dem momentan verwendeten Suchlauf verschieden geformte Intensitätsprofile suchen und finden kann.

Da im Beamshaper bei PITZ hauptsächlich Flat-Top-Profile simuliert werden, wurde darauf hin versucht, die Möglichkeiten zur Optimierung eines Flat-Top-Profil zu bestimmen. Hierbei wurde herausgefunden, dass die meiste Intensität bei einem Winkel des Ausgangspolarisator von 0° transmittiert wird. Bei gleichem Winkel des Ausgangpolarisators gibt es verschiedene Kombinationen von Kristallwinkeln, welche das gleiche Intensitätsprofil generieren. Diese unterschiedlichen Kombinationen weisen verschiedene transmittierte Intensitäten

auf. Es wurde außerdem die Intensität des geformten Pulses, bei verschiedenen Phasenverschiebungen durch die Kristalle, bestimmt. Hierbei zeigte sich, dass die Intensität steigt, je mehr man sich der Phasenverschiebung von 0π nähert. Außerdem wurde gezeigt, dass die Struktur des Laserpulses durch Anpassung der Phasenverschiebung, geglättet werden kann.

Es wurde zudem gezeigt, dass mit PulSi Messungen aus dem OSS geladen werden können. Das simulierte Profil kann an das gemessene Profil angepasst werden. Dies wurde nur zu Demonstrationszwecken durchgeführt, da PulSi noch nicht die Einflüsse der nichtlinearen Bauelemente zwischen Beamshaper und OSS simulieren kann. Messungen und Simulationen sind somit nur bedingt vergleichbar.

PulSi besitzt Möglichkeiten der Weiterentwicklung. Es wurden Überlegungen angestellt, den realen Beamshaper bei PITZ, in späteren Entwicklungsversionen von PulSi vollständig mit PulSi steuern zu können. Hierzu muss der automatische Suchlauf Feedback von den Messungen des OSS bekommen und die Kristalle selbstständig steuern. Für diese Steuerung wurden im Quellcode bereits grundlegende Strukturen angelegt.

Literatur

[1] Cristian Gerthsen , Dieter Meschede. *Gerthsen Physik 23. Auflage*. Springer, 2005.

[2] M Khojoyan, M Krasilnikov, A Oppelt, F Stephan, et al. Simulation studies of generating ultra short pulses at pitz. PAC, 2011.

[3] Frank Stephan and Mikhail Krasilnikov. High brightness photo injectors for brilliant light sources. In Eberhard Jaeschke, Shaukat Khan, Jochen R. Schneider, and Jerome B. Hastings, editors, *Synchrotron Light Sources and Free-Electron Lasers*, pages 1–38. Springer International Publishing, 2014.

[4] M. Krasilnikov, F. Stephan, G. Asova, H.-J. Grabosch, M. Groß, L. Hakobyan, I. Isaev, Y. Ivanisenko, L. Jachmann, M. Khojoyan, G. Klemz, W. Köhler, M. Mahgoub, D. Malyutin, M. Nozdrin, A. Oppelt, M. Otevrel, B. Petrosyan, S. Rimjaem, A. Shapovalov, G. Vashchenko, S. Weidinger, R. Wenndorff, K. Flöttmann, M. Hoffmann, S. Lederer, H. Schlarb, S. Schreiber, I. Templin, I. Will, V. Paramonov, and D. Richter. Experimentally minimized beam emittance from an l-band photoinjector. *Phys. Rev. ST Accel. Beams*, 15:100701, Oct 2012.

[5] Ingo Will, Guido Klemz. *Generation of flat-top picosecond pulses by coherent pulse stacking in a multicrystal birefringent filter*. Optics Express, 2008.

[6] R. Clark Jones. *'A New Calculus for the Treatment of Optical Systems, I. Description and Discussion of the Calculus', J. Opt. Soc. Am. 31, 488-493*. Optical Society of America, 1941.

[7] Jenny P. Glusker, Mitchell Lewis, Miriam Rossi. *Crystal Structure Analysis for Chemists and Biologitsts*. John Wiley & Sons, 1994.

[8] N. Ter-Gabrielyan, V. Fromzel, and M. Dubinskii. Linear thermal expansion and thermo-optic coefficients of yvo 4 crystals the 80-320 k temperature range. *Optical Materials Express*, 2(11):1624–1631, 2012.

[9] *Nichtlineare Optik*. http://www.wmi.badw.de/teaching/Lecturenotes/Physik3/Gross_Physik_III_Kap_8.pdf.

[10] Prof. Dr. Roland Sauerbrey. *Nichtlineare Optik, Script TU Dresden*. 2007. http://www.hzdr.de/FWT/vorlesung_ss07/NichtlineareOptik_I.pdf.

[11] Alexander A. Kaminskii, Ken ichi Ueda, Hans J. Eichler, Yasuhiko Kuwano, Hikaru Kouta, Sergei N. Bagaev, Thomas H. Chyba, James C. Barnes, Gad M.A. Gad, Tomoyo Murai, and Jianren Lu. Tetragonal vanadates yvo_4 and $gdvo_4$ - new efficient χ^3-materials for raman lasers. *Optics Communications*, 194(1-3):201 – 206, 2001.

[12] Yoichi Sato and Takunori Taira. Highly accurate interferometric evaluation of thermal expansion and dn/dt of optical materials. *Optical Materials Express*, 4(5):876–888, 2014.

[13] Pavel A Loiko, Konstantin V Yumashev, Vladimir N Matrosov, and Nikolai V Kuleshov. Dispersion and anisotropy of thermo-optic coefficients in tetragonal gdvo 4 and yvo 4 laser host crystals. *Applied optics*, 52(4):698–705, 2013.

[14] M Krasilnikov, J Bähr, M Hänel, F Stephan, and I Will. Experimental optimization of the cathode laser temporal profile. In *Proceedings of DIPAC*, 2007.

[15] Dr. Rüdiger Paschotta. *Quantum Defect.* http://www.rp-photonics.com/quantum_defect.html.

[16] Dr. Rüdiger Paschotta. *Ytterbium-doped Gain Media.* http://www.rp-photonics.com/ytterbium_doped_gain_media.htmll.

[17] P. Lacovara et al. *Room-temperature diode-pumped Yb:YAG laser.* Opt. Lett. 16 (14), 1991.

[18] Takunori Taira, William M. Tulloch, Robert L. Byer. *Modeling of quasi-three-level lasers and operation of cw YB:YAG.* Applied Optics, 1997. http://www.stanford.edu/ rlbyer/PDF_AllPubs/1997/324.pdf.

[19] *Wellenausbreitung in optisch anisotropen Stoffen.* http://bilder.buecher.de/zusatz/12/12549/12549933_lese_1.pdf.

[20] William F. Krupke, L. L. Chase. *Ground-state depleted solid-state lasers: priciples, characteristics ans scaling.* Optical and Quantum Electronics, 1990.

[21] R. Diehl. *High Power Diode Lasers Fundamentals,Technology, Applications.* Springer, 2000.

[22] Bahaa E. A. Saleh, Malvin Carl Teich. *Grundlagen der Photonik.* WILEY-VCH, 2008.

[23] Marc Eichhorn. *Laserphysik, Grundlagen und Anwendungen für Physiker, Maschinenbauer und Ingenieure.* Springer Spektrum, 2013.

[24] Gerd Habenicht. *Kleben Grundlagen,Technologien, Anwendungen.* Springer, 2009.

[25] Jonathan Elegheert, Nathalie Bracke, Philippe Pouliot, Irina Gutsche, Alexander V Shkumatov, Nicolas Tarbouriech. Allosteric competitive inactivation of hematopoietic csf-1 signaling by the viral decoy receptor barf1. 2012.

[26] N. V. Kuleshov, A. A. Lagatsky, A. V. Podlipensky, V. P. Mikhailov, and G. Huber. Pulsed laser operation of yb-doped ky(wo4)2 and kgd(wo4)2. *Opt. Lett.*, 22(17):1317–1319, Sep 1997.

8 Anhang

A Dateiformat der gespeicherten Kristalleinstellungen

```
number of crystals:     13
FWHM:   2,500000
delay:          1,850000
error:  0,001000
angle last polarizer:   0,0000000000000000
                angle[rad]              angle[Grad]             phaseshift
crystal No.1    0,1253550147273676      7,1823132846784397      1,6336281798666925
crystal No.2    0,2694039770963252      15,4357108716585305     1,3194689145077130
crystal No.3    0,3899925761680482      22,3449286558634519     0,9424780000000000
crystal No.4    0,4888972069074375      28,0117465715303275     0,6911500000000000
crystal No.5    0,5850502419533873      33,5209096670367401     0,6283190000000000
crystal No.6    0,6842222909335122      39,2030495192625779     0,6283190000000000
crystal No.7    0,7853693581759259      44,9983495823788289     0,6283190000000000
crystal No.8    0,8865228191690124      50,7940159804238931     0,6283190000000000
crystal No.9    0,9856831517972849      56,4754845351372978     0,6283190000000000
crystal No.10   1,0818346820402069      61,9845614117812218     0,6911500000000000
crystal No.11   1,1807248747979451      67,6505520920348005     0,9424780000000000
crystal No.12   1,3013150293958435      74,5598590013244973     1,3194689145077130
crystal No.13   1,4453682970535258      82,8135032631780916     1,6336281798666925
```

A Quelltext

A.1 main.cpp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <cmath>
4  #include <iostream>
5  #include <vector>
6  #include "global.h"
7
8
9  using std::cin;
10 using std::cout;
11 using std::vector;
12
13
14 #include <QtWidgets/QApplication>
15 #include "mainwindow.h"
16
```

```
17 int main(int argc, char *argv[])
18 {
19     QApplication a(argc, argv);
20     MainWindow Window;
21     Window.show();
22
23       return a.exec();
24
25     return 0;
26
27 }
```

A.2 global.h

```
1  #include "QMutex"
2
3  #define maxkrist 20                    //maximal number of
       crystals
4  #define maxkopie 1048576               //maximal number of
       copies(=2^maxkrist!!!!!!!)
5  #define cgaus 0.8493218                // constant of
       gaussian funktion
6  #define Pi 3.14159265358979323846
7  #define lichtgeschwindigkeit 299792458  // speed of light
8  #define dthetamin 0.0000017453         // minimal width of
       the angle steps
9
10 #ifndef uebergabe
11 #define uebergabe
12
13 typedef struct{
14     bool einheit;                      // input in radian
          measure(true)/degre(false)
15     bool initalisiert;                 // to prove variables
           are defined
16     bool ossdatenplotten;              // display OSS graph
17     bool pulshoehe_anzeigen;           // show heigth of
          pulses in % of the input pulse
18     bool jones;                        // which algorith
          should be used (always true in final version
          !!!!!!!)
19     bool Brechungsindex;               // input of
          refraction index (true) or phase shift(false)
20     bool plotausrichtung;              // search for highest
           point in envelopping function(false) or gaussian
          pulses (true)
```

```
21      bool dateiladen;                    // file loaded
22      int ende;                           // internal break
            value for simulation
23      int Abbruch;                        // internal break
            value for simulation
24      int nr;                             // number of crystal
            chosen to change angle by clicking on buttons
25      int row;                            // row of the table
26      int col;                            // column of the
            table
27      int endeExtern;                     // stop simulation by
             pressing button
28      int counter;                         // counter for
            testing
29      double A[maxkopie][maxkrist];        // possible ways
            the pulses could go through the crystals
30      double tk[maxkopie];                 // delay time of
            copies
31      double abw;
32      double Genauigkeit;                 // acuracy of the
            simulation
33      double breite;                      // width of gausian
            pulse for saving settings
34      double schritt;                     // width of steps on
            x axis
35  } Uebergabe;
36
37  typedef struct{
38      int signumE[maxkrist+1][2];                 // search for
             conversion (old value and new value)
39      double Intensitaet[maxkopie];               // heigth of
            the copies of the input pulse
40      double E[maxkrist+1];                       // heigth of
            the envelopping function on the position of the
            gausian pulses
41      double dh;                                  // average
            heigth of the peaks
42      double Perm[maxkrist+1];                    // number of
            copies on position of a gaussian pulse
43      double tau;                                 // width of
            the input pulse
44      double plotx[1000];                         // x
            component for graphs
45      double ploty[1000];                         // y
            component for graphs
46      double max;                                 // maximal
```

```
        heigth of graph in plot
47  //    int test;                                  // test
        value
48      double wellenlaenge;                       // wavelenght
            of laser light
49      double kreisfrequenz;                      // radian
            measure of the light
50      double hoehe[maxkrist+1];                  // intended
            heigth of the envelopping function on the postion
            of gaussian pulses relstivly to the average heigth
51  } Laserpuls;
52
53  typedef struct{
54      double Theta[maxkrist+1];                  // angle of
            the crystal relativly the input polarizer
55      double Winkel_Grenzen[maxkrist+1][2];      // minimum
            angle[..][0];maximum angle[..][1]
56      double dtheta;                             // width of
            the angle steps
57  } Winkel;
58
59  typedef struct{
60      bool polarisator[maxkrist];                //
            polarizer after crystals (true = polarizer/false=
            no polarizer)
61      int kristalle;                             // number of
            crystals
62      int kopie;                                 // number of
            copies of the input pulse
63      double dt;                                 // delay time
             of the ordinary wva to the extraordinary wave
64      double Dickekristall;                      // thickness
            of the crystal
65      double brechungsindizes[2];                // refraction
             indices of both axes   1. ordinary axis   2.
            extraordinary axis
66      double eingabePhasenverschiebung[maxkrist];        //
             Phaseshift by the crystal
67      double Phasenverschiebung[maxkopie];       // phase
            shift of the wavefunction of the copies
68      double winkel_polarisator[maxkrist];       // angle of
            polarizers
69  } Kristalle;
70
71  // for oss data
72  typedef struct{
```

```
73      bool geladen;                                   // have OSS
            files been loaded(true= yes)
74      int ossmesspunkte;                              // number of
            OSS points
75      double ossx[1000];                              // Oss-value
            x axis
76      double ossy[1000];                              // oss-value
            y axis
77      double ausgangspulshoehe;                       // heigth of
            the input pulse
78      double mitte;                                   // shift the
            oss graph
79  }OSSDatenuebergabe;
80
81  // for plot during simulation
82  typedef struct{
83      int maxpeak;                                    // smallest
            peak
84      int minpeak;                                    // highest
            peak
85      double e[maxkrist+1];                           // height of
            enveloping function un position of gaussian peaks
86  }zeit;
87
88
89  #endif
```

A.3 Funktionen.h

```
1  #include "fstream"
2  #include "global.h"
3  #include "cmath"
4  #include "iostream"
5  #include "vector"
6  #include "QMutex"
7  #include "iomanip"
8
9
10
11
12 //definitions
13
14 inline void Definition(Uebergabe *ueber, Kristalle *krist
       , Laserpuls *puls, Winkel *w)
15 {
16     double id=1;
```

```
17      int h[maxkrist],l=0;
18
19      w->dtheta=0.0174532925199;
20      krist->kopie=pow(2,krist->kristalle);
21
22      //phase shift
23      for(int i=0;i<krist->kristalle;++i)
24      {
25          if(ueber->Brechungsindex==true)
26          {
27              krist->eingabePhasenverschiebung[i]=Pi*krist
                    ->Dickekristall*(krist->brechungsindizes
                    [1]-krist->brechungsindizes[0])/puls->
                    wellenlaenge;
28          }
29      }
30
31      //angular frequency
32
33      puls->kreisfrequenz=2*Pi*lichtgeschwindigkeit/puls->
            wellenlaenge;
34
35      //define vectors
36      for(int i=0; i<=krist->kristalle;++i)
37      {
38          for(int j=0;j<2;++j)
39          {
40              puls->signumE[i][j]=1;
41          }
42      }
43
44      for(int j=0; j<=krist->kopie-1;++j)
45      {
46          for(int i=0; i<=krist->kristalle-1;++i)
47          {
48              ueber->A[j][i]=0;
49          }
50      }
51
52      for(int i=0; i<=krist->kristalle-1;++i)
53      {
54          h[i]=0;
55      }
56
57      for(int i=0;i<2;++i)
58      {
```

```
59          for(int j=0;j<=krist->kristalle;++j)
60          {
61              puls->signumE[j][i]=0;
62          }
63      }
64
65      //matrix with every way the input pulse could take
            through the crystall
66
67      for(int j=0;j<krist->kopie;++j)
68      {
69          //      cout<<j<<"\t";
70          l=0;
71          if(h[l] == 0)
72          {
73              for(int i=0;i<=krist->kristalle-1;++i)
74              {
75                  ueber->A[j][i]=h[i];
76
77                  //std::cout<<ueber->A[j][i]<<"\t";
78              }
79              h[l]=1;
80          }
81          else
82          {
83              for(int i=0;i<=krist->kristalle-1;++i)
84              {
85                  ueber->A[j][i]=h[i];
86                  //std::cout<<ueber->A[j][i]<<"\t";
87              }
88              if(h[l+1] == 0)
89              {
90                  h[l+1]=1;
91                  h[l]=0;
92              }
93              else
94              {
95                  for(int i=l;i<krist->kristalle-1;++i)
96                  {
97                      h[i]=0;
98                      if(i+1<krist->kristalle)
99                      {
100                         if(h[i+1]==0)
101                         {
102                             h[i+1]=1;
103                             break;
```

```
104                             }
105                         }
106                     }
107                 }
108             }
109             //std::cout<<"\n";
110         }
111
112         //define starting angle (angle of solc filter)
113         if(ueber->dateiladen==false)
114         {
115             for(int i=1;i<=krist->kristalle;++i)
116             {
117                 w->Theta[i]=Pi*2*45/360/krist->kristalle*(2*
                        id-1);
118                 id++;
119             }
120         }
121         for(int i=0;i<=krist->kristalle-1;++i)
122         {
123             krist->polarisator[i-1]=false;
124         }
125         // w->Theta[1]=1.4;
126
127         //Polarisator
128         krist->polarisator[krist->kristalle-1]=true;
129
130         //calculate delay an phase shift
131         for(int j=0;j<=krist->kopie-1;++j)
132         {
133             krist->Phasenverschiebung[j]=0;
134             ueber->tk[j]=0;
135             for(int i=0;i<=krist->kristalle-1;++i)
136             {
137                 ueber->tk[j]=ueber->tk[j]+ueber->A[j][i]*
                        krist->dt;
138                 krist->Phasenverschiebung[j]=krist->
                        Phasenverschiebung[j]+krist->
                        eingabePhasenverschiebung[i]*ueber->A[j][i
                        ];
139             }
140         }
141         if(krist->kristalle>0)
142         {
143             ueber->initalisiert=true;
144         }
```

```
145     else
146     {
147         ueber->initalisiert=false;
148     }
149 }
150
151
152
153 //gaussian funktion
154 inline float Gauss(/*Uebergabe *ueber, Kristalle *krist,
        */ Laserpuls *puls, /*Winkel *w,*/ double t,double
        erwartungswert)
155 {
156     float temp;
157     temp=exp((-1)*pow((t-erwartungswert),2)/pow((cgaus*
            puls->tau),2));
158     return temp;
159 }
160
161
162 //calculate intensity of copies
163 inline void Inten(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls, Winkel *w)
164 {
165     for(int j=0;j<=krist->kopie-1;++j)
166     {
167         double h[2];
168         //ueber->jones=true;
169
170
171         w->Theta[0]=0;
172         double Inr=1,Ini=0;
173         if(ueber->jones==true)
174         {
175             h[0]=1;
176             h[1]=0;
177             for(int i=0;i<=krist->kristalle-1;++i)
178             {
179                 if(ueber->A[j][i]==1)
180                 {
181                     Inr=pow(sin(w->Theta[i+1]),2)*h[0]-
                            cos(w->Theta[i+1])*sin(w->Theta[i
                            +1])*h[1];
182                     Ini=(-1)*sin(w->Theta[i+1])*cos(w->
                            Theta[i+1])*h[0]+pow(cos(w->Theta[
                            i+1]),2)*h[1];
```

```
183                     }
184                     if(ueber->A[j][i]==0)
185                     {
186                         Inr=pow(cos(w->Theta[i+1]),2)*h[0]+
                                cos(w->Theta[i+1])*sin(w->Theta[i
                                +1])*h[1];
187                         Ini=cos(w->Theta[i+1])*sin(w->Theta[i
                                +1])*h[0]+pow(sin(w->Theta[i+1])
                                ,2)*h[1];
188                     }
189                     //polarizer after crystal
190                     if(krist->polarisator[i]==true)
191                     {
192                         //std::cout<<Inr<<"\t"<<Ini<<"\t"<<
                                krist->winkel_polarisator[i]<<"\n
                                ";
193                         h[0]=Inr*cos(krist->
                                winkel_polarisator[i])-Ini*sin(
                                krist->winkel_polarisator[i]);
194                         h[1]=0;
195                         //std::cout<<h[0]<<"\n";
196                     }
197                     else
198                     {
199                         h[0]=Inr;
200                         h[1]=Ini;
201                     }
202                 }
203                 //std::cout<<"----------------"<<h[0]<<"\t"<<h
                        [1]<<"\n";
204             }
205             //puls->Intensitaet[j]=h[0]*cos(krist->
                    winkel_polarisator[krist->kristalle])-h[1]*sin
                    (krist->winkel_polarisator[krist->kristalle]);
206             puls->Intensitaet[j]=h[0];
207             //std::cout<<krist->Phasenverschiebung[0]<<"\t"<<
                    krist->Phasenverschiebung[1]<<"\n";
208             //std::cout<<krist->winkel_polarisator[krist->
                    kristalle]<<"\t"<<h[1]*sin(krist->
                    winkel_polarisator[krist->kristalle])<<"\t"<<h
                    [1]<<"\n";
209         }
210     }
211
212     //Berechnung Gesamtkurve
213
```

```
214 /*inline void Gesamtinten(Uebergabe *ueber, Kristalle *
        krist, Laserpuls *puls, Winkel *w)
215 {
216     Inten(ueber, krist, puls, w);
217
218     double zeit,zeitschritt;
219     //anfangszeit
220     zeit=(-1)*puls->tau*2;
221     zeitschritt=(krist->dt*(krist->kristalle+1)+2*puls->
            tau*2)/ueber->Stuetzstellen;
222     for(int i=0;i<ueber->Stuetzstellen;++i)
223     {
224         puls->feld[i]=0;
225         puls->zeitstempel[i]=zeit;
226         for(int j=0;j<krist->kopie;++j)
227         {
228             puls->feld[i]=puls->feld[i]+puls->Intensitaet
                    [j]*Gauss(ueber, krist, puls, w, zeit,
                    ueber->tk[j])*cos(puls->kreisfrequenz*zeit
                    /1000000000000+krist->Phasenverschiebung[j
                    ]);
229
230         }
231         zeit=zeit+zeitschritt;
232     }
233 }*/
234
235 // calculation of the enveloping function
236
237 inline void einhuellende(Uebergabe *ueber, Kristalle *
        krist, Laserpuls *puls, Winkel *w)
238 {
239     Inten(ueber, krist, puls, w);
240     long double zeitschritt,zeit,zeitspanne,
            zeitschrittplot;
241     double efeld;
242     zeitschritt=puls->wellenlaenge/lichtgeschwindigkeit
            /2/20;
243     zeitspanne=puls->tau*2+krist->dt*(krist->kristalle+1)
            +2*puls->tau*2;
244     zeitschrittplot=zeitspanne/1000000000000/1000;
245     for(int i=0;i<1000;++i)
246     {
247         zeit=(-1)*puls->tau*2/1000000000000+i*
                zeitschrittplot-10*zeitschritt;
248         puls->ploty[i]=0;
```

```
249             for( int j=0;j<23;j++)
250             {
251                 efeld=0;
252                 for(int k=0;k<krist->kopie;++k)
253                 {
254                     efeld=efeld+puls->Intensitaet[k]*Gauss(/*
                            ueber, krist,*/ puls, /*w,*/ zeit
                            *1000000000000, ueber->tk[k])*cosl(
                            puls->kreisfrequenz*zeit+krist->
                            Phasenverschiebung[k]);
255                 }
256                 if(fabs(efeld)>puls->ploty[i])
257                 {
258                     puls->ploty[i]=fabs(efeld);
259                     puls->plotx[i]=zeit*1000000000000;
260                 }
261                 zeit=zeit+zeitschritt;
262             }
263             puls->ploty[i]=pow(puls->ploty[i],2);
264         }
265
266 }
267
268 //calculation of the heigth of the envelopping function
        on the position of the gaussian pulses
269
270 inline void Intensitaet(Uebergabe *ueber, Kristalle *
        krist, Laserpuls *puls, Winkel *w)
271 {
272     Inten(ueber, krist, puls, w);
273     double efeld,hilf;
274     long double zeit,zeitschritt;
275     zeitschritt=puls->wellenlaenge/lichtgeschwindigkeit
            /2/20;
276     for( int i=0;i<=krist->kristalle;++i)
277     {
278         hilf=0;
279         zeit=i*krist->dt/1000000000000-10*zeitschritt;
280         for( int j=0;j<23;j++)
281         {
282             efeld=0;
283             for(int k=0;k<krist->kopie;++k)
284             {
285                 efeld=efeld+puls->Intensitaet[k]*Gauss(/*
                        ueber, krist,*/ puls, /*w,*/ zeit
                        *1000000000000, ueber->tk[k])*cosl(
```

```
                        puls->kreisfrequenz*zeit+krist->
                        Phasenverschiebung[k]);
286                 }
287                 if(fabs(efeld)>hilf)
288                 {
289                     hilf=fabs(efeld);
290                 }
291                 zeit=zeit+zeitschritt;
292             }
293             puls->E[i]=hilf;
294             puls->E[i]=pow(puls->E[i],2);
295         }
296         ueber->counter=ueber->counter+1;
297 }
298
299 // average heigth of the peaks
300
301 inline void dPeakhoehe(/*Uebergabe *ueber,*/ Kristalle *
        krist, Laserpuls *puls/*, Winkel *w*/)
302 {
303     puls->dh=0;
304     for(int i=0;i<=krist->kristalle;++i)
305     {
306         puls->dh=puls->dh+puls->E[i];
307     }
308     puls->dh=puls->dh/(krist->kristalle+1);
309 }
310
311 // number of gaussian peaks on every position
312 /*inline void Permutation(Uebergabe *ueber, Kristalle *
        krist, Laserpuls *puls, Winkel *w)
313 {
314     for(int i=0;i<=krist->kristalle;i++)
315     {
316         puls->Perm[i]=0;
317     }
318     int n=0;
319     for(int j=0;j<=krist->kopie-1;j++)
320     {
321         n=0;
322         for(int i=0;i<=krist->kristalle-1;i++)
323         {
324             n=ueber->A[j][i]+n;
325         }
326         puls->Perm[n]=puls->Perm[n]+1;
327     }
```

```
328 }*/
329
330 //search for biggest peak for Simulation
331
332 inline int Peak(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls/*, Winkel *w*/)
333 {
334     double d=0,prozent=0,abwrozent=0;
335     int pmax=0;
336     ueber->abw=0;
337     dPeakhoehe(/*ueber,*/ krist, puls/*, w*/);
338     for(int i=0;i<=krist->kristalle;++i)
339     {
340         d=puls->E[i]-puls->dh;
341
342         if(puls->E[i]>puls->dh)
343         {
344             prozent=puls->E[i]/puls->dh-1;
345         }
346         if(prozent>abwrozent)
347         {
348             abwrozent=prozent;
349             pmax=i;
350             ueber->abw=d;
351         }
352
353
354         //      if(fabs(ueber->abw)<fabs(d))
355         //       {
356         //  ueber->abw=d;
357         //          pmax=i;
358         //       }
359     }
360     return pmax;
361 }
362
363 // search for conversion
364 /*inline void Signum(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls, Winkel *w)
365 {
366     for(int i=0;i<=krist->kristalle;++i)
367     {
368         puls->signumE[i][0]=puls->signumE[i][1];
369         if(puls->E[i]<puls->dh)
370         {
371             puls->signumE[i][1]=-1;
```

```
372             }
373             if(puls->E[i]>puls->dh)
374             {
375                 puls->signumE[i][1]=1;
376             }
377         }
378 }*/
379
380 //average heigth of gaussian peaks in the area from
        anfang to begin
381 /*inline float Bereichdurchschnitt(Uebergabe *ueber,
        Kristalle *krist, Laserpuls *puls, Winkel *w, int
        anfang,int ende)
382 {
383     float hilf=0;
384     for(int i=anfang;i<=ende;++i)
385     {
386         hilf=puls->E[i]+hilf;
387     }
388     hilf=hilf/(ende-anfang+1);
389     return hilf;
390 }*/
391
392
393 //set points on x axis
394 inline void xachse(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls/*, Winkel *w*/)
395 {
396     ueber->schritt=(krist->dt*(krist->kristalle+1)+2*puls
            ->tau*2)/1000;
397     for(int i=0;i<1000;i++)
398     {
399         if(i==0)
400         {
401             puls->plotx[i]=(-1)*puls->tau*2;
402         }
403         else
404         {
405             puls->plotx[i]=puls->plotx[i-1]+ueber->
                schritt;
406         }
407     }
408 }
409
410
411 //plot einhüllende
```

```
412 /*inline void Graph(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls, Winkel *w)
413 {
414     //Inten(ueber, krist, puls, w);
415     xachse(ueber, krist, puls, w);
416
417     for (int i=0;i<1000;i++)
418     {
419         puls->ploty[i]=0;
420     }
421     for(int i=0;i<1000;i++)
422     {
423         for(int j=0;j<krist->kopie;j++)
424         {
425             puls->ploty[i]=puls->ploty[i]+puls->
                    Intensitaet[j]*Gauss(ueber, krist, puls, w
                    , puls->plotx[i],ueber->tk[j]);
426         }
427     }
428 }*/
429
430 //plot gaussian pulses
431 inline void plotgauss(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls/*, Winkel *w*/, double k)
432 {
433     xachse(ueber, krist, puls/*, w*/);
434     //Intensitaet(ueber,krist,puls,w);
435     double j=0;
436     for(int i=0;i<krist->kopie;++i)
437     {
438         if(floor((10000*k*krist->dt)+0.5)==floor((10000*
                ueber->tk[i])+0.5))
439         {
440             j=j+puls->Intensitaet[i];
441         }
442     }
443     if(j<0)
444     {
445         j=fabs(j);
446     }
447     for(int l=0;l<1000;++l)
448     {
449         puls->ploty[l]=puls->ploty[l]+j*Gauss(/*ueber,
                krist,*/ puls, /*w,*/ puls->plotx[l],(k*krist
                ->dt));
450         puls->ploty[l]=pow(puls->ploty[l],2);
```

```
451     }
452 }
453
454 // search for the highest point in the envelloping
        function or the gaussian pulses
455
456 inline void maxpeak(Uebergabe *ueber, Kristalle *krist,
        Laserpuls *puls, /*Winkel *w,*/ bool manuell)
457 {
458     puls->max=0;
459     if(ueber->plotausrichtung==false)
460     {
461         //int k;
462         if(manuell==true)
463         {
464             for(int i=0;i<=krist->kristalle;i++)
465             {
466                 if(ueber->pulshoehe_anzeigen==false)
467                 {
468                     if(puls->ploty[i]>puls->max)
469                     {
470                         puls->max=puls->ploty[i];
471                     }
472                 }
473                 else
474                 {
475                     if(puls->ploty[i]*100>puls->max)
476                     {
477                         puls->max=puls->ploty[i]*100;
478                     }
479                 }
480             }
481         }
482         else
483         {
484             for(int i=0;i<1000;i++)
485             {
486                 if(ueber->pulshoehe_anzeigen==true)
487                 {
488                     if(puls->ploty[i]*100>puls->max)
489                     {
490                         puls->max=puls->ploty[i]*100;
491                     }
492                 }
493                 else
494                 {
```

```
495                         if(puls->ploty[i]>puls->max)
496                         {
497                             puls->max=puls->ploty[i];
498                         }
499                     }
500                 }
501             }
502         }
503         else
504         {
505             double j;
506             for(int i=0;i<=krist->kristalle;++i)
507             {
508                 j=0;
509                 for(int k=0;k<krist->kopie;k++)
510                 {
511                     if(ueber->tk[k]==i*krist->dt)
512                     {
513                         j=j+puls->Intensitaet[k];
514                     }
515                 }
516                 //j=fabs(j);
517                 j=pow(j,2);
518                 if(ueber->pulshoehe_anzeigen==true)
519                 {
520                     if(puls->max<j*100)
521                     {
522                         puls->max=j*100;
523                     }
524                 }
525                 else
526                 {
527                     if(puls->max<j)
528                     {
529                         puls->max=j;
530                     }
531                 }
532             }
533         }
534         puls->max=puls->max+puls->max*0.2;
535     }
536
537     // changes the intensity like the nonlinear crystals on
        the way to the OSS. to see how the pulse looks in the
        OSS
538     inline double Intensitaetskonverion(double inten, double
```

```
    hoehe_prozent)
539 {
540     double intensitaet;
541     intensitaet=inten;
542 
543     return intensitaet;
544 }
```

A.4 mainwindow.cpp

```
 1 #include "Funktionen.h"
 2 #include "iostream"
 3 #include "unistd.h"
 4 #include "fstream"
 5 #include "mainwindow.h"
 6 #include "ui_mainwindow.h"
 7 #include "cmath"
 8 #include "qcustomplot.h"
 9 #include "ergebnis.h"
10 #include "running.h"
11 #include "oss_datei_laden.h"
12 #include "QStyle"
13 #include "QKeyEvent"
14 #include "QThread"
15 #include "QCoreApplication"
16 #include "Algorithmus.h"
17 #include "erweiterte_einstellungen.h"
18 #include "kristallkonfiguration_laden.h"
19 #include "QApplication"
20 #include "save_plot_data.h"
21 
22 Algorithmus *algo;
23 
24 MainWindow::MainWindow(QWidget *parent) :
25     QMainWindow(parent),
26     ui(new Ui::MainWindow)
27 {
28     ui->setupUi(this);
29     this->setWindowTitle("Pulseshape Simulation (PulSi)");
30     ui->tableWidget->viewport()->installEventFilter(this)
        ;
31     ui->OSS_Kurver_in_Diagramm->setEnabled(false);
32     ui->widget->yAxis->setTickLabels(false);
33     menuBar()->setNativeMenuBar(false);
34 
35 
```

```
36      krist = new Kristalle;
37      w = new Winkel;
38      puls = new Laserpuls;
39      ueber = new Uebergabe;
40      oss =new OSSDatenuebergabe;
41      fenster = new running;
42      einstellungen = new Erweiterte_einstellungen;
43      laden = new Kristallkonfiguration_laden;
44      timer= new zeit;
45      manuel=false;
46      pulseshape_oss=false;
47      abs_hoehe=false;
48
49
50      //Notwendig um in SLOT/SIGNAL verwenden zu können
51      //necessary to use SLOT/SIGNAL
52      qRegisterMetaType<Kristalle>("Kristalle");
53      qRegisterMetaType<Winkel>("Winkel");
54      qRegisterMetaType<Laserpuls>("Laserpuls");
55      qRegisterMetaType<Uebergabe>("Uebergabe");
56
57      //initialize variables
58      for (int i = 0; i < maxkopie; i++)
59      {
60          for (int j = 0; j < maxkrist; j++)
61          {
62              ueber->A[i][j] = 0.0;
63          }
64          ueber->tk[i] = 0.0;
65      }
66      ueber->abw = 0.0;
67      ueber->ende = 0;
68      ueber->einheit = false;
69      ueber->Abbruch = 0;
70      ueber->Genauigkeit = 0.0;
71      ueber->breite = 0.0;
72      ueber->schritt = 0.0;
73      ueber->nr = 0;
74      ueber->row = 0;
75      ueber->col = 0;
76      ueber->initalisiert = false;
77      ueber->endeExtern = 0;
78      ueber->pulshoehe_anzeigen=false;
79      ueber->jones=true;
80      ueber->Brechungsindex=false;
81      ueber->plotausrichtung=false;
```

```
82
83      puls->dh = 0.0;
84      puls->tau = 0.0;
85      puls->max = 0.0;
86      //puls->test = 0;
87      puls->wellenlaenge=0.000000001;
88
89      for (int i = 0; i < 1000; i++)
90      {
91          puls->plotx[i] = 0.0;
92          puls->ploty[i] = 0.0;
93      }
94
95
96      for (int i = 0; i < maxkopie; i++)
97      {
98          puls->Intensitaet[i] = 0.0;
99      }
100
101     for (int i = 0; i < maxkrist + 1; i++)
102     {
103         w->Winkel_Grenzen[i][0]=0;
104         w->Winkel_Grenzen[i][1]=Pi/2;
105         w->Theta[i] = 0.0;
106         puls->E[i] = 0.0;
107         puls->Perm[i] = 0.0;
108         puls->signumE[i][0] = 0;
109         puls->signumE[i][1] = 0;
110         puls->hoehe[i]=1;
111     }
112
113
114     w->dtheta = 0.0;
115
116     krist->kristalle = 0;
117     krist->kopie = 0;
118     krist->dt = 0.0;
119     for( int i=0;i<maxkrist;++i)
120     {
121         krist->eingabePhasenverschiebung[i]=0.0;
122     }
123     krist->Dickekristall=0.0;
124     krist->brechungsindizes[0]=0.0;
125     krist->brechungsindizes[1]=0.0;
126     for(int i=0; i<maxkrist;i++)
127     {
```

```
128             krist->polarisator[i]=false;
129             krist->Phasenverschiebung[i]=0.0;
130             krist->winkel_polarisator[i]=0;
131         }
132 
133 
134         oss->geladen=false;
135         oss->ossmesspunkte=0;
136         oss->ausgangspulshoehe=1;
137         for(int i=0;i<1000;i++)
138         {
139             oss->ossx[i]=0;
140             oss->ossy[i]=0;
141         }
142         oss->mitte=0;
143 
144         for(int i=0;i<=maxkrist;++i)
145         {
146             timer->e[i]=0;
147         }
148 
149         // default settings of the window
150         ui->Brechungsindex_auserordentlich->hide();
151         ui->brechungsindex_ordentlich->hide();
152         ui->wellenlaenge->hide();
153         ui->label->hide();
154         ui->label_3->hide();
155         ui->label_10->setText("Phase shift [Pi]");
156         ui->label_11->hide();
157         ui->tableWidget->resizeColumnsToContents();
158         ui->ausrichtung->setChecked(ueber->plotausrichtung);
159         ui->winkel_manuel->setChecked(manuel);
160         ui->Hoehe_eingangspuls->setEnabled(pulseshape_oss);
161         ui->oss_pulseshape->setEnabled(false);
                // set to false because no funktion of this button
                 yet
162 }
163 
164 MainWindow::~MainWindow()
165 {
166     delete ui;
167 }
168 
169 
170 
171 //make plot
```

```
172 void MainWindow::graph()
173 {
174     QVector<double>x(1000),y(1000);
175     ui->widget->clearGraphs();
176     double durchschnitt=0;                  //average heigth
177     double inten_eingangspuls=0;            //intensity of
            the input pulse
178     //get the heigth of the input pulse
179     if(pulseshape_oss==true)
180     {
181         inten_eingangspuls=ui->Hoehe_eingangspuls->text()
                .toDouble();
182     }
183     else
184     {
185         inten_eingangspuls=1;
186     }
187     //plot whole pulseshape
188     if(manuel==false)
189     {
190         einhuellende(ueber,krist,puls,w);
191
192         x.resize(1000);
193         y.resize(1000);
194             //if activated plot pulseshape after
                    nonlinear elements  !!!!!!!!!!!!not
                    completly implemented!!!!!!!!
195         if(pulseshape_oss==true)
196         {
197             for(int i=0;i<1000;i++)
198             {
199                 puls->ploty[i]=Intensitaetskonverion(
                        inten_eingangspuls,puls->ploty[i]);
200             }
201         }
202         for(int i=0;i<1000;i++)
203         {
204             //heigth in %
205             if(ueber->pulshoehe_anzeigen==true)
206             {
207                 y[i]=puls->ploty[i]/inten_eingangspuls
                        *100;
208             }
209             //absolute heigth
210             else
211             {
```

```
212                 y[i]=puls->ploty[i];
213             }
214             x[i]=puls->plotx[i];
215         }
216         FILE* plot;
217 
218         // write points of enveloping function into file
219         plot=fopen("plot","w");
220         for(int t=0;t<1000;t++)
221         {
222             fprintf(plot,"%f",puls->ploty[t]);
223             fprintf(plot,";");
224             fprintf(plot,"%f",puls->plotx[t]);
225             fprintf(plot,"\n");
226         }
227         fclose(plot);
228 
229         //measure rms, dh,usw
230 
231         double begin,ende,rms=0,dh=0,maxabw=0,maxpeak=0,
                fwhm_anfang=0,fwhm_ende=0,fwhm_breite;
232         int lauf=0;
233         bool fwhm_begin=false;
234         begin=ui->anfang_eigenschaft->text().toDouble();
235         ende=ui->Ende_Eigenschaft->text().toDouble();
236         for(int i=0;i<1000;++i)
237         {
238             if((puls->plotx[i]>=begin)&&(puls->plotx[i]<=
                    ende))
239             {
240                 dh=dh+puls->ploty[i];
241                 lauf=lauf+1;
242             }
243             if(puls->ploty[i]>maxpeak)
244             {
245                 maxpeak=puls->ploty[i];
246             }
247         }
248         dh=dh/lauf;
249         for(int i=0;i<1000;++i)
250         {
251             if((puls->plotx[i]>=begin)&&(puls->plotx[i]<=
                    ende))
252             {
253                 rms=rms+pow((puls->ploty[i]-dh),2);
254                 if(fabs(puls->ploty[i]-dh)>maxabw)
```

```
255                     {
256                         maxabw=fabs(puls->ploty[i]-dh);
257                     }
258                 }
259                 if(puls->ploty[i]>maxpeak/2)
260                 {
261                     if(fwhm_begin==false)
262                     {
263                         fwhm_anfang=puls->plotx[i];
264                         fwhm_begin=true;
265                     }
266                     if(fwhm_begin==true)
267                     {
268                         fwhm_ende=puls->plotx[i];
269                     }
270                 }
271             }
272             rms=sqrt(rms/lauf);
273             fwhm_breite=fwhm_ende-fwhm_anfang;
274             if(ueber->pulshoehe_anzeigen==true)
275             {
276                 dh=dh*100/inten_eingangspuls;
277                 rms=rms*100/inten_eingangspuls;
278                 maxabw=maxabw*100/inten_eingangspuls;
279             }
280             ui->Hoehe->setText("Heigth:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"+
                    QString("%1").arg(dh,0,'f',4));
281             ui->RMS->setText("RMS:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"+
                    QString("%1").arg(rms,0,'f',4));
282             ui->groeste_Abw->setText(("max.␣deviation:␣␣"+
                    QString("%1").arg(maxabw,0,'f',4)));
283             ui->FWHM->setText("FWHM:␣"+QString("%1").arg(
                    fwhm_breite,0,'f',2));
284         }
285         // plot if manual input is chosen
286         else
287         {
288             Intensitaet(ueber,krist,puls,w);
289             x.resize(krist->kristalle+1);
290             y.resize(krist->kristalle+1);
291             for(int i=0;i<=krist->kristalle;++i)
292             {
293                 puls->ploty[i]=puls->E[i];
294             }
295             if(pulseshape_oss==true)
296             {
```

```
297                 //if activated plot pulseshape after
                        nonlinear elements   !!!!!!!!!!!!!not
                        completly implemented!!!!!!!!
298                 for(int i=0;i<1000;i++)
299                 {
300                     puls->ploty[i]=Intensitaetskonverion(
                            inten_eingangspuls,puls->ploty[i]);
301                 }
302             }
303             for( int i=0; i<=krist->kristalle;++i)
304             {
305                 x[i]=i*krist->dt;
306                 //heigth in %
307                 if(ueber->pulshoehe_anzeigen==true)
308                 {
309                     y[i]=puls->ploty[i]*100/
                            inten_eingangspuls;
310                     durchschnitt=durchschnitt+puls->ploty[i
                            ]*100/inten_eingangspuls;
311                 }
312                 //absolute heigth
313                 else
314                 {
315                     y[i]=puls->ploty[i];
316                     durchschnitt=durchschnitt+puls->ploty[i];
317                 }
318             }
319             durchschnitt=durchschnitt/(krist->kristalle+1);
320         }
321
322         ui->widget->addGraph();
323         // plot settings if manual settings are chosen
324         if(manuel==true)
325         {
326             ui->widget->graph(0)->setScatterStyle(
                    QCPScatterStyle(QCPScatterStyle::ssCross,10));
327             ui->widget->graph(0)->setBrush(QBrush(Qt::NoBrush
                    ));
328             ui->widget->graph(0)->setLineStyle(QCPGraph::
                    lsNone);
329         }
330         ui->widget->graph(0)->setName("Simulation:␣pulse");
331         ui->widget->graph(0)->setData(x,y);
332         // plot range
333         if(manuel==false)
334         {
```

```
335             ui->widget->xAxis->setRange((-1)*puls->tau*2,
                    krist->dt*(krist->kristalle+1)+2*puls->tau*2);
336             maxpeak(ueber, krist, puls, /*w,*/manuel);
337             ui->widget->yAxis->setRange(0,puls->max);
338         }
339         else
340         {
341             ui->widget->xAxis->setRange((-1)*puls->tau*2,
                    krist->dt*(krist->kristalle+1)+2*puls->tau*2);
342             maxpeak(ueber, krist, puls,/* w,*/manuel);
343             ui->widget->yAxis->setRange(0,puls->max);
344         }
345         x.resize(1000);
346         y.resize(1000);
347         double dj=0;
348
349         // plot gaussian peaks
350         for(int j=0;j<=krist->kristalle;j++)
351         {
352             for(int i=0;i<1000;i++)
353             {
354                 puls->ploty[i]=0;
355             }
356             plotgauss(ueber, krist, puls/*, w*/, dj);
357
358         //if activated plot pulseshape after nonlinear
                elements  !!!!!!!!!!!!!not completly implemented
                !!!!!!!!
359             if(pulseshape_oss==true)
360             {
361                 for(int i=0;i<1000;i++)
362                 {
363                     puls->ploty[i]=Intensitaetskonverion(
                            inten_eingangspuls,puls->ploty[i]);
364                 }
365             }
366             for (int i=0;i<1000;i++)
367             {
368                 //heigth in %
369                 if(ueber->pulshoehe_anzeigen==true)
370                 {
371                     y[i]=puls->ploty[i]*100/
                            inten_eingangspuls;
372                 }
373                 //absolute heigth
374                 else
```

```
375                 {
376                     y[i]=puls->ploty[i];
377                 }
378                 x[i]=puls->plotx[i];
379             }
380             ui->widget->addGraph();
381             if(j>0)
382             {
383                 ui->widget->graph(j)->removeFromLegend();
384             }
385             ui->widget->graph(j+1)->setName("Simulation: sum
                    of copies");
386             ui->widget->graph(j+1)->setPen(QColor
                    (255,204,204,255));
387             ui->widget->graph(j+1)->setData(x,y);
388             dj=dj+1;
389         }
390
391         // plot oss data
392         if(oss->geladen==true)
393         {
394             if(ueber->ossdatenplotten==true)
395             {
396                 QString a;
397                 double mitte_alt=0,mitte=0;
398                 mitte_alt=oss->mitte;
399                 a=ui->mitte_oss->text();
400                 mitte=(-1)*a.toDouble();
401
402                 // shift OSS plot
403                 double hilf=0,hilf1=0,hilf3=0;
404                 bool hilf2=false;
405                 for(int i=0;i<=oss->ossmesspunkte;i++)
406                 {
407                     if(hilf<oss->ossy[i])
408                     {
409                         hilf=oss->ossy[i];
410                     }
411                 }
412                 for(int i=0;i<=oss->ossmesspunkte;i++)
413                 {
414                     if((oss->ossy[i]>hilf/2)&&(hilf2==false))
415                     {
416                         hilf1=oss->ossx[i];
417                         hilf2=true;
418                     }
```

```
419                 if((oss->ossy[i]>hilf/2)&&(hilf2==true))
420                 {
421                     hilf3=oss->ossx[i];
422                 }
423             }
424             if(krist->kristalle==0)
425             {
426                 for(int i=0;i<=oss->ossmesspunkte;i++)
427                 {
428                     oss->ossx[i]=oss->ossx[i]-hilf1;
429                 }
430             }
431             // shift oss data horizontally
432             if(krist->kristalle>0)
433             {
434                 for(int i=0;i<=oss->ossmesspunkte;i++)
435                 {
436                     oss->ossx[i]=oss->ossx[i]-(mitte-
                        mitte_alt);
437                 }
438             }
439             oss->mitte=oss->mitte+mitte-mitte_alt;
440
441
442             for(int i=0;i<1000;i++)
443             {
444                 if(i<oss->ossmesspunkte)
445                 {
446                     x[i]=oss->ossx[i];
447                     if(ueber->pulshoehe_anzeigen==true)
448                     {
449                         y[i]=oss->ossy[i]*100;
450                     }
451                     else
452                     {
453                         y[i]=oss->ossy[i];
454                     }
455                 }
456                 if(i>=oss->ossmesspunkte)
457                 {
458                     x[i]=x[i-1];
459                     y[i]=y[i-1];
460                 }
461             }
462         }
463         ui->widget->addGraph();
```

```
464             ui->widget->graph((krist->kristalle)+2)->setName(
                    "Measurement OSS");
465             ui->widget->graph(krist->kristalle+2)->
                    addToLegend();
466             ui->widget->graph((krist->kristalle)+2)->setPen(
                    QColor(255,0,000,255));
467             ui->widget->graph((krist->kristalle)+2)->setData(
                    x,y);
468             if(ueber->ossdatenplotten==false)
469             {
470                 ui->widget->graph(krist->kristalle+2)->
                        removeFromLegend();
471             }
472         }
473         // set labels and data if manuall input is chosen
474         if(manuel==true)
475         {
476             QVector<double> a(2),b(2);
477             int nummer;
478             a[0]=puls->plotx[0];
479             a[1]=puls->plotx[999];
480             b[0]=durchschnitt;
481             b[1]=durchschnitt;
482             ui->widget->addGraph();
483             if(ueber->ossdatenplotten==true)
484             {
485                 nummer=krist->kristalle+3;
486             }
487             else
488             {
489                 nummer=krist->kristalle+2;
490             }
491             ui->widget->graph(nummer)->setName("Average
                    heigth");
492             ui->widget->graph(nummer)->addToLegend();
493             ui->widget->graph(nummer)->setPen(QColor
                    (0,100,0,255));
494             ui->widget->graph(nummer)->setData(a,b);
495         }
496         ui->widget->legend->setBrush(Qt::transparent);
497         ui->widget->legend->setVisible(true);
498         ui->widget->xAxis->setLabel("Pulse Length[ps]");
499 
500         if(ueber->pulshoehe_anzeigen==true||abs_hoehe==true)
501         {
502             if(ueber->pulshoehe_anzeigen==true)
```

```
503             {
504                 ui->widget->yAxis->setLabel("Intensity[%]");
505             }
506             else
507             {
508                 ui->widget->yAxis->setLabel("Intensity");
509             }
510             ui->widget->yAxis->setTickLabels(true);
511         }
512         else
513         {
514             ui->widget->yAxis->setLabel("Intensity");
515             ui->widget->yAxis->setTickLabels(false);
516         }
517         ui->widget->replot();
518 
519 }
520 
521 //build table
522 
523 void MainWindow::tabelle()
524 {
525     ui->tableWidget->setRowCount(krist->kristalle);
526     for (int i = 0; i <=2; i++)
527     {
528         for (int j = 0; j <= krist->kristalle-1; j++)
529         {
530             //fill 1. column with angle[]
531             if(i==0)
532             {
533                 ui->tableWidget->setVerticalHeaderItem(j,
                        new QTableWidgetItem("Crystal No."+
                        QString::number(j+1)));
534                 double whilf=(180/Pi*w->Theta[j+1]);
535                 ui->tableWidget->setItem( j, i,new
                        QTableWidgetItem(QString("%1").arg(
                        whilf,0,'f',6) ));
536             }
537             // fill 2. column with angle [rad]
538             if(i==1)
539             {
540                 ui->tableWidget->setItem( j, i,new
                        QTableWidgetItem(QString("%1").arg(w->
                        Theta[j+1],0,'f',8) ));
541             }
542             // fill 3. column with phase shift
```

```
543                 if(i==2)
544                 {
545                     ui->tableWidget->setItem(j,i,new
                            QTableWidgetItem(QString("%1").arg(
                            krist->eingabePhasenverschiebung[j]/Pi
                            ,0,'f',8)));
546                 }
547                 //mark background red if angle out of borders
548                 if((w->Theta[j+1]<w->Winkel_Grenzen[j+1][0])
                        ||(w->Theta[j+1]>w->Winkel_Grenzen[j
                        +1][1]))
549                 {
550                     ui->tableWidget->item(j,i)->setBackground
                            (Qt::red);
551                 }
552             }
553         }
554         ui->tableWidget->setColumnWidth(0, 80);
555     }
556
557     // set text under start simulation button after the
            simulation endet
558     void MainWindow::simbeendet()
559     {
560         ui->label_7->setText("␣");
561         ueber->ende=1;
562     }
563
564     // input of the beamshaper settings
565     void MainWindow::on_pushButton_2_clicked()
566     {
567         QString a;
568         ui->kristallnr->clear();
569
570         ui->label_7->setText("");
571
572         //set number of crystals
573         a=ui->lineEdit_2->text();
574         krist->kristalle=a.toInt();
575
576         //width of input pulse
577         a=ui->lineEdit_4->text();
578         puls->tau=a.toDouble();
579
580         //delay of ordinary and extraordinary wave
581         a=ui->lineEdit_5->text();
```

```
582         krist->dt=a.toDouble();
583
584         //accuracy
585         a=ui->lineEdit_6->text();
586         ueber->Genauigkeit=a.toDouble()/100;
587
588         if(ueber->Brechungsindex==true)
589         {
590             //refraction index
591             a=ui->Brechungsindex_auserordentlich->text();
592             krist->brechungsindizes[1]=a.toDouble();
593             a=ui->brechungsindex_ordentlich->text();
594             krist->brechungsindizes[0]=a.toDouble();
595
596             //thickness of the crystall
597             a=ui->dikce_kristall->text();
598             krist->Dickekristall=a.toDouble()/100;
599
600             //waveleght
601             a=ui->wellenlaenge->text();
602             puls->wellenlaenge=a.toDouble()/1000000000;
603         }
604         else
605         {
606             //phaseshift
607             a=ui->dikce_kristall->text();
608             for(int i=0;i<krist->kristalle;++i)
609             {
610                 krist->eingabePhasenverschiebung[i]=a.
                        toDouble()*Pi;
611             }
612         }
613
614         ueber->dateiladen=false;
615         if(krist->kristalle<21)
616         {
617             Definition(ueber, krist, puls, w);
618
619
620             tabelle();
621             graph();
622         }
623         for(int i=0;i<krist->kristalle;i++)
624         {
625             ui->kristallnr->addItem(QString("%1").arg(i+1));
626         }
```

```
627         for(int i=0;i<=krist->kristalle;++i)
628         {
629             puls->hoehe[i]=1;
630         }
631     }
632
633     // start simulation
634     void MainWindow::on_sim_start_clicked()
635     {
636         ueber->Abbruch=0;
637         ueber->ende=0;
638         ueber->endeExtern=0;
639
640         QPalette color;
641         color.setColor(ui->label_7->backgroundRole(),Qt::red)
                ;
642         color.setColor(ui->label_7->foregroundRole(),Qt::red)
                ;
643         ui->label_7->setPalette(color);
644         ui->label_7->setText("");
645
646         // output of errors
647
648         if(puls->wellenlaenge<=0)
649         {
650             ui->label_7->setText("Wavelength to small");
651             ueber->Abbruch=1;
652         }
653         if(ueber->Genauigkeit<0.0000000001)
654         {
655             ui->label_7->setText("Accuracy to small");
656             ueber->Abbruch=1;
657         }
658         if(ueber->Genauigkeit==0)
659         {
660             ui->label_7->setText("Error: Accuracy=0");
661             ueber->Abbruch=1;
662         }
663         if(krist->kristalle==0)
664         {
665             ui->label_7->setText("Error: no crystals");
666             ueber->Abbruch=1;
667         }
668         if(ueber->Abbruch==0)
669         {
670
```

```
671         //block start button
672         ui->sim_start->setEnabled(false);
673
674         // start thread for simulation and connecting
                stuff
675         QThread *seqThread = new QThread;
676
677         algo = new Algorithmus(ueber, krist, puls, w,
                timer);
678
679
680         QObject::connect(algo, SIGNAL(
                AlgorithmusDatenUpdate(Laserpuls*, Winkel*)),
                this, SLOT(update(Laserpuls*, Winkel*)));
681
682         algo->moveToThread(seqThread);
683
684         connect(seqThread, SIGNAL(started()), algo, SLOT(
                ausfuehren()));
685         connect(algo, SIGNAL(beendet()), seqThread, SLOT(
                quit()));
686
687         connect(algo, SIGNAL(beendet()),algo, SLOT(
                deleteLater()));
688         connect(seqThread, SIGNAL(finished()), seqThread,
                 SLOT(deleteLater()));
689         connect(algo, SIGNAL(beendet()), this, SLOT(
                AlgorithmusBeendet()));
690
691         connect(seqThread, SIGNAL(started()), fenster,
                SLOT(start()));
692         connect(seqThread,SIGNAL(finished()),fenster,
                SLOT(fensterschliessen()));
693         connect(this,SIGNAL(anzahlkristalle(int)),fenster
                ,SLOT(anzahlkristalle(int)));
694         connect(algo,SIGNAL(updategraf(zeit*)),fenster,
                SLOT(graph(zeit*)));
695
696         connect(fenster,SIGNAL(quit()),this,SLOT(
                simbeendet()));
697         emit anzahlkristalle(krist->kristalle);
698
699         seqThread->start();
700         ui->label_7->setText("Simulation running");
701         fenster->verbinde(algo->ueber, algo->timer);
702         fenster->show();
```

```
703
704         }
705
706 }
707
708 // save crystall settings
709
710 void MainWindow::on_actionErgenisse_spiechern_triggered()
711 {
712     ueber->breite=puls->tau;
713     //open new window for saving data
714     Ergebnis *save= new Ergebnis;
715     connect(this,SIGNAL(Daten(Uebergabe*,Kristalle*,
            Laserpuls*,Winkel*)),save,SLOT(Daten_ergebnis(
            Uebergabe*,Kristalle*,Laserpuls*,Winkel*)));
716     emit Daten(ueber,krist,puls,w);
717     save->show();
718 }
719
720 // open window to save plot
721 void MainWindow::on_actionDiagramm_speichern_triggered()
722 {
723     save_plot_data *speichern = new save_plot_data;
724     connect(speichern,SIGNAL(dateiname(QString)),this,
            SLOT(plot_speichern(QString)));
725     speichern->show();
726     //ui->widget->savePng("Plot.png");
727 }
728
729 // save plot
730 void MainWindow::plot_speichern(QString name)
731 {
732     ui->widget->savePng(name);
733 }
734
735
736 //angle steps in degree/ radian measure
737 void MainWindow::on_checkBox_clicked()
738 {
739     if(ueber->einheit==true)
740     {
741         ueber->einheit=false;
742     }
743     else
744     {
745         ueber->einheit=true;
```

```
746         }
747
748     }
749
750     // number of crystall for changing crystalls by pressing
        +/- button
751     void MainWindow::on_kristallnr_currentIndexChanged(int
        index)
752     {
753         ueber->nr=index+1;
754     }
755
756     // changing crystall angles(+)
757     void MainWindow::on_plus_clicked()
758     {
759         double schritt;
760         QString a;
761         a=ui->winkelschritteingabe->text();
762         //angle in [rad]
763         if(ueber->einheit==true)
764         {
765             schritt=a.toDouble();
766         }
767         //angle in []
768         else
769         {
770             schritt=a.toDouble()*Pi/180;
771         }
772
773         w->Theta[ueber->nr]=w->Theta[ueber->nr]+schritt;
774         tabelle();
775         graph();
776     }
777
778     // changing crystall angles(-)
779     void MainWindow::on_minus_clicked()
780     {
781         double schritt;
782         QString a;
783         a=ui->winkelschritteingabe->text();
784         //angle in [rad]
785         if(ueber->einheit==true)
786         {
787             schritt=a.toDouble();
788         }
789         //angle in []
```

```
790         else
791         {
792             schritt=a.toDouble()*Pi/180;
793         }
794
795         w->Theta[ueber->nr]=w->Theta[ueber->nr]-schritt;
796         tabelle();
797         graph();
798     }
799
800     //find changed cell in table
801     void MainWindow::on_tableWidget_cellChanged(int row, int
        column)
802     {
803         ueber->row=row;
804         ueber->col=column;
805     }
806
807
808     //accept changed values in table by pressing enter
809
810
811     void MainWindow::keyReleaseEvent(QKeyEvent *event)
812     {
813         if(event->key()==Qt::Key_Return)
814         {
815             QTableWidgetItem* a;
816             a=ui->tableWidget->item(ueber->row,ueber->col);
817             //read out of changed cell in the table if
                    beamshaper is defined
818             if(ueber->initalisiert==true)
819             {
820                 if(ueber->col==0)
821                 {
822                     w->Theta[ueber->row+1]=(a->text().
                            toDouble())*Pi/180;
823                 }
824                 if(ueber->col==1)
825                 {
826                     w->Theta[ueber->row+1]=a->text().toDouble
                            ();
827                 }
828                 if(ueber->col==2)
829                 {
830                     krist->eingabePhasenverschiebung[ueber->
                            row]=a->text().toDouble()*Pi;
```

```
831                 }
832             }
833 //recalculate phase shift
834             for(int j=0;j<krist->kopie;++j)
835             {
836                 krist->Phasenverschiebung[j]=0;
837                 for(int i=0;i<=krist->kristalle-1;++i)
838                 {
839                     krist->Phasenverschiebung[j]=krist->
                            Phasenverschiebung[j]+krist->
                            eingabePhasenverschiebung[i]*ueber->A[
                            j][i];
840                 }
841             }
842             tabelle();
843             graph();
844         }
845         //restore old values if ESC is pressed
846         if(event->key()==Qt::Key_Escape)
847         {
848             tabelle();
849         }
850 }
851
852 // enable start button for simulation and set some
        variables to 0 after simulation endet
853 void MainWindow::AlgorithmusBeendet()
854 {
855     ui->sim_start->setEnabled(true);
856     fenster->ueber = NULL;
857     //algo->ueber = NULL;
858     algo = NULL;
859 }
860
861 //send data to mainwindow
862 void MainWindow::update(Laserpuls *pulsP, Winkel *wP)
863 {
864     *puls = *pulsP;
865     *w = *wP;
866
867
868     graph();
869     tabelle();
870 }
871
872 //get OSS data
```

```
873 void MainWindow::ossdaten(OSSDatenuebergabe *Daten)
874 {
875     *oss=*Daten;
876     std::cout<<"number of points OSS:"<<oss->
          ossmesspunkte<<"\n";
877     ui->OSS_Kurver_in_Diagramm->setEnabled(oss->geladen);
878 }
879
880 //load OSS file
881 void MainWindow::on_actionOSS_Datei_laden_triggered()
882 {
883     OSS_Datei_laden *ossfileladen=new OSS_Datei_laden;
884     connect(this,SIGNAL(Daten(Uebergabe*,Kristalle*,
          Laserpuls*,Winkel*)),ossfileladen,SLOT(
          Datenuebergabe_OSS(Uebergabe*,Kristalle*,Laserpuls
          *,Winkel*)));
885     emit Daten(ueber,krist,puls,w);
886     connect(ossfileladen,SIGNAL(OSSDaten(
          OSSDatenuebergabe*)),this,SLOT(ossdaten(
          OSSDatenuebergabe*)));
887     connect(ossfileladen,SIGNAL(geladen()),this,SLOT(
          graph()));
888     ossfileladen->show();
889 }
890
891 // set varaible true/ false to show/ dont show oss plot
892 void MainWindow::on_OSS_Kurver_in_Diagramm_clicked()
893 {
894     if(ueber->ossdatenplotten==true)
895     {
896         ueber->ossdatenplotten=false;
897     }
898     else
899     {
900         ueber->ossdatenplotten=true;
901     }
902     graph();
903 }
904
905 // set variable to true/ false to show / or dont show
      heigth in percent
906 void MainWindow::on_checkBox_2_clicked()
907 {
908     if(ueber->pulshoehe_anzeigen==true)
909     {
910         ueber->pulshoehe_anzeigen=false;
```

```
911         }
912         else
913         {
914             ueber->pulshoehe_anzeigen=true;
915             if(abs_hoehe==true)
916             {
917                 abs_hoehe=false;
918                 ui->absolute_hoehe->setChecked(abs_hoehe);
919             }
920         }
921         graph();
922     }
923
924     // open window for advanced settings
925     void MainWindow::
            on_actionErweiterte_Einstellungen_triggered()
926     {
927         connect(this,SIGNAL(Daten(Uebergabe*,Kristalle*,
                Laserpuls*,Winkel*)),einstellungen,SLOT(
                Daten_Einstellung(Uebergabe*,Kristalle*,Laserpuls
                *,Winkel*)));
928         connect(einstellungen,SIGNAL(daten(Uebergabe*,
                Kristalle*,Laserpuls*,Winkel*)),this,SLOT(Laden(
                Uebergabe*,Kristalle*,Laserpuls*,Winkel*)));
929         connect(einstellungen,SIGNAL(brechungsindex()),this,
                SLOT(Interface()));
930         emit Daten(ueber,krist,puls,w);
931         einstellungen->show();
932     }
933
934     //build Interface by settings
935     void MainWindow::Interface()
936     {
937         if(ueber->Brechungsindex==false)
938         {
939             ui->Brechungsindex_auserordentlich->hide();
940             ui->brechungsindex_ordentlich->hide();
941             ui->wellenlaenge->hide();
942             ui->label->hide();
943             ui->label_3->hide();
944             ui->label_10->setText("Phase shift [Pi]");
945             ui->label_11->hide();
946             puls->wellenlaenge=0.000000001;
947         }
948         else
949         {
```

```
950             ui->Brechungsindex_auserordentlich->show();
951             ui->brechungsindex_ordentlich->show();
952             ui->wellenlaenge->show();
953             ui->label->show();
954             ui->label_3->show();
955             ui->label_10->setText("Thickness of the crystals
                    [cm]");
956             ui->label_11->show();
957             puls->wellenlaenge=0;
958         }
959 }
960
961
962 //load crystal settings
963 void MainWindow::
        on_actionKristallkonfiguration_laden_triggered()
964 {
965     connect(this,SIGNAL(Daten(Uebergabe*,Kristalle*,
            Laserpuls*,Winkel*)),laden,SLOT(daten(Uebergabe*,
            Kristalle*,Laserpuls*,Winkel*)));
966     connect(laden,SIGNAL(Daten(Uebergabe*,Kristalle*,
            Laserpuls*,Winkel*)),this,SLOT(Laden(Uebergabe*,
            Kristalle*,Laserpuls*,Winkel*)));
967     emit Daten(ueber,krist,puls,w);
968     laden->show();
969 }
970
971 // build window if crystall settings are loaded
972 void MainWindow::Laden(Uebergabe *ueberP, Kristalle *
        kristP, Laserpuls *pulsP, Winkel *wp)
973 {
974     *ueber=*ueberP;
975     *krist=*kristP;
976     *puls=*pulsP;
977     *w=*wp;
978     ueber->dateiladen=true;
979     ui->lineEdit_2->setText(QString::number(krist->
            kristalle));
980     ui->lineEdit_4->setText(QString("%1").arg(puls->tau
            ,0,'f',8));
981     ui->lineEdit_5->setText(QString("%1").arg(krist->dt
            ,0,'f',8));
982     ui->lineEdit_6->setText(QString("%1").arg(ueber->
            Genauigkeit*100,0,'f',8));
983     ui->kristallnr->clear();
984     for(int i=0;i<krist->kristalle;i++)
```

```
985         {
986             ui->kristallnr->addItem(QString("%1").arg(i+1));
987         }
988         Definition(ueber,krist,puls,w);
989         graph();
990         tabelle();
991     }
992
993     //adjust graph to gaussian pulses/ enveloping function
994     void MainWindow::on_ausrichtung_clicked()
995     {
996         if(ueber->plotausrichtung==true)
997         {
998             ueber->plotausrichtung=false;
999         }
1000        else
1001        {
1002            ueber->plotausrichtung=true;
1003        }
1004        graph();
1005    }
1006
1007    //faster painting of the graph for changing angles by
        hand
1008    void MainWindow::on_winkel_manuel_clicked()
1009    {
1010        if(manuel==false)
1011        {
1012            manuel=true;
1013        }
1014        else
1015        {
1016            manuel=false;
1017        }
1018        graph();
1019    }
1020
1021    // close every window of the program if mainwindow is
        closed
1022    void MainWindow::closeEvent(QCloseEvent *)
1023    {
1024        qApp->quit();
1025    }
1026
1027    //plot the pulseshape how it looks in the oss
1028    void MainWindow::on_oss_pulseshape_clicked()
```

```
1029 {
1030     if(pulseshape_oss==false)
1031     {
1032         pulseshape_oss=true;
1033         ui->Hoehe_eingangspuls->setEnabled(pulseshape_oss
                 );
1034     }
1035     else
1036     {
1037         pulseshape_oss=false;
1038         ui->Hoehe_eingangspuls->setEnabled(pulseshape_oss
                 );
1039     }
1040     graph();
1041 }
1042
1043 // sets output of the the absolute values in the plot
        true or false
1044 void MainWindow::on_absolute_hoehe_clicked()
1045 {
1046     if(abs_hoehe==false)
1047     {
1048         abs_hoehe=true;
1049         //check if data output in percent is activated
1050         if(ueber->pulshoehe_anzeigen==true)
1051         {
1052             ueber->pulshoehe_anzeigen=false;
1053             ui->checkBox_2->setChecked(ueber->
                     pulshoehe_anzeigen);
1054         }
1055     }
1056     else
1057     {
1058         abs_hoehe=false;
1059     }
1060     graph();
1061 }
1062
1063 // button change accuracy
1064 void MainWindow::on_genauigkeit_aendern_clicked()
1065 {
1066     QString a;
1067     a=ui->lineEdit_6->text();
1068     ueber->Genauigkeit=a.toDouble()/100;
1069 }
```

A.5 mainwindow.h

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include "QTableWidget"
6  #include "running.h"
7  #include "erweiterte_einstellungen.h"
8  #include "kristallkonfiguration_laden.h"
9  #include "global.h"
10 #include "QString"
11
12
13 namespace Ui {
14 class MainWindow;
15 }
16
17 class MainWindow : public QMainWindow
18 {
19     Q_OBJECT
20
21 public:
22     explicit MainWindow(QWidget *parent = 0);
23     ~MainWindow();
24
25 public slots:
26
27     void on_pushButton_2_clicked();
28
29     void on_sim_start_clicked();
30
31     void on_actionErgenisse_spiechern_triggered();
32
33     void on_actionDiagramm_speichern_triggered();
34
35     void on_checkBox_clicked();
36
37     void graph();
38
39     void tabelle();
40
41     void on_kristallnr_currentIndexChanged(int index);
42
43     void on_plus_clicked();
44
```

```
45      void on_minus_clicked();
46
47      void on_tableWidget_cellChanged(int row, int column);
48
49      void simbeendet();
50
51      void AlgorithmusBeendet();
52
53      void update(Laserpuls *pulsP, Winkel *wP);
54
55      void ossdaten(OSSDatenuebergabe *Daten);
56
57      void Interface();
58
59      void Laden(Uebergabe *ueberP, Kristalle *kristP,
            Laserpuls *pulsP, Winkel *wp);
60
61      void plot_speichern(QString);
62
63  signals:
64
65      void enter(QKeyEvent*);
                // pressed key
66
67      void stoppThread(int);
68
69      void Daten(Uebergabe *ueberP, Kristalle *kristP,
            Laserpuls *pulsP, Winkel *wP);      // send values
70
71      void anzahlkristalle(int k);
                //send number of crystals
72
73      void datei_existiert(int datei);
                //signal thats file with given name does already
                exist
74
75
76  public:
77      Ui::MainWindow *ui;
78
79      running *fenster;
80      Erweiterte_einstellungen *einstellungen;
81      Kristallkonfiguration_laden *laden;
```

```
82
83      Kristalle *krist;
84      Winkel *w;
85      Laserpuls *puls;
86      Uebergabe *ueber;
87      OSSDatenuebergabe *oss;
88      zeit *timer;
89      bool manuel;              // change angles by hand
90      bool pulseshape_oss;      // pulse shape in the oss
91      bool abs_hoehe;           // absolute pulse heigth
            should be shown or not
92
93
94  private slots:
95
96      void on_actionOSS_Datei_laden_triggered();
97
98      void on_OSS_Kurver_in_Diagramm_clicked();
99
100     void on_checkBox_2_clicked();
101
102     void on_actionErweiterte_Einstellungen_triggered();
103
104     void on_actionKristallkonfiguration_laden_triggered()
            ;
105
106     void on_ausrichtung_clicked();
107
108     void keyReleaseEvent(QKeyEvent *);
109
110     void on_winkel_manuel_clicked();
111
112     void closeEvent(QCloseEvent *);
113
114     void on_oss_pulseshape_clicked();
115
116     void on_absolute_hoehe_clicked();
117     void on_genauigkeit_aendern_clicked();
118 };
119
120
121
122 #endif // MAINWINDOW_H
```

A.6 Algorithmus.cpp

```
1  #include "Algorithmus.h"
2  #include "QtWidgets"
3  #include "iostream"
4  #include "Funktionen.h"
5  #include "unistd.h"
6  #include "sys/time.h"
7  #include "time.h"
8
9
10
11 Algorithmus::Algorithmus(Uebergabe *ueberP, Kristalle *
       kristP, Laserpuls *pulsP, Winkel *wP, zeit *z)
12 {
13
14      krist = new Kristalle;
15      w = new Winkel;
16      puls = new Laserpuls;
17      ueber = new Uebergabe;
18      timer =new zeit;
19
20
21
22      *krist = *kristP;
23      *w = *wP;
24      *puls = *pulsP;
25      *ueber = *ueberP;
26      *timer = *z;
27
28 }
29
30
31 Algorithmus::~Algorithmus()
32 {
33
34 }
35
36
37 //run algorithm
38 void Algorithmus::ausfuehren()
39 {
40      timeval start, ende;
                                                          // time
           the computition needed
41      clock_t clock_start, clock_end;
                                                  // CPU time
          computition needed
```

```
42      clock_t clock();
43      long double Zeit=0;
                                                        //
        time needed
44
45      double dwa=0;

        // direction crystal angle is changed
46      double genau=0.5;
                                                          //
        accuracy internal
47      double hilf=0,hilf2=0,hilf4;
                                                  // auxilary
        variables for several uses
48      double alt_E[maxkrist+1];
                                                    // to reset
        values of E after some loops (saves computition
        time)
49      double alt_dh;

        // to reset values of dh after some loops (saves
        computition time)
50      int pmax=0;

        //Peak which is changed
51      int n=0;

        // number of crystall which is changed
52      int iteration=0;
                                                          //
        number of iterations
53      ueber->counter=0;
54
55      //initialize timemeasurement
56      gettimeofday(&start,NULL);
57      clock_start=clock();
58
59      pmax=krist->kristalle-1;
60      Intensitaet(ueber, krist, puls, w);
61      dPeakhoehe(/*ueber,*/ krist, puls/*, w*/);
62
63      // send current intensities to realtimeplot of the
        status of the simulation
64      timer->maxpeak=0;
65      timer->minpeak=0;
66      for(int i=0;i<=krist->kristalle;++i)
```

```
67      {
68          timer->e[i]=puls->E[i];
69          if(timer->e[timer->maxpeak]<puls->E[i])
70          {
71              timer->maxpeak=i;
72          }
73          if(timer->e[timer->minpeak]>puls->E[i])
74          {
75              timer->minpeak=i;
76          }
77      }
78      emit updategraf(timer);
79
80      while((ueber->ende==0) && (ueber->endeExtern==0))
81      {
82          iteration=iteration+1;
83          Inten(ueber, krist, puls, w);
84          Intensitaet(ueber, krist, puls, w);
85          Peak(ueber, krist, puls/*, w*/);
86          dPeakhoehe(/*ueber,*/ krist, puls/*, w*/);
87          //dtheta=dthetamin;
88          w->dtheta=0.0174532925199;
89          //w->dtheta=0.0000017453;
90          //if(genau<=ueber->Genauigkeit)
91          //{
92          genau=0.5;
93          //}
94
95          if(ueber->ende==2)
96          {
97              break;
98          }
99
100         // changing peak which is changed
101         while((fabs(puls->E[pmax]-puls->dh*puls->hoehe[
                pmax])<(puls->dh*ueber->Genauigkeit/krist->
                kristalle/2))&&(pmax!=0))
102         {
103             pmax=pmax-1;
104         }
105         dwa=0;
106         ueber->Abbruch=0;
107         //defining in which direction the crystal will be
                turned (dwa) and which crystall will be
                turned (n)
108
```

```
109             hilf=puls->E[pmax];
110
111             //chosing crystall depending on chosen peak
112
113             if((pmax==0)||(pmax==krist->kristalle))
114             {
115                 if(pmax==0)
116                 {
117                     n=krist->kristalle-pmax;
118                 }
119                 if(pmax==krist->kristalle)
120                 {
121                     n=krist->kristalle-pmax+1;
122                 }
123             }
124             if((pmax!=0)&&(pmax!=krist->kristalle))
125             {
126                 n=krist->kristalle-pmax;
127             }
128
129             // save old intensities
130             for(int i=0;i<=krist->kristalle;++i)
131             {
132                 alt_E[i]=puls->E[i];
133             }
134             std::cout<<"\n";
135
136             double hilf3=0;
137             if(w->Theta[n]>w->Winkel_Grenzen[n][1])
138             {
139                 w->Theta[n]=w->Winkel_Grenzen[n][1];
140             }
141             if(w->Theta[n]<w->Winkel_Grenzen[n][0])
142             {
143                 w->Theta[n]=w->Winkel_Grenzen[n][0];
144             }
145             hilf3=fabs(puls->E[pmax]-puls->dh*puls->hoehe[
                    pmax]);
146             w->Theta[n]=w->Theta[n]+dthetamin;
147
148             // checking if angle is out of borders
149             if((w->Theta[n]>w->Winkel_Grenzen[n][1])||(w->
                    Theta[n]<w->Winkel_Grenzen[n][0]))
150             {
151                 if(w->Theta[n]>w->Winkel_Grenzen[n][1])
152                 {
```

```
153                 w->Theta[n]=w->Winkel_Grenzen[n][1];
154                 dwa=-1;
155             }
156             if(w->Theta[n]<w->Winkel_Grenzen[n][0])
157             {
158                 w->Theta[n]=w->Winkel_Grenzen[n][0];
159                 dwa=-1;
160             }
161         }
162         // if angle not out of borders set direction in
                which the angle is changed
163         else
164         {
165             Intensitaet(ueber, krist, puls, w);
166             if(hilf3>fabs(puls->E[pmax]-puls->dh*puls->
                    hoehe[pmax]))
167             {
168                 dwa=1;
169             }
170             if(hilf3<fabs(puls->E[pmax]-puls->dh*puls->
                    hoehe[pmax]))
171             {
172                 dwa=-1;
173             }
174         }
175         // set back angle and intensity
176         w->Theta[n]=w->Theta[n]-dthetamin;
177         //Intensitaet(ueber, krist,puls,w);
178
179         for(int i=0;i<=krist->kristalle;++i)
180         {
181             puls->E[i]=alt_E[i];
182         }
183
184         //test if every peak fits to given form if not
                start loop
185
186         for(int i=0;i<=krist->kristalle;i++)
187         {
188             if(fabs(puls->dh*puls->hoehe[i]-puls->E[i])
                    <=(puls->dh*ueber->Genauigkeit))
189             {
190                 ueber->ende=2;
191             }
192             else
193             {
```

```
194                     ueber->ende=0;
195                     break;
196                 }
197
198             }
199
200             for(int i=0;i<=krist->kristalle;++i)
201             {
202                 std::cout<<puls->E[i]<<"\n";
203             }
204
205             std::cout<<"average:"<<puls->dh<<"\n"<<"peak
                    changes:"<<pmax<<"\n";
206             // turning crystall until acurcy is reached
207
208             while(((fabs(puls->dh*puls->hoehe[pmax]-puls->E[
                    pmax]))>(puls->dh*ueber->Genauigkeit/krist->
                    kristalle/2))&&ueber->ende==0)
209             {
210                 /*
211                 // checking if angle is out of borders
212                 if((w->Theta[n]>=w->Winkel_Grenzen[n][1])||(w
                        ->Theta[n]<=w->Winkel_Grenzen[n][0]))
213                 {
214                     if(w->Theta[n]>w->Winkel_Grenzen[n][1])
215                     {
216                         w->Theta[n]=w->Winkel_Grenzen[n][1];
217                         ueber->Abbruch=3;
218                     }
219                     if(w->Theta[n]<w->Winkel_Grenzen[n][0])
220                     {
221                         w->Theta[n]=w->Winkel_Grenzen[n][0];
222                         ueber->Abbruch=4;
223                     }
224                 }*/
225                 // checking if angle is smaller than the
                        angle of the last crystal/ bigger than the
                        angle of the next crystall
226                 if(pmax!=0&&n!=1)
227                 {
228                     if((w->Theta[n]>=w->Theta[n+1])&&(dwa>0))
229                     {
230                         dwa=-1;
231                         if(hilf==1)
232                         {
233                             ueber->Abbruch=2;
```

```
234                         hilf=0;
235                     }
236                     hilf=1;
237                 }
238                 if((w->Theta[n]<=w->Theta[n-1])&&(dwa<0))
239                 {
240                     dwa=1;
241                     if(hilf==1)
242                     {
243                         ueber->Abbruch=2;
244                         hilf=0;
245                     }
246                     hilf=1;
247                 }
248             }
249             if(pmax==0&&n<krist->kristalle)
250             {
251                 if((w->Theta[n]>=w->Theta[n+1])&&(dwa>0))
252                 {
253                     dwa=-1;
254                     if(hilf==1)
255                     {
256                         ueber->Abbruch=2;
257                         hilf=0;
258                     }
259                     hilf=1;
260                 }
261             }
262             if(ueber->Abbruch>1)
263             {
264                 break;
265             }
266
267             // Test if peak is smaller than the last one
                    and if the current peak is higher than the
                     average peak heigth,
268             // to prevent oscillation around the average
                    value. This is specialy needed if phase
                    shift is not equal 0.5 Pi.
269             if(pmax<krist->kristalle-1)
270             {
271                 if((puls->E[pmax]>(puls->dh*puls->hoehe[
                        pmax]))&&((puls->E[pmax]/puls->hoehe[
                        pmax])<(puls->E[pmax+1]/puls->hoehe[
                        pmax+1])))
272                 {
```

```
273                     ueber->ende=1;
274                 }
275             }
276
277             if(ueber->ende==0)
278             {
279                 //adjusting angle steps
280
281                 hilf4=fabs(puls->E[pmax]-puls->dh*puls->
                        hoehe[pmax]);
282                 hilf2=w->Theta[n];
283
284                 //save old values of puls->E[] und puls->
                        dh
285
286                 for(int i=0;i<=krist->kristalle;++i)
287                 {
288                     alt_E[i]=puls->E[i];
289                 }
290                 alt_dh=puls->dh;
291
292                 w->Theta[n]=w->Theta[n]+dwa*w->dtheta;
293
294                 // checking if angle is out of borders
295                 if((w->Theta[n]>w->Winkel_Grenzen[n][1])
                        ||(w->Theta[n]<w->Winkel_Grenzen[n
                        ][0]))
296                 {
297                     if(w->Theta[n]>w->Winkel_Grenzen[n
                            ][1])
298                     {
299                         w->Theta[n]=w->Winkel_Grenzen[n
                                ][1];
300                         ueber->Abbruch=3;
301                     }
302                     if(w->Theta[n]<w->Winkel_Grenzen[n
                            ][0])
303                     {
304                         w->Theta[n]=w->Winkel_Grenzen[n
                                ][0];
305                         ueber->Abbruch=4;
306                     }
307                 }
308
309                 // if angle not out of borders continue
                        loop
```

```
310                 else
311                 {
312                     Intensitaet(ueber, krist, puls, w);
313                     dPeakhoehe(/*ueber,*/krist,puls/*,w*/
                            );
314
315                     /* test if accuracy is reached after
                            last change of the angle. If the
                            accuracy is reached set back angle
                             and Intensities and proceed with
                            smaller angle step and accuracy.
316                 If the difference between peak heigth and
                     average peak height is growing, set
                    angle step to a smaller value and and
                    set back values of the angle and the
                    intensity.*/
317                     if(((genau*puls->dh)>fabs(puls->E[
                            pmax]-puls->dh*puls->hoehe[pmax]))
                            &&(genau>ueber->Genauigkeit/krist
                            ->kristalle/2))
318                     {
319
320                         if(w->dtheta==0)
321                         {
322                             ueber->ende=2;
323                         }
324                         std::cout<<"accuracy reached: "<<
                                genau<<" -> reduce step width
                                of angle/ accuracy "<<"\n";
325                         genau=genau*0.4;
326                         w->dtheta=0.5*w->dtheta;
327                         w->Theta[n]=hilf2;
328                         for(int i=0;i<=krist->kristalle
                                ;++i)
329                         {
330                             puls->E[i]=alt_E[i];
331                         }
332                         puls->dh=alt_dh;
333                     }
334                     else
335                     {
336                         if(hilf4<fabs(puls->E[pmax]-puls
                                ->dh*puls->hoehe[pmax]))
337                         {
338                             w->dtheta=0.5*w->dtheta;
339                             w->Theta[n]=hilf2;
```

```
340                             for(int i=0;i<=krist->
                                    kristalle;++i)
341                             {
342                                  puls->E[i]=alt_E[i];
343                             }
344                             puls->dh=alt_dh;
345                             std::cout<<"reduce step width
                                    of angle"<<"\n";
346                         }
347                     }
348                 }
349
350
351                 // stop loop if minimal step width of the
                        angle is reached
352
353                 if(w->dtheta<dthetamin)
354                 {
355                     std::cout<<"stop: step width of angle
                            to small"<<"\n";
356                     break;
357                 }
358             }
359         }
360
361         // next crystal
362         if(pmax==0)
363         {
364             pmax=krist->kristalle-1;
365         }
366         else
367         {
368             pmax=pmax-1;
369         }
370
371         //refresh plot of current pulse shape
372         timer->maxpeak=0;
373         timer->minpeak=0;
374         for(int i=0;i<=krist->kristalle;++i)
375         {
376             timer->e[i]=puls->E[i];
377             if(timer->e[timer->maxpeak]<puls->E[i])
378             {
379                 timer->maxpeak=i;
380             }
381             if(timer->e[timer->minpeak]>puls->E[i])
```

```
382                 {
383                     timer->minpeak=i;
384                 }
385             }
386             emit updategraf(timer);
387 
388             // check if simulation stopped by user
389 
390             if(ueber->ende==1)
391             {
392                 ueber->ende=0;
393             }
394 
395             // emit signal with current data if algorythm is
                    stopped by any reason
396             if(ueber->ende>0||ueber->endeExtern>0)
397             {
398                 ueber->ende=1;
399                 std::cout<<"search ended"<<"\n";
400                 emit AlgorithmusDatenUpdate(puls, w);
401                 usleep(200000);
402                 break;
403             }
404         }
405 
406         // get timestamps at the end of the Simulation
407         gettimeofday(&ende,NULL);
408         clock_end=clock();
409 
410         // output runtime measurements
411 
412         std::cout<<"Iterations:"<<iteration<<"\n"<<"\n";
413         std::cout<<"Time"<<"\n"<<"\n";
414         std::cout<<"start time:"<<start.tv_sec<<"\n"<<"end
                time:"<<ende.tv_sec<<"\n";
415         Zeit=((double)ende.tv_sec+(double)ende.tv_usec
                /1000000-(double)start.tv_sec+(double)start.
                tv_usec/1000000);
416         std::cout<<"time needed:"<<Zeit<<"sec"<<"\n"<<"\n";
417         std::cout<<"Computition Time"<<"\n"<<"\n";
418         std::cout<<"start time:"<<clock_start<<"\n"<<"end
                time:"<<clock_end<<"\n";
419         std::cout<<"clocks per second:"<<CLOCKS_PER_SEC<<"\n"
                ;
420         std::cout<<"time needed:"<<clock_end-clock_start<<"
                clocks"<<"\t"<<(float)(clock_end-clock_start)/
```

```
        CLOCKS_PER_SEC<<"sec"<<"\n";
421
422     std::cout<<"counter:"<<ueber->counter<<"\n";
423
424     std::cout<<"\n"<<"finished"<< "\n";
425
426     // emit signal that the algorithm finished his work
427     emit beendet();
428
429     delete ueber;
430     delete krist;
431     delete w;
432     delete puls;
433 }
```

A.7 Algorithmus.h

```
1  #ifndef ALGORITHMUS_H
2  #define ALGORITHMUS_H
3  #include <QWidget>
4  #include <mainwindow.h>
5  #include <QString>
6  #include "global.h"
7
8
9
10 class Algorithmus : public QObject
11 {
12      Q_OBJECT
13
14
15 public:
16     Algorithmus(Uebergabe *ueberP, Kristalle *kristP,
           Laserpuls *pulsP, Winkel *wP, zeit *z);
17
18     ~Algorithmus();
19
20     Kristalle *krist;
21     Winkel *w;
22     Laserpuls *puls;
23     Uebergabe *ueber;
24     zeit *timer;
25
26 public slots:
27      void ausfuehren();
28      //void stoppAlgorithmus(int Abbruch);
```

```
29
30 signals:
31     void beendet();
32     void AlgorithmusDatenUpdate(Laserpuls *pulsP, Winkel
           *wP);
33     void updategraf(zeit *z);
34
35 };
36
37 #endif // WORKER_H
```

A.8 running.cpp

```
1  #include "running.h"
2  #include "ui_running.h"
3  #include "iostream"
4  #include "mainwindow.h"
5  #include <QtWidgets>
6
7  #ifndef var
8  #define var
9  /*
10 Winkel w;
11 Kristalle krist;
12 Uebergabe ueber;
13 Laserpuls puls;
14 */
15 #endif
16
17 running::running(QWidget *parent) :
18     QDialog(parent),
19     ui(new Ui::running)
20 {
21     ui->setupUi(this);
22     ueber = NULL;
23
24 }
25
26 running::~running()
27 {
28     delete ui;
29 }
30 // setting up window
31 void running::verbinde(Uebergabe *ueberP, zeit *z)
32 {
33     ueber = ueberP;
```

```
34      timer = z;
35      ui->label->setText("simulation running");
36
37  }
38
39  // set plotrange to number of peaks
40  void running::anzahlkristalle(int k)
41  {
42      anzkristalle=k;
43      ui->widget->xAxis->setRange(0,anzkristalle+1);
44  }
45
46  // plot graph of peak heigth while simulation is running
47  void running::graph(zeit *z)
48  {
49      timer = z;
50      double dE=0;
51      QVector<double> x(anzkristalle+1),y(anzkristalle+1);
52      for(int i=0; i<=anzkristalle;++i)
53      {
54          x[i]=i;
55          y[i]=timer->e[i]*100;
56          dE=dE+timer->e[i];
57      }
58      dE=dE/(anzkristalle+1)*100;
59      ui->widget->yAxis->setRange(timer->e[timer->minpeak
            ]*100,timer->e[timer->maxpeak]*100);
60      ui->widget->clearGraphs();
61      ui->widget->addGraph();
62      ui->widget->graph(0)->setScatterStyle(QCPScatterStyle
            (QCPScatterStyle::ssCircle,4));
63      ui->widget->graph(0)->setData(x,y);
64      x.resize(2);
65      y.resize(2);
66      x[0]=0;
67      y[0]=dE;
68      x[1]=anzkristalle;
69      y[1]=dE;
70      ui->widget->addGraph();
71      ui->widget->graph(1)->setPen(QColor(0,100,0,255));
72      ui->widget->graph(1)->setData(x,y);
73      ui->widget->replot();
74  }
75
76  // end simulation by pushing button
77  void running::on_pushButton_clicked()
```

```
78 {
79     if (ueber != NULL)
80     {
81         ueber->endeExtern = 1;
82         ueber = NULL;
83 
84     }
85 }
86 
87 // set text / stop simulation if someone hit the stop
       button
88 void running::fensterschliessen()
89 {
90     //std::cout << "Simulation beendet" << std::endl;
91     ui->label->setText("simulation finished");
92     emit quit();
93 }
94 
95 // set text if simulation started
96 void running::start()
97 {
98     ui->label->setText("simulation running");
99 }
```

A.9 running.h

```
1  #ifndef RUNNING_H
2  #define RUNNING_H
3  
4  #include <QDialog>
5  #include "global.h"
6  #include "QTimer"
7  
8  
9  namespace Ui {
10 class running;
11 }
12 
13 class running : public QDialog
14 {
15     Q_OBJECT
16 
17 public:
18     explicit running(QWidget *parent = 0);
19     ~running();
20 
```

```
21      void verbinde(Uebergabe *ueberP, zeit *z);
22
23      Uebergabe *ueber;
24
25      zeit *timer;
26
27      double anzkristalle;          // number of crystals
28
29
30
31  private slots:
32      void on_pushButton_clicked();
33
34      void fensterschliessen();
35
36      void start();
37
38      void anzahlkristalle(int k);
39
40      void graph(zeit*z);
41
42  signals:
43      void quit();
44
45  private:
46      Ui::running *ui;
47  };
48
49  #endif // RUNNING_H
```

A.10 erweiterte_einstellungen.cpp

```
1   #include "erweiterte_einstellungen.h"
2   #include "ui_erweiterte_einstellungen.h"
3   #include "global.h"
4   #include "iostream"
5   #include "unistd.h"
6   #include "QKeyEvent"
7
8   Erweiterte_einstellungen::Erweiterte_einstellungen(
        QWidget *parent) :
9       QDialog(parent),
10      ui(new Ui::Erweiterte_einstellungen)
11  {
12      ui->setupUi(this);
13      connect(this,SIGNAL(schliesen()),this,SLOT(close()));
```

```
14
15
16      grafic = new grafic_input;
17  }
18
19  Erweiterte_einstellungen::~Erweiterte_einstellungen()
20  {
21      delete ui;
22  }
23
24  // building window
25
26  void Erweiterte_einstellungen::Daten_Einstellung(
        Uebergabe *u,Kristalle *k,Laserpuls *l,Winkel *wi)
27  {
28      ueber=u;
29      puls=l;
30      w=wi;
31      krist=k;
32      ueberlappende_pulse=false;
33      jones_intern=ueber->jones;
34      brechungsindex_intern=ueber->Brechungsindex;
35      for(int i=0;i<krist->kristalle;++i)
36      {
37          if(i==krist->kristalle)
38          {
39              polarisator_intern=true;
40          }
41          if(krist->polarisator[i]==false)
42          {
43              polarisator_intern=false;
44              break;
45          }
46      }
47      ui->ueberlappende_pulse->setChecked(
            ueberlappende_pulse);
48      ui->Polarisator->setChecked(polarisator_intern);
49      ui->kein_polarisator->setChecked(!polarisator_intern)
            ;
50      ui->Jones->setChecked(jones_intern);
51      ui->Jones->setEnabled(false);
52      ui->Brechungsindex->setChecked(brechungsindex_intern)
            ;
53      ui->Phasenverschiebung->setChecked(!
            brechungsindex_intern);
54      ui->Fehler->setText("");
```

```
55      ui->winkel_endpolarisator->setText(QString("%1").arg(
            krist->winkel_polarisator[krist->kristalle-1]*180/
            Pi,0,'f',3));
56      QTableWidgetItem *tooltip=ui->Peakhoehen->
            horizontalHeaderItem(0);
57      tooltip->setToolTip("Heigth relativly to average
            heigth of the peaks");
58      tabelle();
59      tabelle_hoehe();
60      if(krist->kristalle<1)
61      {
62          ui->grafic_input->setEnabled(false);
63      }
64      else
65      {
66          ui->grafic_input->setEnabled(true);
67      }
68  }
69
70  // building table for the borders of the crystall angles
71
72  void Erweiterte_einstellungen::tabelle()
73  {
74      ui->grenzen->setRowCount(krist->kristalle);
75      for (int i = 0; i <=1; i++)
76      {
77          for (int j = 0; j <= krist->kristalle-1; j++)
78          {
79              double whilf=0;
80              if(i==0)
81              {
82                  ui->grenzen->setVerticalHeaderItem(j,new
                        QTableWidgetItem("crystal No."+QString
                        ::number(j+1)));
83                  whilf=(180/Pi*w->Winkel_Grenzen[j+1][0]);
84                  ui->grenzen->setItem( j, i,new
                        QTableWidgetItem(QString("%1").arg(
                        whilf,0,'f',6) ));
85              }
86              if(i==1)
87              {
88                  whilf=(180/Pi*w->Winkel_Grenzen[j+1][1]);
89                  ui->grenzen->setItem( j, i,new
                        QTableWidgetItem(QString("%1").arg(
                        whilf,0,'f',6) ));
90              }
```

```
91          }
92      }
93  }
94
95  //building table for the wanted higth of the peaks for
        the simulation
96
97  void Erweiterte_einstellungen::tabelle_hoehe()
98  {
99      ui->Peakhoehen->setRowCount(krist->kristalle+1);
100     for(int i=0; i<=krist->kristalle;++i)
101     {
102         ui->Peakhoehen->setVerticalHeaderItem(i,new
                QTableWidgetItem("on position of peak No."+
                QString::number((i+1))));
103         ui->Peakhoehen->setItem(i,0,new QTableWidgetItem(
                QString("%1").arg(puls->hoehe[i]*100)));
104     }
105 }
106
107 // button deactivated
108 void Erweiterte_einstellungen::on_Jones_clicked()
109 {
110     if(jones_intern==true)
111     {
112         jones_intern=false;
113     }
114     else
115     {
116         jones_intern=true;
117
118     }
119 }
120
121
122 // polarisator after every crystal or not
123 void Erweiterte_einstellungen::on_Polarisator_clicked()
124 {
125     if(polarisator_intern==true)
126     {
127         polarisator_intern=false;
128         ui->kein_polarisator->setChecked(!
                polarisator_intern);
129     }
130     else
131     {
```

```
132             polarisator_intern=true;
133             ui->kein_polarisator->setChecked(!
                    polarisator_intern);
134         }
135     }
136
137     // no polarizer after every crystal
138     void Erweiterte_einstellungen::
            on_kein_polarisator_clicked()
139     {
140         if(polarisator_intern==true)
141         {
142             polarisator_intern=false;
143             ui->Polarisator->setChecked(polarisator_intern);
144         }
145         else
146         {
147             polarisator_intern=true;
148             ui->Polarisator->setChecked(polarisator_intern);
149         }
150     }
151
152     // dellete current changes
153     void Erweiterte_einstellungen::on_Abbruch_clicked()
154     {
155         emit schliesen();
156     }
157
158     // save settings
159
160     void Erweiterte_einstellungen::on_Speichern_clicked()
161     {
162         bool fehler=false;
163         QString a;
164         QTableWidgetItem *b;
165         QPalette color;
166         double hilf2;
167         color.setColor(ui->Fehler->foregroundRole(),Qt::red);
168         color.setColor(ui->Fehler->backgroundRole(),Qt::red);
169         ui->Fehler->setPalette(color);
170         //polarizers
171         if(polarisator_intern==true)
172         {
173             for(int i=0;i<krist->kristalle;++i)
174             {
175                 krist->polarisator[i]=true;
```

```
176             }
177         }
178         if(polarisator_intern==false)
179         {
180             for(int i=0;i<krist->kristalle-1;++i)
181             {
182                 krist->polarisator[i]=false;
183             }
184         }
185         // refraction index or phase shift
186         ueber->Brechungsindex=brechungsindex_intern;
187         for(int i=0;i<krist->kristalle;++i)
188         {
189
190             b=ui->grenzen->item(i,0);
191             w->Winkel_Grenzen[i+1][0]=b->text().toDouble()*Pi
                    /180;
192             b=ui->grenzen->item(i,1);
193             w->Winkel_Grenzen[i+1][1]=b->text().toDouble()*Pi
                    /180;
194         }
195         hilf2=0;
196         // checking for input errors in the table
197         for(int i=0;i<=krist->kristalle;++i)
198         {
199             double hilf;
200             b=ui->Peakhoehen->item(i,0);
201             hilf=b->text().toDouble()/100;
202             hilf2=hilf2+hilf;
203             if(hilf<0)
204             {
205                 fehler=true;
206                 ui->Fehler->setText("error:␣negativ␣heigth␣of
                        ␣puls(table)");
207             }
208             if(ueberlappende_pulse==false)
209             {
210                 if(krist->kristalle==i)
211                 {
212                     if(((krist->kristalle+1-0.001)>hilf2)||((
                            krist->kristalle+1+0.001)<hilf2))
213                     {
214                         fehler=true;
215                         ui->Fehler->setText("error:␣sum␣not␣
                                equal␣100%(table)");
216                     }
```

```
217                 }
218             }
219         }
220         double winkel_pol=0;
221 
222         // angle of the last polarizer
223         winkel_pol=ui->winkel_endpolarisator->text().toDouble
                ()*Pi/180;
224 
225         //write all data in global variables
226         if(fehler==false)
227         {
228             for(int i=0;i<maxkrist;++i)
229             {
230                 krist->winkel_polarisator[i]=winkel_pol;
231             }
232             for(int i=0;i<krist->kristalle-1;i++)
233             {
234                 krist->polarisator[i]=polarisator_intern;
235             }
236             for(int i=0;i<=krist->kristalle;++i)
237             {
238                 b=ui->Peakhoehen->item(i,0);
239                 puls->hoehe[i]=b->text().toDouble()/100;
240             }
241             emit brechungsindex();
242             emit daten(ueber,krist,puls,w);
243             emit schliesen();
244         }
245 }
246 
247 // input of refarction index
248 void Erweiterte_einstellungen::on_Brechungsindex_clicked
        ()
249 {
250     if(brechungsindex_intern==true)
251     {
252         brechungsindex_intern=false;
253         ui->Phasenverschiebung->setChecked(!
                brechungsindex_intern);
254     }
255     else
256     {
257         brechungsindex_intern=true;
258         ui->Phasenverschiebung->setChecked(!
                brechungsindex_intern);
```

```
259     }
260 }
261
262 // input of phase shift
263 void Erweiterte_einstellungen::
        on_Phasenverschiebung_clicked()
264 {
265     if(brechungsindex_intern==true)
266     {
267         brechungsindex_intern=false;
268         ui->Brechungsindex->setChecked(
              brechungsindex_intern);
269     }
270     else
271     {
272         brechungsindex_intern=true;
273         ui->Brechungsindex->setChecked(
              brechungsindex_intern);
274     }
275 }
276
277 // reset table with minimum/maximum angles of the
        crystalls
278 void Erweiterte_einstellungen::on_zurucksetzen_clicked()
279 {
280     tabelle();
281 }
282
283 // set broders of the angles of the crystalls +/- certain
        angle around the current crystal angle
284 void Erweiterte_einstellungen::on_pm_winkel_clicked()
285 {
286     QString Eingabe;
287     double c,a;
288     Eingabe=ui->winkel_eingabe->text();
289     c=Eingabe.toDouble();
290     for(int i=0;i<krist->kristalle;++i)
291     {
292         a=w->Theta[i+1]*180/Pi-c;
293         ui->grenzen->setItem(i,0,new QTableWidgetItem(
              QString("%1").arg(a)));
294         a=w->Theta[i+1]*180/Pi+c;
295         ui->grenzen->setItem(i,1,new QTableWidgetItem(
              QString("%1").arg(a)));
296     }
297 }
```

```
298
299 // reset table with heigth of the peaks
300 void Erweiterte_einstellungen::
        on_zuruecksetzten_pulshoehe_clicked()
301 {
302     tabelle_hoehe();
303 }
304
305 // should be checked if the table with the peak heigths
        is in summ 100%
306 void Erweiterte_einstellungen::
        on_ueberlappende_pulse_clicked()
307 {
308     if(ueberlappende_pulse==true)
309     {
310         ueberlappende_pulse=false;
311     }
312     else
313     {
314         ueberlappende_pulse=true;
315     }
316     emit ueberlappend(ueberlappende_pulse);
317 }
318
319 //open window for grafic input an connecting signals
320 void Erweiterte_einstellungen::on_grafic_input_clicked()
321 {
322     connect(this,SIGNAL(ueberlappend(bool)),grafic,SLOT(
            ueberlappende_pulse(bool)));
323     connect(this,SIGNAL(grafic_sig(Kristalle*,Laserpuls*)
            ),grafic,SLOT(laden(Kristalle*,Laserpuls*)));
324     connect(grafic,SIGNAL(tabelle(double,int)),this,SLOT(
            aktualisiere(double,int)));
325
326     emit ueberlappend(ueberlappende_pulse);
327     emit grafic_sig(krist,puls);
328     grafic->show();
329 }
330
331 //close window for grafic input if advanced options
        window is closed
332 void Erweiterte_einstellungen::closeEvent(QCloseEvent *)
333 {
334     grafic->close();
335 }
336
```

```
337 //refreshes table of pulse heigths if changed in window
        grafic input
338 void Erweiterte_einstellungen::aktualisiere(double hoehe,
        int puls)
339 {
340         ui->Peakhoehen->setItem(puls,0,new
                QTableWidgetItem(QString("%1").arg(hoehe*100))
                );
341 }
```

A.11 erweiterte_einstellungen.h

```
 1 #ifndef ERWEITERTE_EINSTELLUNGEN_H
 2 #define ERWEITERTE_EINSTELLUNGEN_H
 3
 4 #include <QDialog>
 5 #include "global.h"
 6 #include "grafic_input.h"
 7
 8 namespace Ui {
 9 class Erweiterte_einstellungen;
10 }
11
12 class Erweiterte_einstellungen : public QDialog
13 {
14     Q_OBJECT
15
16 public:
17     explicit Erweiterte_einstellungen(QWidget *parent =
            0);
18     ~Erweiterte_einstellungen();
19
20     Uebergabe *ueber;
21     Laserpuls *puls;
22     Kristalle *krist;
23     Winkel *w;
24
25 public slots:
26         void Daten_Einstellung(Uebergabe *u,Kristalle *k,
               Laserpuls *l,Winkel *wi);
27
28         void aktualisiere(double hoehe,int puls);
29
30 private slots:
31
32     void on_Jones_clicked();
```

```
33
34      void on_Polarisator_clicked();
35
36      void on_kein_polarisator_clicked();
37
38      void on_Abbruch_clicked();
39
40      void on_Speichern_clicked();
41
42      void on_Brechungsindex_clicked();
43
44      void on_Phasenverschiebung_clicked();
45
46      void tabelle();
47
48      void on_zurucksetzen_clicked();
49
50      void on_pm_winkel_clicked();
51
52      void tabelle_hoehe();
53
54      void on_zuruecksetzten_pulshoehe_clicked();
55
56      void on_ueberlappende_pulse_clicked();
57
58      void on_grafic_input_clicked();
59
60      void closeEvent(QCloseEvent *);
61
62  public:
63      grafic_input *grafic;

            //window for grafic input of the pulse shape
64
65
66  private:
67      Ui::Erweiterte_einstellungen *ui;
68      bool jones_intern;

            //which kind of equeation should be used
69
70      bool polarisator_intern;

            //placing the polarizers after crystals
71
72      bool brechungsindex_intern;
```

```
        //input of refraction index or phase shift
73
74      bool ueberlappende_pulse;

        //do the pulses overlap or not
75
76      int zeile,spalte;

        //row/column for tables
77
78  signals:
79      void schliesen();

        //signal that windows is closed
80
81      void brechungsindex();

        //signal to show input of refraction index and
        wavelength
82
83      void daten(Uebergabe *ueberP, Kristalle *kristP,
        Laserpuls *pulsP, Winkel *wP);      //send changed
        values
84
85      void grafic_sig(Kristalle* kristP,Laserpuls *pulsP);
                                            //pass data to
        grafic input window
86
87      void ueberlappend(bool);

        //emits if the pulses should be in sum 100%
88  };
89
90  #endif // ERWEITERTE_EINSTELLUNGEN_H
```

A.12 grafic_input.cpp

```
1  #include "grafic_input.h"
2  #include "ui_grafic_input.h"
3  #include "global.h"
4  #include "QKeyEvent"
5  #include "iostream"
6  #include "qcustomplot.h"
7  #include "QList"
8
```

```
 9  grafic_input::grafic_input(QWidget *parent) :
10      QDialog(parent),
11      Bewegung(new QTimer(this)),
12      ui(new Ui::grafic_input)
13  {
14      ui->setupUi(this);
15      ui->Graph->setInteraction(QCP::iSelectItems,true);
16
17      selecteditem=-1;
18      selected=false;
19
20      Bewegung->setInterval(25);
                                                          //
            refreshes the new position of the item [ms]
21
22      //conects signals from graph to the Window
23      connect(ui->Graph,SIGNAL(itemClick(QCPAbstractItem*,
            QMouseEvent*)),this,SLOT(select(QCPAbstractItem*,
            QMouseEvent*)));
24      connect(ui->Graph,SIGNAL(mouseRelease(QMouseEvent*)),
            this,SLOT(mouseReleaseEvent(QMouseEvent*)));
25      connect(ui->Graph,SIGNAL(mouseMove(QMouseEvent*)),
            this,SLOT(mouseMoveEvent(QMouseEvent*)));
26
27  }
28
29  grafic_input::~grafic_input()
30  {
31      delete ui;
32  }
33
34  // load data and build window
35  void grafic_input::laden(Kristalle *kristP, Laserpuls *
        pulsP)
36  {
37      krist=kristP;
38      puls=pulsP;
39      min=0;
40      max=0;
41      //find highest point
42      for(int i=0;i<=krist->kristalle;i++)
43      {
44          if(max<(puls->hoehe[i]*100))
45          {
46              max=puls->hoehe[i]*100;
47          }
```

```
48      }
49      max=max+max*0.1;
50      ui->obere_grenze->setText(QString("%1").arg(max,0,'f'
            ,2));
51      ui->Graph->xAxis->setLabel("Peak No");
52      ui->Graph->yAxis->setLabel("Heigth");
53      max=max+max*0.1;
54      ui->Graph->clearItems();
55      Punkte.clear();
56      Punkte.reserve(krist->kristalle);
                                                    // reserves
            storage space for points
57      for(int i=0;i<=krist->kristalle;i++)
58      {
59          Punkte.append(new QCPItemRect(ui->Graph));
60          ui->Graph->addItem(Punkte[i]);
61      }
62      graph();
63  }
64
65  //plot data into graph
66  void grafic_input::graph()
67  {
68      QVector<double>x(1),y(1);
69      double Durchschnitt=0;
70
71      x.resize(krist->kristalle+1);
72      y.resize(krist->kristalle+1);
73      ui->Graph->xAxis->setRange(-1,krist->kristalle+2);
74      ui->Graph->yAxis->setRange(min,max);
75      if(ueberlappende_Pulse==false)
76      {
77          //shift all peaks that the average heigth is 100
78          for(int i=0;i<=krist->kristalle;i++)
79          {
80              Durchschnitt=Durchschnitt+puls->hoehe[i];
81          }
82          Durchschnitt=Durchschnitt/(krist->kristalle+1);
83          for(int i=0;i<=krist->kristalle;i++)
84          {
85              puls->hoehe[i]=puls->hoehe[i]/Durchschnitt;
86          }
87      }
88      for(int i=0;i<=krist->kristalle;i++)
89      {
90          x[i]=i;
```

```
91              y[i]=puls->hoehe[i]*100;
92          }
93
94
95          for(int i=0;i<=krist->kristalle;i++)
96          {
97              Punkte[i]->bottomRight->setCoords(i-0.01*(krist->
                    kristalle+2),(puls->hoehe[i]*100-(max-min)
                    /100));
98              Punkte[i]->topLeft->setCoords((i+0.01*(krist->
                    kristalle+2)),(puls->hoehe[i]*100+(max-min)
                    /100));
99              Punkte[i]->setBrush(QBrush(Qt::blue));
100             Punkte[i]->setSelectable(true);
101             emit tabelle(puls->hoehe[i],i);
102         }
103         ui->Graph->replot();
104 }
105
106
107
108 // accept changes by pressing key
109 void grafic_input::keyReleaseEvent(QKeyEvent *event)
110 {
111         if(event->key()==Qt::Key_Return)
112         {
113             max=ui->obere_grenze->text().toDouble();
114             min=ui->untere_grenze->text().toDouble();
115             graph();
116         }
117 }
118
119
120 //releases point if a point if clipped to the mouse
121 void grafic_input::mouseReleaseEvent(QMouseEvent *event)
122 {
123         if(selected==true)
124         {
125             selected=false;
126             Punkte[selecteditem]->setSelected(false);
127             std::cout<<"selection reversed"<<"\n";
128             graph();
129         }
130 }
131
132 //moves point if the one is clipped to the mouse
```

```
133 void grafic_input::mouseMoveEvent(QMouseEvent *event)
134 {
135     if(selected==true)
136     {
137         double mausposition;
138         mausposition=ui->Graph->yAxis->pixelToCoord(event
                ->pos().y());
139         puls->hoehe[selecteditem]=mausposition/100;
140     }
141     graph();
142 }
143
144 //action if user clickes on a item
145 void grafic_input::select(QCPAbstractItem *item,
        QMouseEvent *mouse)
146 {
147     std::cout<<ui->Graph->xAxis->pixelToCoord(mouse->x())
            <<"x"<<"\t"<<ui->Graph->yAxis->pixelToCoord(mouse
            ->y())<<"y"<<"\n";
148     //selects a item in the graph if no item is selected
149     if(selected==false)
150     {
151         for(int i=0; i<=krist->kristalle;i++)
152         {
153             if(Punkte[i]->selected()==true)
154             {
155                 selecteditem=i;
156                 Punkte[selecteditem]->setSelected(true);
157                 selected=true;
158                 std::cout<<"point selected:"<<"\t"<<i<<"\
                        n";
159             }
160         }
161     }
162     //deactivates selection if a item is selected
163     else
164     {
165         Punkte[selecteditem]->setSelected(false);
166         selected=false;
167     }
168 }
169
170 //activates or deactivates that the programm shift all
        peaks to an average heigth of 100
171 void grafic_input::ueberlappende_pulse(bool a)
172 {
```

```
173     ueberlappende_Pulse=a;
174 }
```

A.13 grafic_input.h

```
 1 #ifndef GRAFIC_INPUT_H
 2 #define GRAFIC_INPUT_H
 3
 4 #include <QDialog>
 5 #include "global.h"
 6 #include "qcustomplot.h"
 7 #include "qtimer.h"
 8
 9 namespace Ui {
10 class grafic_input;
11 }
12
13 class grafic_input : public QDialog
14 {
15     Q_OBJECT
16
17 public:
18     explicit grafic_input(QWidget *parent = 0);
19     ~grafic_input();
20
21     Laserpuls *puls;
22     Kristalle *krist;
23
24 private:
25     Ui::grafic_input *ui;
26
27     double max;
                                                    //
        upper border of the plot
28
29     double min;
                                                    //
        lower border of the plot
30
31     QList<QCPItemRect *> Punkte;
                                    //Points in plot
32
33     QTimer *Bewegung;
                                                    //to get
        signal if the point stopped moving
34
```

```
35      int selecteditem;
                                                        //store
            selected item id
36
37      bool selected;
                                                          //item
            selected
38
39      bool ueberlappende_Pulse;
                                                // test if the
            heigth of the pulses should be in sum 100 %
40
41
42  public slots:
43      void laden(Kristalle *kristP,Laserpuls *pulsP);
                    // load data
44
45      void ueberlappende_pulse(bool);
                                        // load if sum of pulses
            should be 100%
46
47  private slots:
48
49      void graph();
                                                          // plot
            graph
50
51      void keyReleaseEvent(QKeyEvent *);
                                // accept changes by pressing
             keys
52
53      void mouseReleaseEvent(QMouseEvent *);
                            // process Signal if mouse button
             released
54
55      void mouseMoveEvent(QMouseEvent *);
                                // process Signal if mouse
            moves an item
56
57      void select(QCPAbstractItem*, QMouseEvent*);
                        // select an item(points in the plot)
58
59  signals:
60      void tabelle(double,int);
                                                // send data of
            the plot to the table
```

```
61
62 };
63
64 #endif // GRAFIC_INPUT_H
```

A.14 Ergebnis.cpp

```
1  #include "ergebnis.h"
2  #include "ui_ergebnis.h"
3  #include "global.h"
4  #include "fstream"
5  #include "cmath"
6  #include "iostream"
7  #include "locale"
8
9
10 Ergebnis::Ergebnis(QWidget *parent) :
11     QDialog(parent),
12     ui(new Ui::Ergebnis)
13 {
14     ui->setupUi(this);
15 }
16
17 Ergebnis::~Ergebnis()
18 {
19     delete ui;
20 }
21
22 // save data to file
23 void Ergebnis::on_pushButton_clicked()
24 {
25     setlocale(LC_NUMERIC,"de_DE");
26     //write results in file
27     QString Name;
28     //input file name
29     Name=ui->Dateiname->text();
30     Name=Name+".txt";
31     QByteArray ba = Name.toLatin1();
32     const char *n = ba.data();
33     //check if file already exists
34     std::ifstream FileTest(n);
35     if(FileTest)
36     {
37         ui->Fehler->setText("file already exists");
38     }
39     else
```

```
40      {
41          //write data into txt file
42          FILE* Ergebnisse;
43          Ergebnisse=fopen(n,"w");
44          fprintf(Ergebnisse,"number of crystals:");
45          fprintf(Ergebnisse,"\t");
46          fprintf(Ergebnisse,"%i",krist->kristalle);
47          fprintf(Ergebnisse,"\n");
48          fprintf(Ergebnisse,"FWHM:");
49          fprintf(Ergebnisse,"\t");
50          fprintf(Ergebnisse,"%f",ueber->breite);
51          fprintf(Ergebnisse,"\n");
52          fprintf(Ergebnisse,"delay:");
53          fprintf(Ergebnisse,"\t");
54          fprintf(Ergebnisse,"\t");
55          fprintf(Ergebnisse,"%f",krist->dt);
56          fprintf(Ergebnisse,"\n");
57          fprintf(Ergebnisse,"error:");
58          fprintf(Ergebnisse,"\t");
59          fprintf(Ergebnisse,"%f",ueber->Genauigkeit);
60          fprintf(Ergebnisse,"\n");
61          fprintf(Ergebnisse,"angle last polarizer:");
62          fprintf(Ergebnisse,"\t");
63          fprintf(Ergebnisse,"%.16f",krist->
                winkel_polarisator[krist->kristalle-1]);
64          fprintf(Ergebnisse,"\n");
65
66          fprintf(Ergebnisse,"                angle[rad]
                angle[Grad]    phaseshift  angle relativ to
                SOLC angles[rad][grad]");
67          fprintf(Ergebnisse,"\n");
68          for (int i=1;i<=krist->kristalle;i++)
69          {
70              fprintf(Ergebnisse,"crystal No.");
71              fprintf(Ergebnisse,"%i",i);
72              fprintf(Ergebnisse,"\t");
73              fprintf(Ergebnisse,"%.16f",w->Theta[i]);
74              fprintf(Ergebnisse,"\t");
75              fprintf(Ergebnisse,"%.16f",(w->Theta[i]*180/
                    Pi));
76              fprintf(Ergebnisse,"\t");
77              fprintf(Ergebnisse,"%.16f",krist->
                    eingabePhasenverschiebung[i-1]);
78              fprintf(Ergebnisse,"\t");
79              fprintf(Ergebnisse,"\t");
80              fprintf(Ergebnisse,"%.16f",w->Theta[i]-(Pi
```

```
                    *2*45/360/krist->kristalle*(2*i-1)));
81                  fprintf(Ergebnisse,"\t");
82                  fprintf(Ergebnisse,"%.16f",(w->Theta[i]-(Pi
                        *2*45/360/krist->kristalle*(2*i-1)))*180/
                        Pi);
83                  fprintf(Ergebnisse,"\n");
84              }
85              fclose(Ergebnisse);
86              ui->Fehler->setText("file saved");
87          }
88
89  }
90
91  // get data from mainwindow
92  void Ergebnis::Daten_ergebnis(Uebergabe *u,Kristalle *k,
        Laserpuls *l,Winkel *wi)
93  {
94      ueber=u;
95      puls=l;
96      w=wi;
97      krist=k;
98  }
```

A.15 Ergebnis.h

```
1   #ifndef ERGEBNIS_H
2   #define ERGEBNIS_H
3
4   //#include "QDialog.h"
5   #include "mainwindow.h"
6   #include "global.h"
7
8   namespace Ui {
9   class Ergebnis;
10  }
11
12  class Ergebnis : public QDialog
13  {
14      Q_OBJECT
15
16  public:
17      explicit Ergebnis(QWidget *parent = 0);
18      ~Ergebnis();
19
20      Uebergabe *ueber;
21      Laserpuls *puls;
```

```
22      Kristalle *krist;
23      Winkel *w;
24
25  public slots:
26          void Daten_ergebnis(Uebergabe *u, Kristalle *k,
                Laserpuls *l, Winkel *wi);
27
28
29  private slots:
30      void on_pushButton_clicked();
31
32  private:
33      Ui::Ergebnis *ui;
34  };
35
36  #endif // ERGEBNIS_H
```

A.16 save_plot_data.cpp

```
1   #include "save_plot_data.h"
2   #include "ui_save_plot_data.h"
3   #include "QString"
4   #include "QFile"
5
6   save_plot_data::save_plot_data(QWidget *parent) :
7       QDialog(parent),
8       ui(new Ui::save_plot_data)
9   {
10      ui->setupUi(this);
11  }
12
13  save_plot_data::~save_plot_data()
14  {
15      delete ui;
16  }
17
18  void save_plot_data::on_save_clicked()
19  {
20      bool vorhanden=false;
21      QString name=ui->name->text();
22      // test if file name.png exists
23      if(QFile::exists(name+".png"))
24      {
25          vorhanden=true;
26      }
27      if(vorhanden==false)
```

```
28      {
29          // test if file name.txt exists if both do not
                exist files are saved
30          if(QFile::copy("plot",name+".txt"))
31          {
32              ui->error->setText("saved");
33              emit dateiname(name+".png");
34          }
35      }
36      // error if one of both files already exists
37      else
38      {
39          ui->error->setText("File with name "+name+".txt /.
                png already exists");
40      }
41  }
```

A.17 save_plot_data.h

```
1  #ifndef SAVE_PLOT_DATA_H
2  #define SAVE_PLOT_DATA_H
3
4  #include <QDialog>
5  #include "QString"
6
7  namespace Ui {
8  class save_plot_data;
9  }
10
11 class save_plot_data : public QDialog
12 {
13     Q_OBJECT
14
15 public:
16     explicit save_plot_data(QWidget *parent = 0);
17     ~save_plot_data();
18
19 private slots:
20     void on_save_clicked();
21
22 private:
23     Ui::save_plot_data *ui;
24
25 signals:
26     void dateiname(QString name);
27
```

```
28 };
29
30 #endif // SAVE_PLOT_DATA_H
```

A.18 kristallkonfiguration_laden.cpp

```
1  #include "kristallkonfiguration_laden.h"
2  #include "ui_kristallkonfiguration_laden.h"
3  #include "QFileDialog"
4  #include "QFileSystemModel"
5  #include "QTextStream"
6  #include "iostream"
7  #include "global.h"
8  #include "unistd.h"
9  #include "locale"
10 #include "QItemSelection"
11 #include "QItemSelectionModel"
12 #include "Funktionen.h"
13
14 Kristallkonfiguration_laden::Kristallkonfiguration_laden(
       QWidget *parent) :
15     QDialog(parent),
16     ui(new Ui::Kristallkonfiguration_laden)
17 {
18     ui->setupUi(this);
19     model =new QFileSystemModel;
20     model->setRootPath(QDir::currentPath());
21     ui->Dateiliste->setModel(model);
22     ui->Dateiliste->setColumnWidth(0,300);
23     auswahl=ui->Dateiliste->selectionModel();
24     connect(auswahl,SIGNAL(currentChanged(QModelIndex,
           QModelIndex)),this,SLOT(auswahl_geaendert(
           QModelIndex,QModelIndex)));
25 }
26
27 Kristallkonfiguration_laden::~Kristallkonfiguration_laden
       ()
28 {
29     delete ui;
30 }
31
32 // get data from mainwindow
33 void Kristallkonfiguration_laden::daten(Uebergabe *u,
       Kristalle *k, Laserpuls *l, Winkel *wi)
34 {
35     ueber=u;
```

```
36      krist=k;
37      puls=l;
38      w=wi;
39  }
40
41  // load file
42  void Kristallkonfiguration_laden::on_laden_clicked()
43  {
44      int anz=0;
45      QFile datei(pfad);
46      datei.open(QIODevice::ReadOnly);
47      QTextStream inhalt(&datei);
48      QString zeile = inhalt.readLine();
49      while(!zeile.isNull())
50      {
51          QStringList teile=zeile.split("\t");
52          teile.replaceInStrings(",",".");
53          // number of crystalls
54          if(anz==0)
55          {
56              krist->kristalle=teile[1].toInt();
57              std::cout<<"crystals:"<<"\t"<<krist->
                    kristalle<<"\n";
58          }
59          // FWHM of the incomming pulse
60          if(anz==1)
61          {
62              puls->tau=teile[1].toDouble();
63              std::cout<<"FWHM:"<<"\t"<<puls->tau<<"\n";
64          }
65          //delaytime by the crystalls
66          if(anz==2)
67          {
68              krist->dt=teile[2].toDouble();
69              std::cout<<"delay:"<<"\t"<<krist->dt<<"\n";
70          }
71          //accuracy of the Simultation
72          if(anz==3)
73          {
74              ueber->Genauigkeit=teile[1].toDouble();
75              std::cout<<"accuracy:"<<"\t"<<ueber->
                    Genauigkeit<<"\n";
76          }
77          //angle of the last polarizer
78          if(anz==4)
79          {
```

```
80                  krist->winkel_polarisator[krist->kristalle
                        -1]=teile[1].toDouble();
81                  std::cout<<"last polarizor:"<<"\t"<<krist->
                        winkel_polarisator[krist->kristalle-1]<<"\
                        n";
82              }
83              //angles an phaseshift of the crystalls
84              if(anz>5)
85              {
86                  w->Theta[anz-5]=teile[1].toDouble();
87                  krist->eingabePhasenverschiebung[anz-6]=teile
                        [3].toDouble();
88                  std::cout<<"angle:"<<"\t"<<w->Theta[anz-5]<<"
                        \t";
89                  std::cout<<"phase shift:"<<"\t"<<krist->
                        eingabePhasenverschiebung[anz-6]<<"\n";
90              }
91              anz=anz+1;
92              zeile = inhalt.readLine();
93          }
94          std::cout<<pfad.toStdString()<<"\n";
95          emit Daten(ueber, krist, puls, w);
96          datei.close();
97      }
98
99      // chose the file to load
100     void Kristallkonfiguration_laden::auswahl_geaendert(const
            QModelIndex &index1, const QModelIndex &index2)
101     {
102         pfad=model->fileInfo(index1).absoluteFilePath();
103     }
```

A.19 kristallkonfiguration_laden.h

```
1   #ifndef KRISTALLKONFIGURATION_LADEN_H
2   #define KRISTALLKONFIGURATION_LADEN_H
3
4   #include <QDialog>
5   #include <QFileSystemModel>
6   #include <global.h>
7   #include "QItemSelection"
8   #include "QItemSelectionModel"
9
10  namespace Ui {
11  class Kristallkonfiguration_laden;
12  }
```

```
13
14 class Kristallkonfiguration_laden : public QDialog
15 {
16     Q_OBJECT
17
18 public:
19     explicit Kristallkonfiguration_laden(QWidget *parent
           = 0);
20     ~Kristallkonfiguration_laden();
21
22     Uebergabe *ueber;
23     Laserpuls *puls;
24     Kristalle *krist;
25     Winkel *w;
26
27 private slots:
28
29     void on_laden_clicked();
30
31     void auswahl_geaendert(const QModelIndex &index1,
           const QModelIndex &index2);
32
33 public slots:
34     void daten(Uebergabe *u,Kristalle *k,Laserpuls *l,
           Winkel *wi); // send values
35
36 private:
37     Ui::Kristallkonfiguration_laden *ui;
38
39     QFileSystemModel *model;
40
41     QItemSelectionModel *auswahl;
42
43     QString pfad;
44
45 signals:
46         void Daten(Uebergabe *ueberP, Kristalle *kristP,
               Laserpuls *pulsP, Winkel *wP);
47 };
48
49 #endif // KRISTALLKONFIGURATION_LADEN_H
```

A.20 oss_datei_laden.cpp

```
1 #include "oss_datei_laden.h"
2 #include "ui_oss_datei_laden.h"
```

```
3  #include "QFileDialog"
4  #include "QFileSystemModel"
5  #include "QTextStream"
6  #include "iostream"
7  #include "global.h"
8  #include "unistd.h"
9
10 OSS_Datei_laden::OSS_Datei_laden(QWidget *parent) :
11     QDialog(parent),
12     ui(new Ui::OSS_Datei_laden)
13 {
14     ui->setupUi(this);
15     model =new QFileSystemModel;
16     oss =new OSSDatenuebergabe;
17     // build window
18     model->setRootPath(QDir::currentPath());
19     ui->Dateiliste->setModel(model);
20     ui->Dateiliste->setColumnWidth(0,300);
21     ui->dateiendung->addItem("all");
22     ui->dateiendung->addItem(".oss");
23     ui->dateiendung->addItem(".txt");
24     oss->mitte=0;
25 }
26
27 OSS_Datei_laden::~OSS_Datei_laden()
28 {
29     delete ui;
30     delete oss;
31 }
32
33 // set filter for special file name endings
34 void OSS_Datei_laden::on_dateiendung_currentIndexChanged(
       int index)
35 {
36     QStringList filter;
37     if(index==0)
38     {
39         filter<<"*";
40     }
41     if(index==1)
42     {
43         filter<<"*.oss";
44     }
45     if(index==2)
46     {
47         filter<<"*.txt";
```

```
48      }
49      model->setNameFilters(filter);
50  }
51
52  // load oss file
53  void OSS_Datei_laden::on_datei_laden_clicked()
54  {
55      double x[1000],y[1000],nulllinie=0;
56      int anz=0,lauf=0;
57      QString a;
58      anz=zeilenanzahl();
59      std::cout<<"number of points:"<<"\t"<<anz<<"\n";
60      QFile datei(pfad);
61      datei.open(QIODevice::ReadOnly);
62      QTextStream inhalt(&datei);
63      QString zeile = inhalt.readLine();
64
65      // write data into vector for later processing
66      while(!zeile.isNull())
67      {
68          zeile = inhalt.readLine();
69          QStringList teile=zeile.split(";");
70          a=teile[0];
71              oss->ossx[lauf]=teile[0].toDouble();
72              oss->ossy[lauf]=teile[1].toDouble();
73              //std::cout<<oss->ossx[lauf]<<"\t"<<oss->ossy
                    [lauf]<<"\t"<<lauf<<"\n";
74              if(lauf==431)
75              {
76              }
77          lauf++;
78          if(inhalt.atEnd())
79          {
80              oss->ossmesspunkte=lauf;
81              break;
82          }
83      }
84
85      // find ground line
86      nulllinie=0;
87      for(int i=0;i<10;++i)
88      {
89          nulllinie=nulllinie+oss->ossy[i];
90      }
91      nulllinie=nulllinie/10;
92
```

```
93      //input of the heigth of the input pulse
94      a=ui->Hoehe_Eingangspuls->text();
95      oss->ausgangspulshoehe=a.toDouble();
96
97      // setting data to ground line and divide by heigth
            of the input pulse
98      for(int i=0;i<=oss->ossmesspunkte;i++)
99      {
100         oss->ossy[i]=(oss->ossy[i]-nulllinie)/oss->
                ausgangspulshoehe;
101     }
102     oss->geladen=true;
103
104     emit OSSDaten(oss);
105     datei.close();
106     emit geladen();
107 }
108
109 // count number of points in the file
110
111 int OSS_Datei_laden::zeilenanzahl()
112 {
113     QFile datei(pfad);
114     int anzahlzeilen=0;
115     datei.open(QIODevice::ReadOnly);
116     QTextStream inhalt(&datei);
117     QString zeile = inhalt.readLine();
118     while(!zeile.isNull())
119     {
120         zeile = inhalt.readLine();
121         anzahlzeilen++;
122     }
123     datei.close();
124     return anzahlzeilen;
125 }
126
127 // get file path
128
129 void OSS_Datei_laden::on_Dateiliste_clicked(const
        QModelIndex &index)
130 {
131     pfad=model->fileInfo(index).absoluteFilePath();
132     std::cout<<pfad.toStdString()<<"\n";
133 }
134
135 // send data to mainwindow
```

```
136
137 void OSS_Datei_laden::Datenuebergabe_OSS(Uebergabe *u,
        Kristalle *k, Laserpuls *l, Winkel *wi)
138 {
139     ueber=u;
140     krist=k;
141     puls=l;
142     w=wi;
143 }
```

A.21 oss_datei_laden.h

```
 1 #ifndef OSS_DATEI_LADEN_H
 2 #define OSS_DATEI_LADEN_H
 3
 4 #include <QDialog>
 5 #include "QFileSystemModel"
 6 #include "global.h"
 7
 8 namespace Ui {
 9 class OSS_Datei_laden;
10 }
11
12 class OSS_Datei_laden : public QDialog
13 {
14     Q_OBJECT
15
16 public:
17     explicit OSS_Datei_laden(QWidget *parent = 0);
18     ~OSS_Datei_laden();
19
20     OSSDatenuebergabe *oss;
21     Uebergabe *ueber;
22     Laserpuls *puls;
23     Kristalle *krist;
24     Winkel *w;
25
26
27 public slots:
28
29     void Datenuebergabe_OSS(Uebergabe *u, Kristalle *k,
           Laserpuls *l, Winkel *wi);
30
31    // void Test();
32
33 signals:
```

```
34      void OSSDaten(OSSDatenuebergabe*Daten);
35
36      void geladen();
37
38  private slots:
39      void on_dateiendung_currentIndexChanged(int index);
40
41      void on_datei_laden_clicked();
42
43      void on_Dateiliste_clicked(const QModelIndex &index);
44
45      int zeilenanzahl();
46
47  private:
48      Ui::OSS_Datei_laden *ui;
49
50      QFileSystemModel *model;
51
52      QString pfad;
53
54  };
55
56  #endif // OSS_DATEI_LADEN_H
```