



MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

**Design and construction of a fibre-coupled grating-based
single-photon spectrometer with sub-nanometer resolution
at 1550 nm**

verfasst von / submitted by

Peter Schiansky BSc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc.)

Wien, 2017 / Vienna, 2017

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

A 066 876

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Masterstudium Physik UG2002

Betreut von / Supervisor:

Univ.-Prof. Dr. Philip Walther

Mitbetreut von / Co-Supervisor:

MSc. Chiara Greganti

Acknowledgements

Here, I want to express my deepest gratitude to everyone who supported me during my writing of this thesis:

First of all, I would like to thank my supervisor, Univ.-Prof. Dr. Philip Walther, not only for introducing me to the field of photonic quantum computation, but also for presenting me with the opportunities to work with his group and write this thesis, and for his help in improving it.

Furthermore, I want to express my gratefulness to all the members of the Walther group for making these last months such an unforgettable experience. I am particularly thankful for the advice given by and fruitful discussions with Chiara Greganti, Amir Moqanaki and Francesco Massa, who also helped me to get used to the real and everyday's life in the laboratory. My special thanks go to Chiara for invaluable discussions, guidance, explanations, support, and for her never-ending patience when things did not work out as expected.

Fabian Laudenbach and Michael Hentschel, from the Austrian Institute of Technology, provided valuable and much appreciated insights into the single-photon spectrometer design.

Last but not least, I would like to thank my family: my parents for their continuous support in all regards, and my sister Chii for supporting me while facing challenging times and for correcting my English and proof-reading this work.

Abstract

The utilisation of intrinsic properties of quantum systems can benefit several applications of different fields in physics which previously relied on classical systems, and has led to the development of new research areas, comprising quantum cryptography, quantum computation (including quantum simulation) and quantum metrology. Single photons serve as ideal quantum systems in many ways: They can offer a high degree of indistinguishability and their intrinsic mobility allows for the transportation of quantum information at the speed of light by exploiting established technology such as optical fibres and integrated optics. In addition, processing single photons does not require vacuum or cryogenic conditions, which underlines the applicability of optical platforms. The vast development of photonic quantum technology has led to scalable architectures based on integrated sources, processors and detectors. However, almost all multi-photon processing schemes are building on particular optical properties, such as multi-photon interference, that require a precise characterisation of the generated single photons.

Early multi-photon experiments relied on sources producing photons at around 800 nm wavelength. In recent years, however, focus has shifted towards sources operating at telecom wavelengths. This is due to the fact that the zero-dispersion wavelength of fused-silica fibres lies in this region and their attenuation is lowest there, with a minimum at 1550 nm. Since losses in individual single-photon sources exponentially degrade the efficiency of multi-photon sources, this wavelength is ideal for multi-photon experiments. The primary purpose of this thesis was to build a cost-effective but accurate monochromator with single-photon resolution at 1550 nm to help characterize single-photon sources operating at this wavelength.

The device is based on a planar diffraction grating, fibre coupled and offers a resolution of about 0.1 nm. It was used for the characterisation of single photons from pico-second pulsed PPKTP sources and helped to set up a multi-photon setup. This thesis covers the design process as well as the implementation of a control software and measurements of the sources.

Zusammenfassung

Diverse bisher üblicherweise auf Prinzipien der klassischen Physik beruhende Anwendungen unterschiedlicher Gebiete der Physik können aus der Nutzung intrinsischer Eigenschaften von Quantensystemen Profit ziehen. Diese Tatsache führte zur Entwicklung verschiedener neuer Forschungsgebiete wie, unter Anderem, Quanten Kryptographie, Quanten Computation (das auch das Gebiet der Quanten Simulation umfasst), und Quanten Metrologie. Hinsichtlich vieler Gesichtspunkte stellen Einzelphotonen ideale Quantensysteme dar: Sie können eine sehr hohe Ununterscheidbarkeit aufweisen und ihre intrinsische Mobilität ermöglicht den Transport von Quanteninformation mit Lichtgeschwindigkeit. Für letzteres kann auf bewährte Technologien zurückgegriffen werden, wie in etwa auf optische Glasfasern und integrierte Optik. Des Weiteren verlangt die Verarbeitung von Einzelphotonen nicht nach einem Vakuum oder kryogenen Bedingungen, was die Anwendbarkeit optischer Plattformen nochmals unterstreicht. Die rasante Entwicklung photonischer Quantentechnologie führte zu skalierbaren Architekturen, basierend auf integrierten Quellen, Verarbeitungseinheiten und Detektoren. Allerdings basieren die meisten Multiphotonenverarbeitungsmodelle auf spezifischen optischen Eigenschaften, wie Multiphotoneninterferenz, die eine präzise Charakterisierung der generierten Einzelphotonen unabdingbar machen.

Frühe Multiphotonenexperimente basierten auf Quellen die Photonen bei in etwa 800 nm Wellenlänge erzeugten. In den letzten Jahren verschob sich der Schwerpunkt allerdings zu Quellen die im Telekommunikationswellenlängenbereich operieren. Der Grund dafür liegt einerseits darin, dass die Nulldispersionswellenlänge von Quarzglasfasern in diesem Bereich liegt und andererseits dass deren Dämpfung dort am geringsten ist, mit einem Minimum bei 1550 nm. Da Verluste in individuellen Einzelphotonenquellen die Effizienz von Multiphotonenquellen exponentiell vermindern ist diese Wellenlänge ideal für Multiphotonenexperimente. Der Hauptzweck dieser Arbeit bestand darin, einen kosteneffizienten, dabei aber genauen, Monochromator mit Einzelphotonenauflösung bei 1550 nm zu konstruieren um die Charakterisierung von Einzelphotonenquellen bei dieser Wellenlänge zu unterstützen.

Das fertige Gerät basiert auf einem planen optischen Beugungsgitter, ist fasergekoppelt und erreicht eine Auflösung von in etwa 0.1 nm. Es wurde zur Charakterisierung von Einzelphotonen von Pikosekunden-gepulsten PPKTP Quellen eingesetzt und half dabei einen Multiphotonenaufbau einzurichten. Diese Arbeit umfasst den Entwicklungsprozess des Gerätes sowie die Implementierung einer Steuerungssoftware und Messungen an den Quellen.

Contents

1	Introduction	7
1.1	Elements of the spectrometer in detail	8
1.1.1	Diffraction Gratings	8
1.1.2	Off-Axis parabolic mirrors	11
1.1.3	Rotation motor	12
1.1.4	Optical fibres	19
1.2	Different types of spectrometers	22
1.2.1	Czerny-Turner mount	22
1.2.2	Littrow mount	23
1.2.3	Ebert-Fastie mount	24
1.2.4	Monk-Gillieson mount	24
1.2.5	Rowland mount	25
2	Determination of required components	26
3	Experimental Setup	30
4	Software	34
4.1	Motor control	35
4.2	Supported detectors	35
4.2.1	AIT TTM8000	35
4.2.2	Thorlabs PM100D	35
4.3	The measurement procedure	36
4.4	Calibration	38
4.5	Additional Features	39
4.5.1	Automatisation	39
4.5.2	Rough wavelength selection	39
5	Characterisation	40
5.1	Angular positioning uncertainty	40
5.2	Calibration	40
5.3	Bandpass	40
5.4	Wavelength range and wavelength accuracy	41
6	Applications	44
6.1	Spectral properties of a quasi phase matched collinear Type-II SPDC source in PPKTP around 1550nm	44
6.1.1	Background	44
6.1.2	Measurements	49

7 Conclusion and Outlook	56
Appendices	61
A Dinspect: Example spectrometer configuration file	62
B Dinspect: code documentation	64

Chapter 1

Introduction

Colour was not always believed to be an intrinsic property of light, rather it was thought of as being a property of every medium light travels through, or is reflected off[1]. This changed when Newton carried out various experiments using prisms and realized that this optical element does not colour the light, but separates colour already present in the light of the sun. The term "spectrum" was born [2, 3]. Up to now, the interest in determining the spectral components of different light sources and extracting knowledge out of these spectra has not abated: Since the development of early theories of absorption and emission[4] it is known that one can learn about the chemical constituents of media by studying absorption lines in their spectra[5]. This paved the way to study atmospheres of celestial bodies which, in turn, led to the observation of a chemical element (Helium) in our solar system even before it could be detected on earth [6].

The terminology used to classify spectrometric devices is as follows:

Spectrometric device: A device separating physical properties by wavelength.

Spectrometer: A spectrometric device, referring both to instruments scanning whole ranges of wavelengths at once as well as those selecting single wavelengths at a time.

Monochromator: A spectrometer which scans single wavelengths at a time, by imaging only a single band of wavelengths onto a slit.

Spectrograph: A spectrometer observing multiple parts of spectra at once. This can either be done by imaging a range of wavelengths onto a detector array or by imaging different bands of wavelengths onto multiple slits (the device is then sometimes called *polychromator*).

Spectrometers continue to prove beneficial in many fields like astronomy, biology, medicine, chemistry or material sciences. They are also used in quantum optics where spectrometers are, for example, utilised in the characterisation of entangled two photon states[7]. The device constructed in the course of this thesis was designed with the goal to be able to characterise a specific source of such states.

There are various types of spectrometer schemes, some of which will be presented later on in Sec. 1.2, after discussing in greater detail the properties of the components involved in such a setup. Therefore one exemplary setup will be showcased now to reason which elements are in need of further analysis. The sample setup is pictured in Fig. 1.1: Light with an unknown spectrum enters the apparatus through a slit (S1) after which it is collimated using a lens (L1). It hits the grating (G) which diffracts the beam into different directions, depending on the wavelength. One specific wavelength, which can be selected by rotating the grating, is focused by a

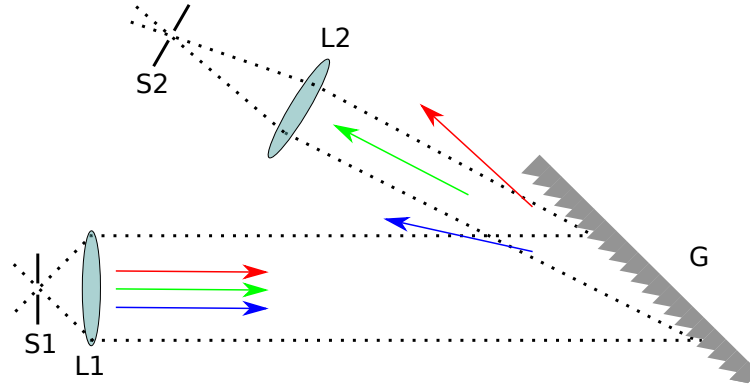


Figure 1.1: Exemplary scheme of a spectrometer setup: Light passes through a slit (S1) and is collimated by a lens (L1). A grating (G) diffracts light of different wavelengths into different directions. By rotating the grating, a specific wavelength is selected to be focused by another lens (L2) onto the exit slit (S2), after which it can be observed.

lens (L2) onto a slit (S2), at which the light exits the device.

In section 1.1, the components just specified will be examined. Section 1.2 will give an overview of different schemes of spectrometers. The calculation of the properties of the required components will be given in section 2. The real setup will be presented in section 3. Section 4 will deal with the description of the software written to control the device and process the data received from the detectors. An exemplary configuration file holding the necessary parameters will be discussed in appendix A and a thorough documentation of the code can be found in appendix B. Some measurements carried out with the device will be presented in section 6.

1.1 Elements of the spectrometer in detail

1.1.1 Diffraction Gratings

To use a more general term, this section may also be called 'diffracting or refracting element' as a spectrometer can of course be built using other optical elements which spatially separate incident light depending on its wavelength, for example prisms. The performance of diffraction gratings in spectrometric applications, however, can surpass the one of prisms nowadays (since the second half of the 19th century [8]).

While the first gratings described in literature were devices consisting of two screws with hair or thin wires spanned between them[9–11], today they are either ruled into various materials by a diamond tool or etched after photographically recording a stationary interference pattern[12].

Fig. 1.2 picturises the notation in use: A beam of light is incident on the grating with an angle α relative to the grating normal GN , which is the line perpendicular to the grating surface. For a fixed specific wavelength it is diffracted into several directions. The diffraction angles are denoted with β_m , where m is the diffraction order. D_v is the difference between the angle of incidence and the angle of diffraction. The angle between the grating surface and the facets is called blaze angle Θ_B .

If the path difference between two beams incident on different grating facets of a grating with groove period d (see Fig. 1.3), $d \cdot (\sin(\alpha) + \sin(\beta))$, is an integer

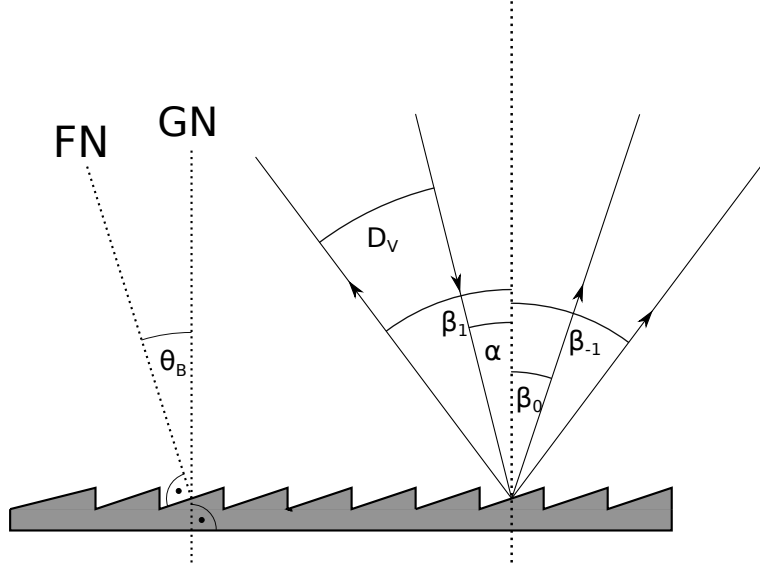


Figure 1.2: Sketch of a ruled diffraction grating. Angle of incidence α , angles of diffraction β_m , deviation angle D_v , blaze-angle Θ_B . GN=Grating normal, FN=Facet normal.

multiple of the wavelength of the incident light, constructive interference occurs. Thus, for a beam incident on a specific grating, under a certain fixed angle, different wavelengths are diffracted into different directions. In its mathematically expressed form, this behaviour is called the grating equation [11]:

$$G \cdot m \cdot \lambda = \sin(\alpha) + \sin(\beta) \quad (1.1)$$

where $G = 1/d$ is called grating constant or groove density.

One has to pay attention to the convention used to determine the sign of the angles. Here, α and β_m shall be assigned signs according to the mathematical direction of rotation, where these angles are measured starting from the grating normal. Hence in Fig. 1.2, $\alpha, \beta_1 > 0$ and $\beta_0, \beta_{-1} < 0$. Thus, the angle between incident and diffracted beam, called the deviation angle D_v , is

$$D_v = \beta - \alpha. \quad (1.2)$$

Using this and Equ. 1.1, one obtains the following relations for the angles of incidence and diffraction:

$$\alpha = \arcsin\left(\frac{G \cdot m \cdot \lambda}{2 \cos(D_v/2)}\right) - \frac{D_v}{2} \quad (1.3a)$$

$$\beta = \arcsin\left(\frac{G \cdot m \cdot \lambda}{2 \cos(D_v/2)}\right) + \frac{D_v}{2} \quad (1.3b)$$

As in general $\alpha \neq \beta$, the diameter of the diffracted beam d_o will differ from the diameter of the incident beam d_i , according to

$$d_o = d_i \frac{\cos(\beta)}{\cos(\alpha)} \quad (1.4)$$

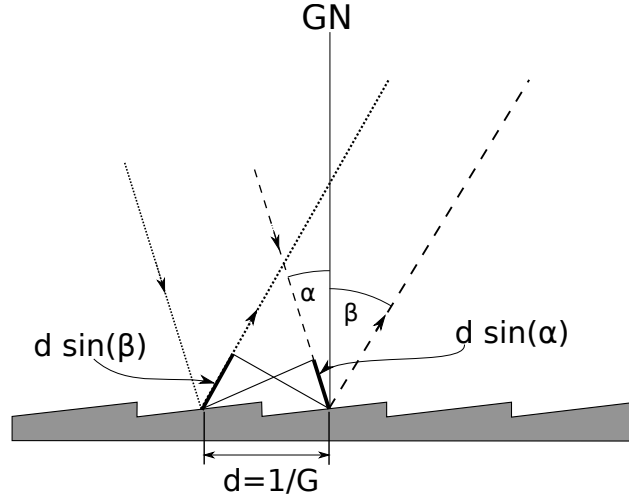


Figure 1.3: Path differences between rays incident on different grating facets.

This behaviour is called anamorphic magnification, as the alteration of the beam diameter is only happening in one plane. Imagining an axis parallel to the groove pattern of the grating, the diameter of the beam lying in the plane perpendicular to this axis is modified.

Considering that the image of the input slit at the output slit should be equal to the width of the output slit, the latter can be determined using

$$w_{\text{out}} = w_{\text{in}} \frac{\cos(\beta)}{\cos(\alpha)} \frac{f_{\text{out}}}{f_{\text{in}}}. \quad (1.5)$$

The angular dispersion D can be obtained by reordering Equ. 1.1 into an expression for β and differentiating with respect to λ :

$$D = \frac{d\beta}{d\lambda} = \frac{d}{d\lambda}(\arcsin(Gm\lambda - \sin(\alpha))) = \frac{m}{d \cos(\beta)} = \frac{\sin(\alpha) + \sin(\beta)}{\lambda \cos(\beta)} \quad (1.6)$$

Multiplying the angular dispersion D by the effective focal length r' of the system results in the linear dispersion. In systems where collimated light is diffracted by a grating and focused by a lens or mirror, r' is simply the focal length of the focusing element. If, instead, the diffracted beam is converging, r' is the distance between the focal point and the centre of the grating. Sometimes, the reciprocal linear dispersion $P = 1/(r'D)$ is stated instead, as it is convenient to have a measure of the change in wavelength in relation to a change in position along the spectrum (thus, a common choice of units of P is nm/mm):

$$P = \frac{1}{r' \cdot D} = \frac{\cos(\beta)}{G \cdot m \cdot r'} \quad (1.7)$$

With a continuous spectrum as input, the output selected with a spectrometer cannot be a single discrete wavelength, but it will be a continuous range of wavelengths, $(\lambda, \lambda + \Delta\lambda)$, as well. The length of this band of wavelengths passing through the setup is called the bandpass BP. For systems in which the width of the image of

the input slit is approximately the same as the width of the exit slit, $BP \approx w_{\text{out}} \cdot P$, which is

$$BP = \frac{w_{\text{in}} \cos(\alpha)}{G \cdot m \cdot f_{\text{in}}} \quad (1.8)$$

The gratings ability to separate two wavelengths is expressed in terms of a quantity called resolving power R :

$$R = \frac{W \cdot (\sin(\alpha) + \sin(\beta))}{\lambda}, \quad (1.9)$$

where W is the illuminated width of the grating. To separate the wavelengths λ and $\lambda + \Delta\lambda$, the resolving power has to be $R > \frac{\lambda}{\Delta\lambda}$.

1.1.2 Off-Axis parabolic mirrors

An off-axis parabolic mirror is a segment of a parabolic mirror (in fact, one method of manufacturing is to cut them out of so-called 'parent' parabolic mirrors). Their function is to focus collimated light to (or collimate light from a point source in) their focal point, while folding the optical path and thus spatially separating the focal point from the rest of the beam to allow easier access to that point.

Concerning the spectrometer which was to be built, for the purpose of collimating light from the input slit as well as focusing light to the output slit, off-axis parabolic mirrors were favoured over lenses. Like achromatic lenses they are free of spherical- and chromatic aberrations[13], but in addition to that fold the optical path and thus allow for a smaller device.

The parameters of an off-axis parabolic mirror are the following (See Fig 1.4 for reference):

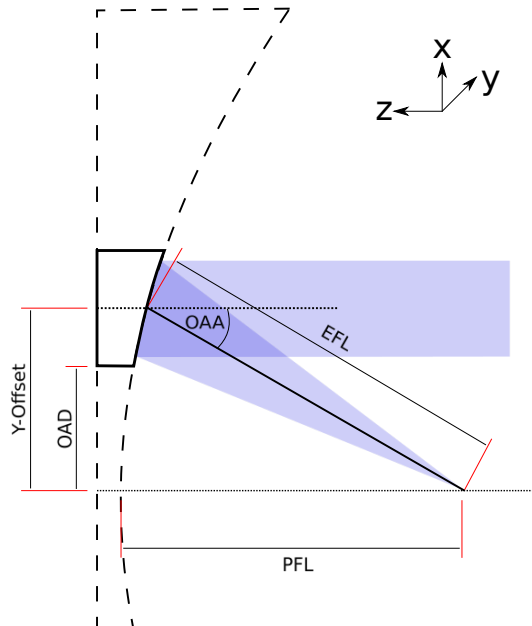


Figure 1.4: Scheme of an off-axis parabolic mirror. EFL: Effective focal length. PFL: Parental focal length. OAA: Off-axis angle. OAD: Off-axis distance.

Parental focal length (PFL) f_P : The focal length of the parent paraboloid which defines the shape of the surface through $z = r^2/(4 \cdot f_P)$, where z is the distance from the vertex on the optical axis and r is the radial distance from vertex.

Effective focal length (EFL) f_E : Sometimes also called 'slant focal length'. Distance between the mechanical centre of the mirror and the focal point.

Zonal radius X : sometimes also called 'Y-offset' (for a different choice of coordinates than in the reference figure, of course), is the distance between the optical axis of the parent paraboloid and the line parallel to it that passes through the mechanical centre of the off-axis parabolic mirror.

Off-axis distance (OAD): The distance between the optical axis of the parent paraboloid and the inner (closest) edge of the off-axis parabolic mirror. For a mirror of diameter d , $OAD = X + d/2$.

Off-axis angle (OAA) δ : The angle between the optical axis of the parent paraboloid and the line that passes through the focal point and the mechanical centre of the mirror.

These parameters satisfy the following relations:

$$Y = 2f_P \tan\left(\frac{\delta}{2}\right) \quad (1.10a)$$

$$f_P = f_E \cos^2\left(\frac{\delta}{2}\right) \quad (1.10b)$$

Consequently, out of the 5 parameters listed, only two are necessary for a complete description of an off-axis parabolic mirror of given diameter.

As long as the beam incident on the mirror is parallel to the optical axis of the parent paraboloid, the position and shape of the focal point will stay constant. Thus, purely translational errors in the relative positioning between mirror and collimated beam do not affect the focal point. Rotational misalignments, however, will cause a shifting of the focal point. Considering rotations $\Delta\eta, \Delta\phi, \Delta\xi$ of the mirror around the axes x, y, z depicted in Fig. 1.4 with origin at the mechanical centre of the mirror yields the following relations[14]:

A rotation of $\Delta\eta$ around x leads to a translation in y -direction of

$$\Delta y = 2f_P \Delta\eta. \quad (1.11)$$

Rotating by $\Delta\phi$ around the y -axis results in a translation in z -direction of

$$\Delta z = f_E \frac{\sin(2\Delta\phi)}{\sin(\delta + 2\Delta\phi)} \approx \frac{2f_E \Delta\phi}{\sin(\delta)}, \quad (1.12)$$

and a rotation of $\Delta\xi$ around z yields a translation in y -direction of

$$\Delta y \approx f_E \sin(\delta) \Delta\xi. \quad (1.13)$$

1.1.3 Rotation motor

The parameters describing how well a device like a rotation motor can achieve positioning along its degrees of freedom are called accuracy, repeatability, resolution and tilt error motion.

Repeatability refers to the ability to revert to a desired position after a movement to any other position has been executed. This quantity may be split into the unidirectional repeatability, which is measured using just one direction of rotation, and the bidirectional repeatability, which is determined taking into account both rotational directions.

Accuracy is the difference between desired and actual position. Repeatability is a necessary requirement for a proper definition of accuracy, as trying to verify a non repeatable measurement is pointless.

Resolution is the smallest difference between two positions a positioning device can achieve.

Tilt error motion is a tilt of the axis of rotation, measured through one revolution.

For the purpose of constructing a spectrometer, bidirectional repeatability is negligible, as it is always possible to scan the spectrum using only one specific direction. Unidirectional repeatability, however, is crucial, as it determines the repeatability of the measurement.

For the accuracy again the requirements are more loose as this quantity can be handled in calibration, as long as the repeatability is adequate.

The resolution should be sufficient to allow for spectral steps smaller than the bandpass of the spectrometer. Otherwise, part of the spectrum is missed.

Tilt error motion affects the vertical position of the focal point, hence its importance depends on the length of the optical path between grating and exit slit as well as on the kind of exit slit in use. A slit with a vertical extension covering the whole area in which the focal point moves will not be affected, whereas a pinhole or a fibre may be affected.

Two methods to achieve a rotation of the grating were taken into consideration: commercial rotation stages and a variant of the sine drive.

Sine drive

A sine drive consists of a linear actuator and, if required, a lever. The object which is to be rotated is fixed along the rotation axis. In a classical sine-drive, an actuator pushes against a fixed point on the object (or a lever attached to it), resulting in a rotational movement. The extension z of the actuator is directly related to the sine of the rotation angle Θ , as the name implies:

$$\sin(\Theta) = \frac{\Delta z}{d_2}, \quad (1.14)$$

where d_2 is the distance between the point at which the object is kept constant and the point at which the actuator touches the object. See Fig. 1.5 for a schematic of the apparatus and a description of the nomenclature.

If, however, the actuator is not attached to a specific point of the object but moving along a straight line, the sine drive becomes a tangent drive. In this case, the distance d_1 in Fig. 1.5 is kept constant. As $\delta = \pi/2 - \Theta$ and $\sin(\pi/2 - \Theta) = \cos(\Theta)$, the relation between Θ and the actuator extension reads

$$\tan(\Theta) = \frac{z}{d_1} \quad (1.15)$$

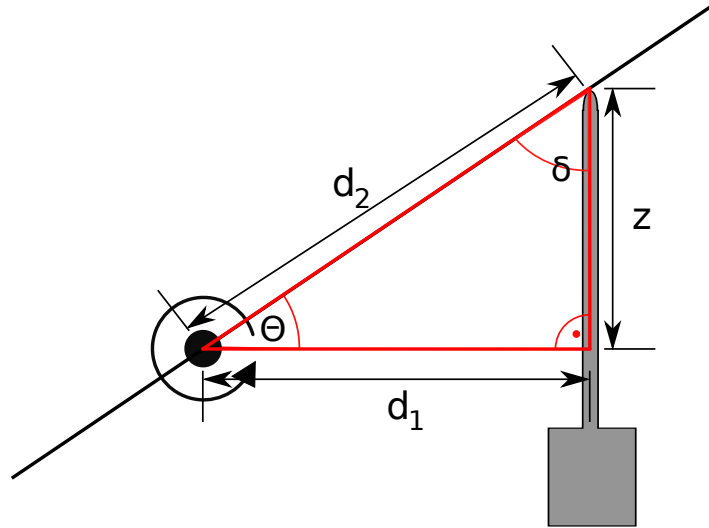


Figure 1.5: Scheme of a sine drive. z : extension of the actuator. d_1 : shortest distance between the rotation axis of the object and the axis of extension of the actuator. d_2 : shortest distance between the rotation axis of the object and the point at which the actuator touches the object. Θ : angle of rotation. In a classical sine drive d_2 is constant, whereas in the "tangent drive" variation d_1 is.

It is of course not possible to achieve full 360° of rotation this way, but if only small angle deviations are needed this option may be cheaper than a complete rotation stage.

To evaluate the usefulness of a tangent drive for the construction of the device in demand, an exemplary spectrometer operational at wavelengths between 1540 nm and 1560 nm, utilizing a grating with $G = 1200 \text{ mm}^{-1}$ and a deviation angle of 15° is considered. Under these conditions, Equ. 1.3a (or 1.3b) can be used to determine the angle of incidence (or diffraction) on (off) the grating. By doing so for both the lower and upper wavelength given in this example and subtracting the outcomes from each other, the angle which the rotation mechanism has to be capable of accessing can be acquired. In this example, it is 2.003° . If one of the piezo stacks available at Thorlabs (PK4FXP1) with quite long possible maximal extension ($57.5 \mu\text{m}$) was used, the piezo should be, at max, placed $d_{1,\text{max}} = \cot(2^\circ) \cdot 57.5 \mu\text{m} \approx 1.6 \text{ mm}$ away from the axis of rotation, which is probably quite hard to achieve.

As actuators, and piezos in particular, have a limited maximal extension, the achievable maximal rotation angle is dependent on the desired position of the piezo, d_1 . If Δd is the displacement of the piezo from the desired position, and z the extension of the piezo, then the deviation $\Delta\Theta$ from the expected rotation angle Θ is

$$\Delta\Theta = \arctan\left(\frac{d_1}{z}\right) - \arctan\left(\frac{d_1 + \Delta d}{z}\right). \quad (1.16)$$

Using the example from above, $\Delta\Theta$ is plotted in Fig. 1.6. Misplacing the piezo by 0.1 mm yields, at maximal extension of $57.5 \mu\text{m}$, an angle difference of approximately 120 mdeg.

Repeatability, accuracy and resolution do not only depend on the piezo stack itself but on the power supply and feedback mechanism, as well. Therefore it is

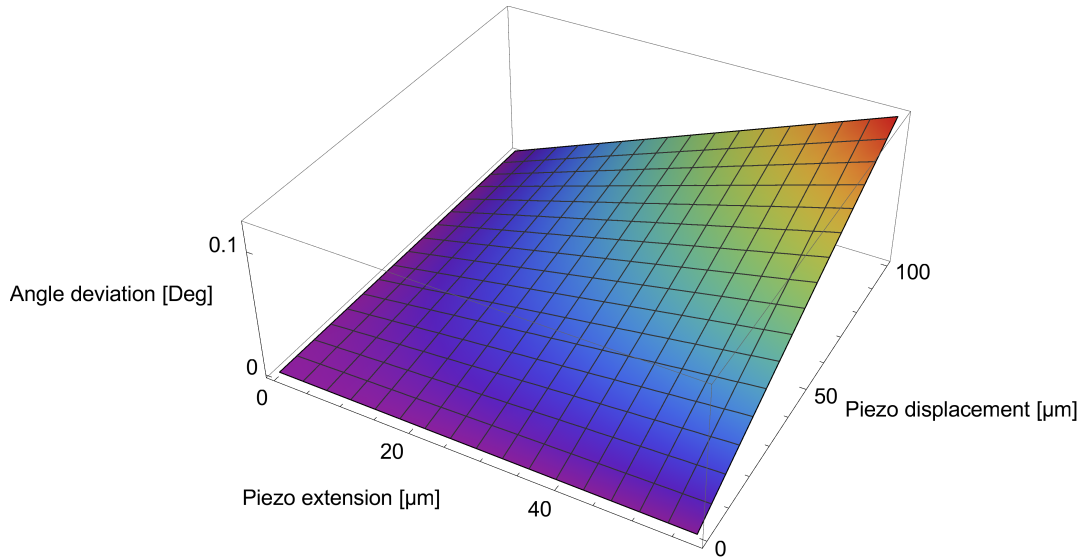


Figure 1.6: Impact of piezo displacement. Misplacing the piezo tip centre by 0.1 mm from the position results in a deviation from the expected angle of 0.12° (at an extension of $60\text{ }\mu\text{m}$).

hard to acquire actual values for bare stacks, which are preferred here because pre-built piezo actuators are significantly more expensive.

Printed Circuit Board Motor

The same example as in the last section is used: $1540\text{ nm} < \lambda < 1560\text{ nm}$, $G = 1200\text{ mm}^{-1}$ and $D_v = 15^\circ$. If in addition a bandpass of 0.1 nm is required, using Equ. 1.6 yields that the resolution of the rotation stage should be better than 0.5 mrad . One of the cheapest rotation stages found which offers this resolution and a similar repeatability is the Thorlabs NR360S. Together with an appropriate controller this stage totals at about € 3500, which is why a different solution had to be found.

As a Printed Circuit Board Motor (PCBMotor, from the company of the same name, sold for approximately € 500) was already at hand, this device was tested for its suitability to rotate the grating. The optical encoder of the tested model supports 2880 counts per revolution in full step mode, resulting in a step size of 125 mdeg or approximately 2.18 mrad ; however, it is possible to apply power to the motor in short pulses, a technique which is called micro pulsing (in the style of micro stepping a stepper motor), to lower the steps size.

The PCBMotor uses an incremental encoder, meaning that instead of exhibiting the absolute position the encoder provides a binary value out of which the controller calculates the absolute position. Furthermore, there is no 'zero point' reference encoded. This deficit was overcome by fixing a screw on the static part of the PCBMotor and a handle to the rotating part in such a way that it collides with the screw at a certain point. Therefore the ability to rotate continuously is lost, but it is not needed for this spectrometer anyway.

Figure 1.7 shows a scheme depicting the terminology of the different step sizes: A LED shines light onto the encoder wheel which is attached to the rotor. Due to the structure of this encoder wheel the light is reflected periodically (position wise).

It therefore either passes through a slit and hits a photodiode, or does not do so. Out of the photodiodes signal, the binary information is generated, depending on a threshold. The two binary values will be called 'high' and 'low', denoting the state when the current is above/below the threshold. The transition from low to high will be called 'rising edge', and 'falling edge' from high to low. The distance between two neighbouring edges is called a 'half step', and the distance between two edges of the same kind (high to high or low to low) is called a 'full step'. The commands accepted by the controller include 'go n full steps', 'apply n micro pulses' and 'go to the next rising/falling edge'.

In addition the voltage which is applied to the motor can be altered to change the step size of a micro pulse, but this also does affect the speed of rotation and the maximal rotatable mass. Another way to affect the step size of a micro pulse is to send n pulses back-to-back. Due to friction, sending n pulses consecutively does not result in the same rotation as sending 1 pulse n times with a delay in between them.

The setup used to determine the resolution of the PCBMotor is depicted in Fig. 1.8: A mount holding a fibre-collimator was attached to the PCBMotor. The collimated beam was observed using a DataRay WinCamD-UCD15 with $4.4\mu\text{m} \times 4.4\mu\text{m}$ pixel size and the relative position of its centroid on the CCD sensor was recorded. Two mirrors were used to lengthen the optical path (to 2.68 m) and thus increase the change in position when rotating the collimator.

When the PCBMotor is rotated, the step size can be determined out of the change in position on the sensor and the distance between it and the collimator. This has been done for a motor voltage of 750 mV, sending five micro pulses at a time, and for a motor voltage of 900 mV, for both sending 1 and 5 micro pulses (μP) at a time. The calculated step sizes are shown in Fig. 1.9. While the mean step size is on a promising low level for this spectrometer, the fluctuations indicate that the outcome of a sequence of many steps may not be repeatable.

To compare the variability at different step sizes, the ratio $\Delta s/\bar{s}$ between the difference $\Delta s = s_{\text{max}} - s_{\text{min}}$ between the maximal and minimal step size measured and the mean step size $\bar{s}$ was used as a measure. It is 0.390 ± 0.048 for (750 mV, 5 μP), 0.559 ± 0.079 for (900 mV, 1 μP) and 0.264 ± 0.021 for (900 mV, 5 μP), where the error

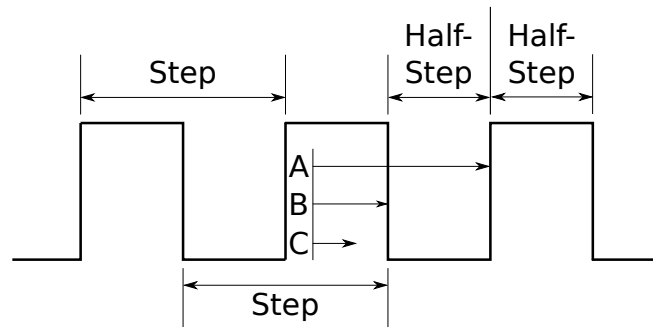


Figure 1.7: Visualisation of the binary encoder-sensor-data and nomenclature of structures within it. The distance between two rising or two falling edges is a 'full step', whereas a 'half step' is between two neighbouring edges. Built in commands of the controller include 'apply micro pulses until the next rising edge is found' (**A**) and the corresponding equivalent for 'falling edge' (**B**). (**C**) represents 'apply n micro pulses'.

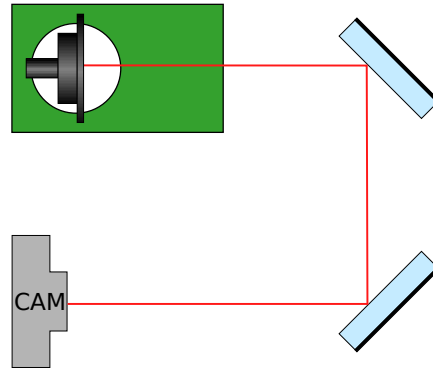


Figure 1.8: Scheme of the setup used to determine the resolution of the PCBMotor. A fibre collimator is mounted on the PCBMotor. The beam is reflected by two mirrors to increase the optical path to 2.68 m and observed by a WinCamD.

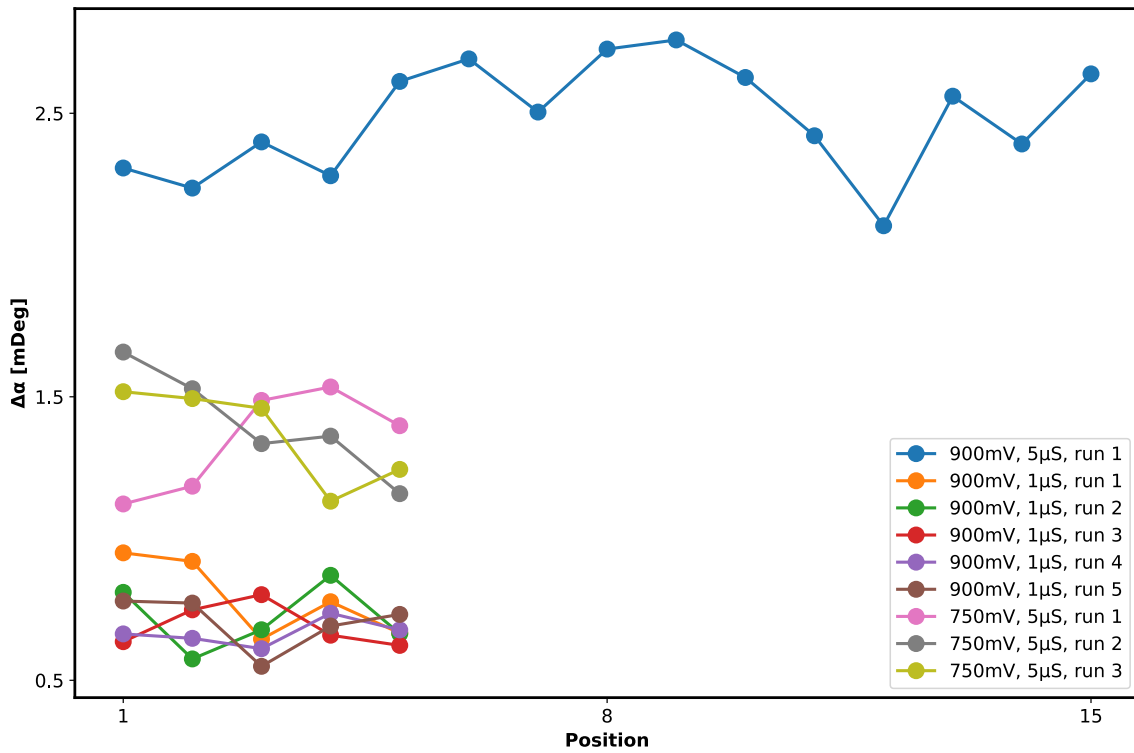


Figure 1.9: Step sizes $\Delta\alpha$ for different voltages applied to the PCBMotor and different number of consecutive micro pulses (μ P): (750 mV, 5 μ P), (900 mV, 1 μ P), (900 mV, 5 μ P). The mass of the fibre collimator setup loaded on the PCBMotor was 143.5 g.

accounts for one standard deviation of the mean value. These numbers indicate that the variability decreases with increasing voltage as well as with increasing number of micro pulses sent at a time.

The mass loaded on the PCBMotor during these measurements (fibre collimator, optical mount, post and some screws, washers and adapters) was 143.5 g. As the contact between rotor and stator of the PCBMotor is secured by a spring pressing them together, a change in mass of the payload is supposed to alter the performance of the PCBMotor. A fictional setup of a grating (Thorlabs GR50-1208, 90 g), mount (Thorlabs KM200S, 70 g) and post (Thorlabs TR50/M, 50 g) with a mass of 210 g was simulated by attaching additional posts to the mount holding the fibre collimator, resulting in a total mass of 224 g.

The motor got stuck when using voltages as low as in the last measurements, thus it was increased to 1250 mV. Sending one micro pulse at a time under these conditions resulted in a consecutive decrease of the step size, as depicted in Fig. 1.10. Thus, in the process of choosing the actual components of the spectrometer, it was set value on not exceeding the mass of the fibre collimator setup (without additional mass) used before.

As laid out in the example in Sec. 1.1.3 describing the sine drive mechanism, only a small part of the rotational freedom will be used (approximately 2° , in this example). Thus the tilt error motion was not measured through a full revolution, but only along 1.5° . To do so, a mirror was mounted on the PCBMotor and rotated in steps of 0.125° . A collimated beam was reflected by this mirror to a WinCamD placed 100 mm away from it, where the position of the beams centroid on the sensor was recorded (see Fig. 1.11). For a constant rotation axis, the measurement points are expected to lie on a straight line, approximated by a linear fit. Deviations

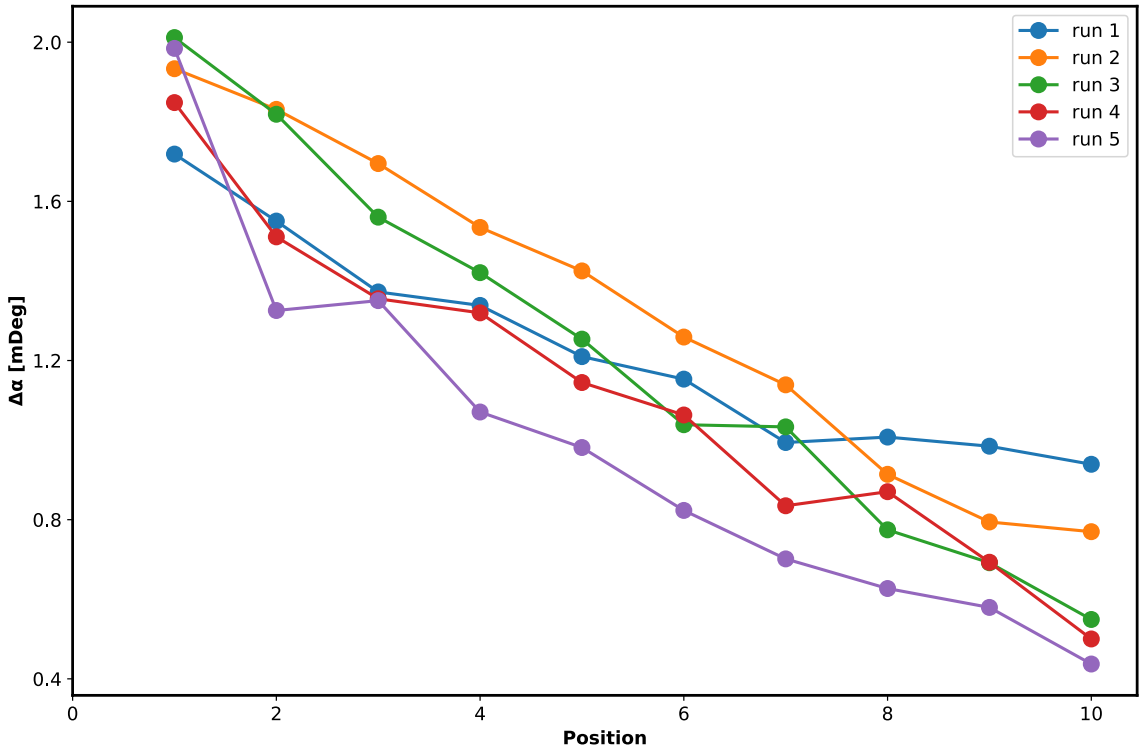


Figure 1.10: Step size $\Delta\alpha$ for a voltage of 1250 mV applied to the PCBMotor, sending 1 micro pulse at a time. The mass loaded on the PCBMotor was 224 g.

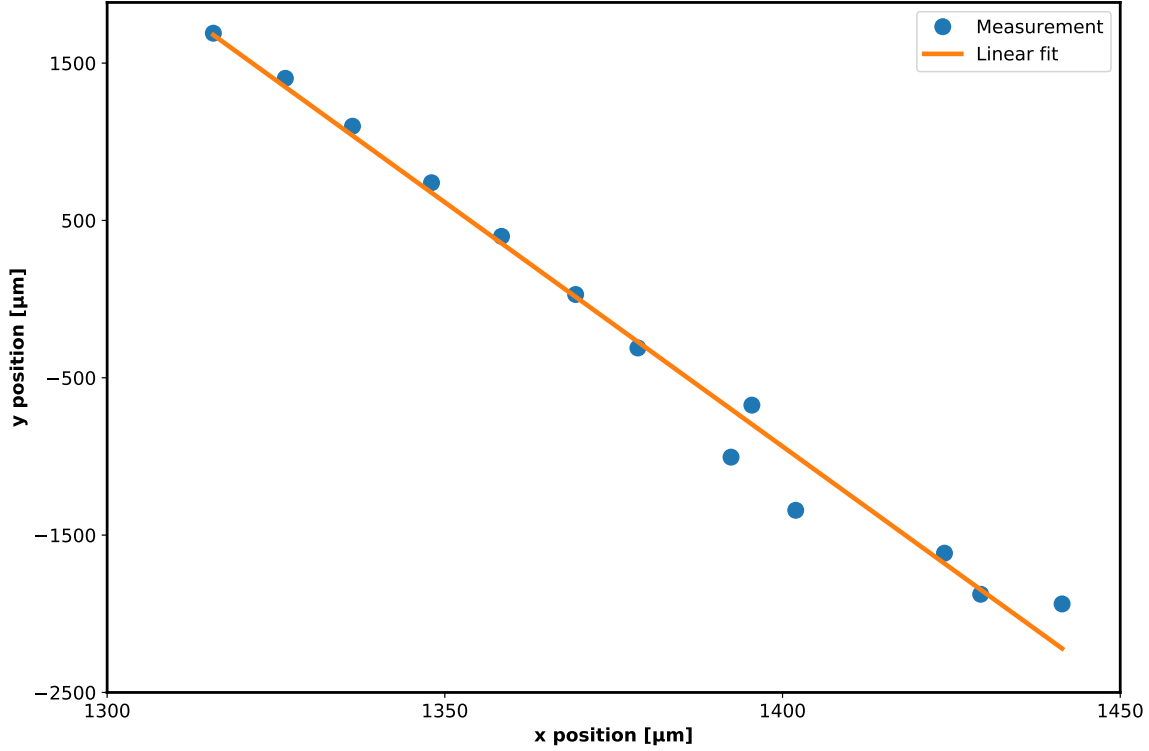


Figure 1.11: Characterisation of mechanical stability: Position of the collimated beam on the WinCamD sensor for 1.5° of rotation, in 0.125° steps. Deviations from the linear fit are treated to be caused by tilt error motion.

from this line are caused by tilt error motion. Along this rotation of 1.5° , the tilt error motion was determined to be $(440 \pm 35) \mu\text{rad}$. Using Equ. 1.11, this can be translated into a shift of the focal spot once the parental focal length is known.

1.1.4 Optical fibres

An optical fibre is a dielectric waveguide, usually of cylindrical geometry. It consists of a core of radius r_c with refractive index n_1 and a surrounding cladding with refractive index n_2 as well as several other protecting layers (buffer, jacket). Fibres can be classified into single mode (SM) fibres, which guide only a single mode, and multi mode (MM) fibres, which guide many modes. Modes are solutions of the Helmholtz equation for a given waveguide geometry.

The Helmholtz equation in a cylindrical coordinate system can be written as

$$\frac{\partial^2 \Psi}{\partial \rho^2} + \frac{1}{\rho} \frac{\partial \Psi}{\partial \rho} + \frac{1}{\rho^2} \frac{\partial^2 \Psi}{\partial \phi^2} + \frac{\partial^2 \Psi}{\partial z^2} + n^2 k^2 \Psi = 0, \quad (1.17)$$

where $\Psi = \Psi(\rho, \phi, z)$ can be any component of the electric or magnetic field. As guided modes propagate along the z -direction (along the fibre), the z -component of Ψ can be factored out: $\Psi(\rho, \phi, z) = \psi(\rho, \phi)e^{-i\beta z}$, where β is a propagation constant. For step-index fibres, in which n_1 and n_2 are constant,

$$\psi(\rho, \phi) = \begin{cases} \psi_1(\rho, \phi) & \rho \leq r_c \\ \psi_2(\rho, \phi) & \rho > r_c. \end{cases} \quad (1.18)$$

The solution can be found to be

$$\psi_1(\rho, \phi) = c_1 J_n\left(\frac{\rho u}{r_c}\right) \cdot \begin{pmatrix} \cos(n\phi) \\ \sin(n\phi) \end{pmatrix} \quad (1.19a)$$

$$\psi_2(\rho, \phi) = c_2 K_n\left(\frac{\rho v}{r_c}\right) \cdot \begin{pmatrix} \cos(n\phi) \\ \sin(n\phi) \end{pmatrix} \quad (1.19b)$$

with the core parameter $u = V\sqrt{1-B}$ and the cladding parameter $v = V\sqrt{B}$, where B is the normalized propagation constant

$$B = \frac{\frac{\beta^2}{k^2} - n_2^2}{n_1^2 - n_2^2} \quad (1.20)$$

and V is the fibre parameter (or V-number)

$$V = k \cdot r_c \sqrt{n_1^2 - n_2^2}, \quad (1.21)$$

where $k = \frac{2\pi}{\lambda}$ is the wave number. J_n is the Bessel function of the first kind and order n

$$J_n(\rho) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m! \Gamma(m+n+1)} \left(\frac{\rho}{2}\right)^{2m+n}, \quad (1.22)$$

and K_n is the modified Bessel function of the second kind and order n

$$\begin{aligned} K_n(\rho) = & \frac{1}{2} \left(\frac{\rho}{2}\right)^{-n} \sum_{m=0}^{n-1} \frac{(n-m-1)!}{m!} \left(-\frac{\rho^2}{4}\right)^m + (-1)^{n+1} \ln\left(\frac{\rho}{2}\right) I_n(\rho) + \\ & + (-1)^n \frac{1}{2} \left(\frac{\rho}{2}\right)^n \sum_{m=0}^{\infty} \left[\frac{\Gamma'(m+1)}{\Gamma(m+1)} + \frac{\Gamma'(n+m+1)}{\Gamma(n+m+1)} \right] \frac{\left(\frac{\rho^2}{4}\right)^m}{m!(n+m)!}, \end{aligned} \quad (1.23)$$

with the modified Bessel function of the first kind and order n I_n and the Gamma function Γ

$$I_n(\rho) = \left(\frac{\rho}{2}\right)^n \sum_{m=0}^{\infty} \frac{\left(\frac{\rho^2}{4}\right)^m}{m! \Gamma(n+m+1)}, \quad (1.24a)$$

$$\Gamma(x) = \int_0^{\infty} t^{x-1} \exp(-t) dt \quad (1.24b)$$

Assuming that $n_1 \approx n_2$, which is true for many fibres, it is approximately valid to require that ψ is continuous and has a continuous derivative at $\rho = r_c$

$$\psi_1(\rho, \phi) \Big|_{\rho=r_c} = \psi_2(\rho, \phi) \Big|_{\rho=r_c} \quad (1.25a)$$

$$\frac{\partial \psi_1(\rho, \phi)}{\partial \rho} \Big|_{\rho=r_c} = \frac{\partial \psi_2(\rho, \phi)}{\partial \rho} \Big|_{\rho=r_c}. \quad (1.25b)$$

Furthermore, this assumption ($n_1 \approx n_2$) implies that the direction of propagation is nearly exclusive along the fibre axis, which means that the fields are mostly transverse. The propagating mode can, therefore, be described by two orthogonal linear polarized components. These modes are thus denoted with LP_{nm} .

Using the following relations for the derivatives of J_n and K_n :

$$\frac{dJ_n(x)}{dx} = \pm J_{n\mp 1}(x) \mp n \frac{J_n(x)}{x} \quad (1.26a)$$

$$\frac{dK_n(x)}{dx} = -K_{n\mp 1}(x) \mp \frac{K_n(x)}{x} \quad (1.26b)$$

leads to the characteristic equation

$$\frac{u \cdot J_{n\pm 1}(u)}{J_n(u)} = \pm \frac{v \cdot K_{n\pm 1}(v)}{K_n(v)}, \quad (1.27a)$$

$$v = \sqrt{V^2 - u^2}. \quad (1.27b)$$

For given V and n , the parameters u , v , the propagation constant β and the field distribution $\psi(\rho, \phi)$ can be determined. In general, this equation has multiple solutions, which may be labelled with $m = 0, 1, 2, \dots, M$, as can be seen in Fig. 1.12 where the left-hand side and the right-hand side of Equ. 1.27a are plotted exemplary for $V = 9$ and $n = 0$. The number of guided modes M increases with increasing V , because $\frac{v \cdot K_{n\pm 1}(v)}{K_n(v)}$ decreases until it vanishes at $u = V$ and $\frac{u \cdot J_{n\pm 1}(u)}{J_n(u)}$ is independent of V . Thus, to find M , one needs to count the number of roots of $J_{n-1}(u)$ (looking at the minus sign in Equ. 1.27a) in the range $0 < u \leq V$. Denoting these roots with u_{nm} , the mode LP_{nm} is guided if $V > u_{nm}$.

For $n = 0$, the first roots are 0, 3.832, 7.016, 10.174, 13.324. For $n = 1$, the first roots are 2.405, 5.520, 8.654, 11.792, 14.931. Thus if $V < 2.405$, only one mode, LP_{01} , is allowed to propagate in the fibre. Under this condition, a fibre is called single moded. As $V = V(\lambda)$, the number of guided modes depends on the wavelength

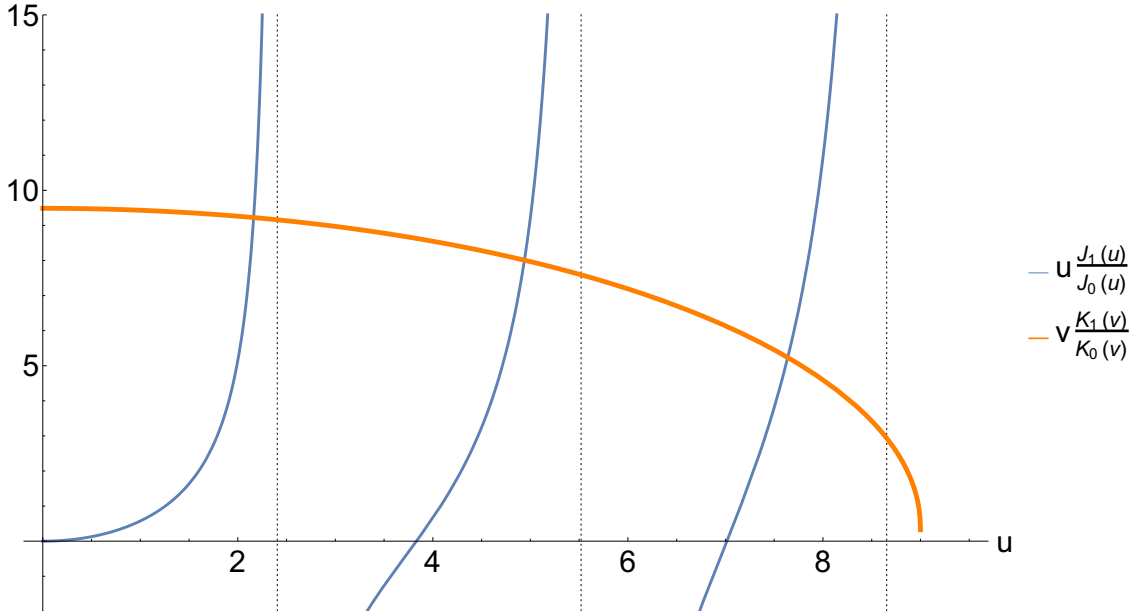


Figure 1.12: Graphical representation of the LHS (blue lines) and RHS (orange lines) of the characteristic equation 1.27a, exemplary for $n = 0$ and $V = 9$. The crossing points between the blue lines and the orange line are the solutions of the characteristic equation, the number of which equals to the number of guided modes.

as well. It is thus expedient to define a cut-off wavelength λ_c of a fibre, which specifies the turning point between single moded ($\lambda > \lambda_c$) and multi moded ($\lambda < \lambda_c$) behaviour:

$$\lambda_c = \frac{2\pi r_c \sqrt{n_1^2 - n_2^2}}{2.405}. \quad (1.28)$$

The square root term defines the numerical aperture $\text{NA} = \sqrt{n_1^2 - n_2^2}$ of the fibre.

When coupling light into a fibre, it is necessary to match the modes of the light which has to be coupled into the fibre to the modes propagating in the fibre. For single mode fibres, an estimation of the coupling efficiency can be obtained by an overlap integral between the field amplitudes of incident light ψ_i and fibre mode ψ_F [15]

$$\eta_c = \frac{|\int \int \psi_i(x, y) \psi_F^*(x, y) dx dy|^2}{\int \int |\psi_i(x, y)|^2 dx dy \int \int |\psi_F(x, y)|^2 dx dy} \quad (1.29)$$

where the LP_{01} mode of a single mode fibre can be approximated by a gaussian

$$\psi_F^{\text{SM}} = \sqrt{\frac{2}{\pi}} \frac{1}{w_{\text{MFD}}} \exp\left(-\frac{x^2 + y^2}{w_{\text{MFD}}^2}\right), \quad (1.30)$$

with the mode field diameter w_{MFD} of the optical fibre, which is the distance between two points in the radial field distribution at which the field dropped to $1/e$ of its maximum. Considering an elliptical gaussian distribution for the light which is meant to be coupled into the fibre, the field amplitude at the focus is

$$\psi_i^{\text{ell}} = \sqrt{\frac{2}{\pi w_x w_y}} \exp\left(-\frac{x^2}{w_x^2}\right) \exp\left(-\frac{y^2}{w_y^2}\right) \quad (1.31)$$

where w_x, w_y are the beam waists in the respective direction.

Putting Eqs. (1.29)–(1.31) together, the maximal efficiency of coupling an elliptical gaussian beam with beam waists w_x and w_y into a single mode fibre with mode field diameter w_{MFD} , at the absence of aberrations and misalignment, is

$$\eta_c^{\text{ell,SM}} = \frac{16w_{\text{MFD}}^2 w_x w_y}{(w_{\text{MFD}}^2 + 4w_x^2)(w_{\text{MFD}}^2 + 4w_y^2)}. \quad (1.32)$$

1.2 Different types of spectrometers

Commonly, the dispersive element and the auxiliary optics that together form a spectrometer and their specific angle configuration are called *mount*. Thus, a particular scheme of a grating based spectrometer setup is a grating mount. Various different grating mounts have been described and built, some of which will be presented in the following section.

1.2.1 Czerny-Turner mount

This is the most popular grating mount in use [16]. It is made up of three optical elements: a plane grating and two concave spherical mirrors. A scheme of this mount is pictured in Fig. 1.13. Spectrometers using these components were built before,

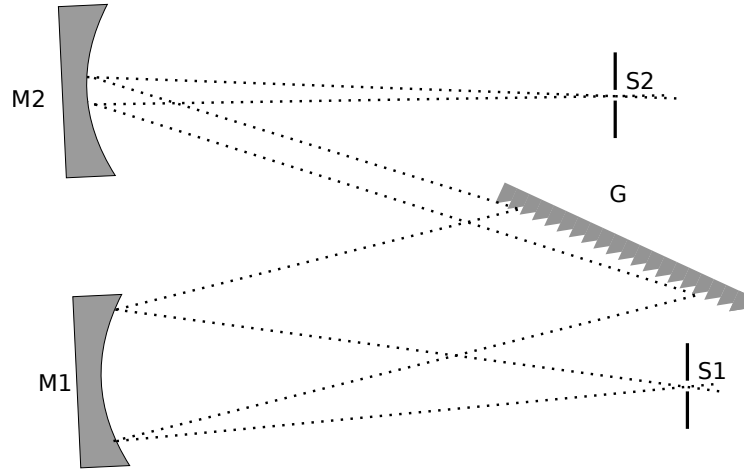


Figure 1.13: Mount: Czerny-Turner Type. S1: Entrance slit. M1: Collimating mirror. G: Grating. M2: Focusing mirror. S2: Exit slit.

but Czerny and Turner published a reminder about the astigmatism behaviour of this type of spectrometer depending on the placement of the mirrors [17]. This mount does not exhibit coma at one wavelength for a specific geometry[16].

1.2.2 Littrow mount

The speciality of this mount [18] is that the deviation angle is set to zero. Thus, incident and reflected beam follow the same path and only one lens/mirror is needed, serving both as collimating and at the same time as focusing element. Usually, input and exit slit are separated by locating them slightly below and above the central plane. See Fig. 1.14 for a scheme of this mount.

A popular application of this mount is to use it as a so called laser tuner in wavelength tunable mid-IR lasers[19].

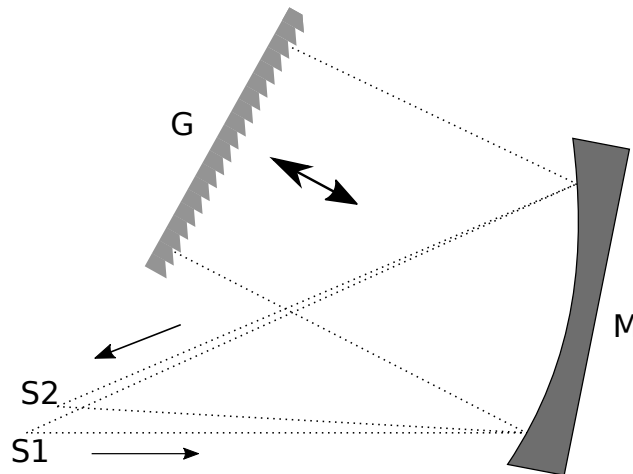


Figure 1.14: Mount: Littrow Type. S1: Entrance slit. M: Mirror or lens used for both collimation and focusing. G: grating. S2: Exit slit. The angle between light incident on and reflected off the grating is 0° . Thus, usually the slits are separated in height.

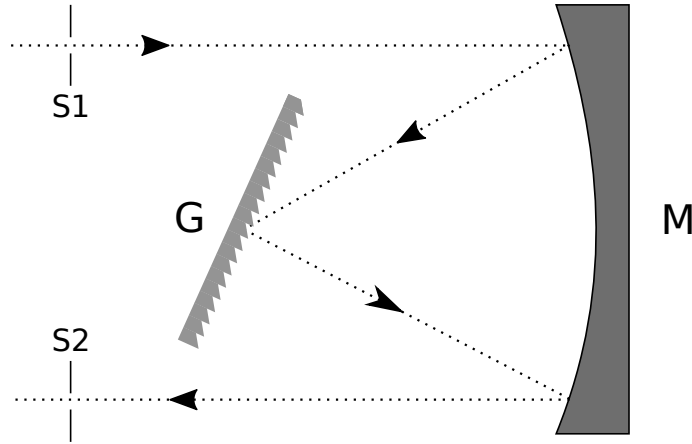


Figure 1.15: Mount: Ebert Type. S1: Entrance slit. G: Grating. M: Mirror used for collimation and focusing. S2: Exit slit.

1.2.3 Ebert-Fastie mount

In using only one big mirror, this setup can be seen as a special case of the Czerny-Turner configuration in which the centres of curvature of collimating and focusing concave mirror coincide and their radii are the same[20]. Relative misalignment of the mirrors thus is prevented by design, but the missing degrees of freedom make it difficult to control aberrations. This issue can be overcome by replacing the straight entrance- and exit slits by curved slits[21, 22]. A scheme of this mount can be found in Fig. 1.15.

1.2.4 Monk-Gillieson mount

This configuration is special in that it does not illuminate the grating with collimated light but rather with a converging beam[23, 24]. In this way, the need for a second mirror or lens is eliminated (see Fig. 1.16 for reference). Light passing through

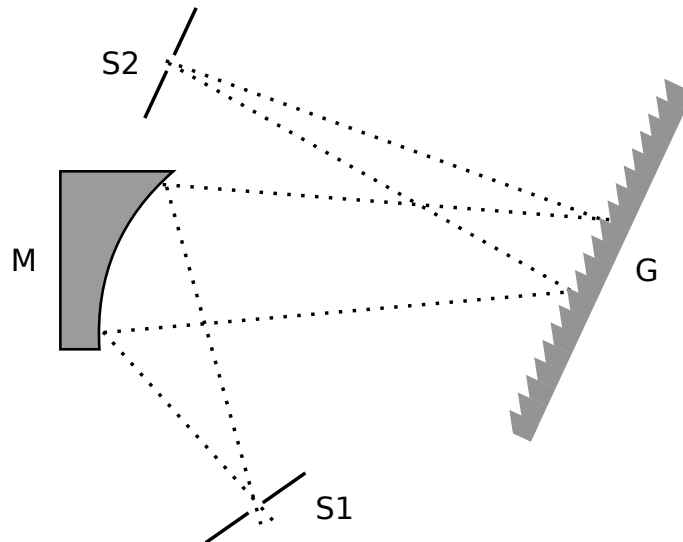


Figure 1.16: Mount: Monk-Gillieson Type. S1: Entrance slit. M: Concave mirror. G: Grating. S2: Exit slit.

a slit is made converging by a concave mirror, refracted by a grating and brought into focus at the exit slit. Optimal focus is only achievable at one wavelength and coma is severe (which is true for all plane grating mounts using non-collimated light incident on the grating)[25]. Thus the possible fields of application for this mount are limited[26].

1.2.5 Rowland mount

Gratings need not be plane of course. Concave gratings exhibit two additional degrees of freedom, the radii of curvature in two planes, which can be used to focus light and thus offer the possibility to omit one or even both mirrors. Therefore this feature is especially beneficial for applications where mirror reflectivities are comparatively low, which is the case for example in the EUV region, in particular for broadband mirrors[27]. In addition, this parameters allow for a partial correction of higher order aberrations[28].

One grating mount utilizing such a grating is the Rowland mount[29], whose scheme is depicted in Fig. 1.17. The geometrical structure of interest in this mount is the rowland circle, a circle passing through the grating centre, of diameter equal to the tangential radius of curvature of the grating. Light originating from point sources located on the rowland circle are, after dispersion and diffraction by the grating, brought into focus on the rowland circle again, with position depending on the wavelength.

Mounts utilizing concave gratings are usually, but not necessarily, used as spectrographs.

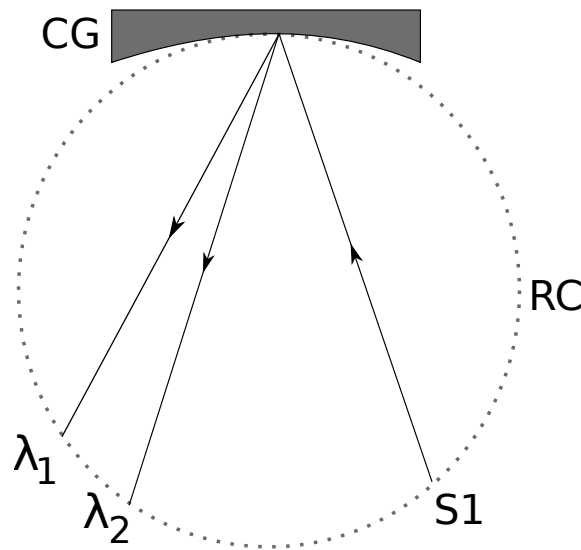


Figure 1.17: Mount: Rowland Type. If a point source is positioned on the rowland circle, the refracted beam will be on focus on the rowland circle again. S1: Entrance slit. CG: Concave grating. RC: Rowland circle.

Chapter 2

Determination of required components

Given that concave diffraction gratings exhibit additional degrees of freedom in production, it is harder to find the right component cheap and in stock. In addition, just rotating a plane grating is technically easier than rotating a slit and simultaneously moving it along a circle. Thus it was decided to use a plane grating.

Using the Monk-Gillieson mount was ruled out because of its strong aberrations. The necessity of curved slits implied the same for the Ebert-Fastie mount. A Czerny-Turner mount was favoured over a Littrow mount because of its greater construction simplicity, as the plane the beam path lies in is parallel to the optical table.

For the sake of convenience and detector independence, fibres were used as input- and output slits.

The first step in the process of determining a set of components to use is to fix the wavelength range in which the spectrometer is supposed to operate. Inspecting Equ. 1.1:

$$G \cdot m \cdot \lambda = \sin(\alpha) + \sin(\beta)$$

and taking into account that the modulus of the sine is definitely smaller than 1, $\sin(\alpha) + \sin(\beta) < 2$, and therefore $G < 2/(m\lambda)$. To prevent overlapping of different diffraction orders, the groove density has to be chosen accordingly. As an example, the case that only the first diffraction order shall be observable is examined. In that case, for the first order to propagate, $G < 2/\lambda$. But to avoid the propagation of the second order, $G > 1/\lambda$. Thus, $1/\lambda < G < 2/\lambda$. For a certain wavelength range $(\lambda_{\min}, \lambda_{\max})$, this translates into

$$1/\lambda_{\min} < G < 2/\lambda_{\max}. \quad (2.1)$$

The minimal required wavelength range of the spectrometer was fixed to be $(1.54 \mu\text{m}, 1.56 \mu\text{m})$, and as most commercially available diffraction gratings are optimized to diffract into the first order, m was set to 1. Therefore, the groove density of the grating should be between 649 mm^{-1} and 1282 mm^{-1} .

Now, the deviation angle D_v is set to a specific value. Starting from $D_v = 0^\circ$, increasing it shortens the optical path, more specifically the path between the grating and both mirrors (see Fig. 2.1), thereby allowing alignment to be simpler, stability to be easier to achieve and the final device to be (up to a certain extent) smaller. On the other hand, increasing D_v usually lowers the diffraction efficiency [12]. A

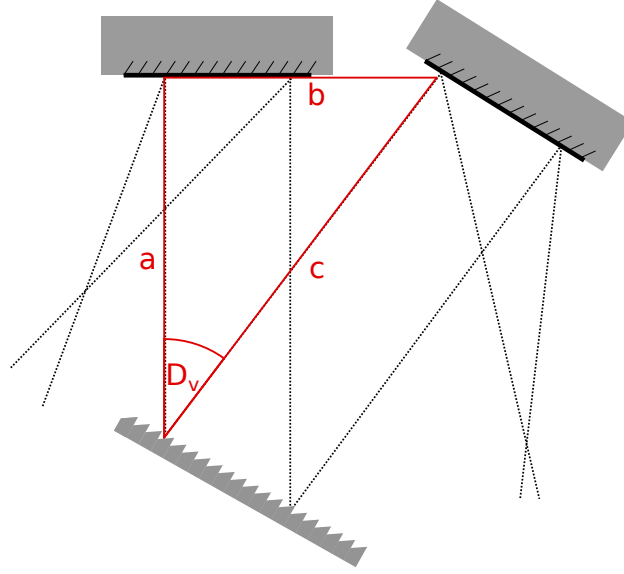


Figure 2.1: Exemplary scheme of a spectrometer setup showing the impact of the numerical value of D_v on the geometry. The distance b is given by the beam-waist, the extension of the optical mount of the mirror and, usually, by the unilluminated part of the mirror. For decreasing deviation angle, the distance between grating and mirrors has to increase.

compromise has to be found. Once D_v is set, ranges for α and β can be determined using Eqs. (1.3a) and (1.3b).

The broadest bandpass ('lowest resolution') acceptable for the the final device is fixed next. Using Equ. 1.8 and the range of α , which was just determined, an upper bound for the ratio between the mode field diameter of the input fibre (which is equivalent to the slit width) w_{in} and the focal length of the collimating mirror f_{in} , $w_{\text{in}}/f_{\text{in}}$, is acquired. Out of all available fibres and off-axis parabolic mirrors, a proper combination has to be found. If elements of such kind are available, one might calculate the diameter of the (diverging, after passing through the fibre) input beam at the collimating mirror:

$$d_{\text{CM}} = 2 \cdot f_{\text{in}} \cdot \tan(\arcsin((NA)_{\text{fb}})) \quad (2.2)$$

where $(NA)_{\text{fb}}$ is the numerical aperture of the fibre. This has, of course, to be compared to the clear aperture of the mirror. Over illumination of the mirror causes a drop in efficiency as well as the possibility of stray light entering the output fibre as additional noise. The possibilities to circumvent over illumination include choosing a bigger mirror, using auxiliary optics to reduce the beam divergence at the input or choosing a different fibre with lower numerical aperture.

Once the diameter of the beam incident on the grating is known, it is possible (and necessary) to verify that the resolving power (Equ. 1.9) of the grating is big enough to separate two wavelengths λ and $\lambda + \Delta\lambda$. The resolving power that has to be exceeded is usually expressed as $\lambda/(\Delta\lambda)$, where λ should be set to the the upper bound of the range at which the spectrometer is expected to be functional and $\Delta\lambda$ is the difference in wavelength one wants to be able to resolve. Thus,

$$R = \frac{W \cdot (\sin(\alpha) + \sin(\beta))}{\lambda} > \frac{\lambda_{\text{max}}}{\Delta\lambda}$$

where W is the illuminated ruled width of the grating, which is the projection of the diameter of the collimated beam onto the grating surface, $d_{\text{CM}}/\cos(\alpha)$. If R does not exceed $\lambda/\Delta\lambda$, one has to find a more appropriate 'input-section' (use a different fibre with higher numerical aperture or replace the collimating mirror with one which has a bigger focal length, which may require a change in diameter of the mirror as well). Also, at this point it has to be verified that, given the diameter of the collimated beam, the angle of its incidence on the grating α , and the physical dimensions of the chosen grating, that the grating is not over-illuminated (again, because of efficiency and stray light).

For $\alpha \neq \beta$, the diameter of the beam refracted from the grating undergoes anamorphic magnification, which means it is modified according to Equ. 1.4. Thus, for the determination of the focal length of the focusing mirror and the mode field diameter of the output fibre, it is usually not as simple as to use the same values as input fibre and collimating mirror exhibit. The ratio $w_{\text{out}}/f_{\text{out}}$ between the mode field diameter of the output fibre w_{out} and the focal length of the focusing mirror f_{out} is given by a transformation of Equ. 1.5. Out of all available combinations, an appropriate one has to be chosen. The alteration of the diameter of the refracted beam also has to be considered when determining the diameter of the focusing mirror.

It has to be mentioned, however, that this modification only happens in one plane, namely the plane perpendicular to the axis the grating is rotated about (hence it is named "anamorphic"). Therefore, if the intensity profile of the beam incident on the grating is circular gaussian, the refracted beam will, usually, exhibit an elliptic gaussian profile. At this point, one might want to check if the theoretically achievable coupling efficiency into the output fibre is acceptable (using Equ. 1.32 if the output slit is a single mode fibre).

Now that a set of components adding up to a setup within the desired specifications was found, the requirements on the rotation stage can be determined. Multiplying the angular dispersion, Equ. 1.6, by the bandpass, Equ. 1.8, gives the lowest resolution the rotation stage has to exhibit to be able to achieve angle differences which do not result in wavelength differences bigger than the bandpass.

As outlined in the last paragraphs, manually putting together a set of standard components which yield the desired result is a somewhat iterative process of choosing certain values, calculating properties and checking if they fit together properly. Therefore, a list of off-the-shelf single mode fibres, gratings and off-axis parabolic mirrors was compiled. A spreadsheet was constructed in which the necessary quantities described in this section were calculated for every possible combination of those components for a certain number of D_v : from 10° to 17.5° in steps of 2.5° , to find a compromise between optical path length (decreasing with increasing D_v , up to a certain extent) and efficiency (also decreasing with increasing D_v). All combinations which did not fulfil the requirements (sufficiently low bandpass, no over illuminated components, enough lines illuminated to reach the theoretical value for the bandpass, with angular dispersion big enough to be able to address each bandpass with at least one position) were discarded, as well as those being inappropriate regarding the examinations which were explained in this section.

Out of the remaining compilations, one was chosen manually. This last selection process consisted of choosing a configuration with relaxed requirements on the resolution of the rotation motor (with comparatively high product of angular dispersion

and bandpass) on the one hand, and on the other hand picking a configuration which (taking into account the anamorphic magnification of the beam) produces a focal spot size at the output fibre which closely matches the mode field diameter of one of the available fibres.

Chapter 3

Experimental Setup

The setup was mounted on a $250 \times 300 \times 13 \text{ mm}^3$ breadboard to ease the composition and maintain the ability to transfer the device into an enclosure after carrying out the alignment and initial tests.

As input, a Thorlabs SMF-28-J9 fibre patch cable of 1 m length and with a mode field diameter of $(10.4 \pm 0.5) \mu\text{m}$ was mounted in a 25 mm lens tube connected to a Thorlabs K6XS 6-axis kinematic mount. Initially, a 1 m Thorlabs M43L multi mode patch cable with a core diameter of $105 \mu\text{m}$ together with a Thorlabs P10S $10 \mu\text{m}$ pinhole right in front of it were mounted in a Thorlabs K6X to function as the exit slit. Later on, MM-fibre and pinhole were replaced with a 1 m SMF-28-J9 patch cable as well.

The off-axis-parabolic mirrors for collimation (Standa 14OAP-1-50.8-30-AL aluminium coated, 30° off-axis angle, 50.8 mm parental focal length, $(54.45 \pm 0.54) \text{ mm}$ effective focal length, 25.4 mm diameter) and focusing (Standa 14OAP-1-12.7-90-AL aluminium coated, 90° off-axis angle, 12.7 mm parental focal length, $(25.40 \pm 0.25) \text{ mm}$ effective focal length, 25.4 mm diameter) were glued into mirror-holders appropriate for Thorlabs KMS/M compact kinematic mounts.

A lens tube was glued onto the rotor of the PCBMotor and a custom made mount for the grating (see Fig. 3.1) was fixed onto a lens-tube end-cap with central bore using a screw and a nut. The mount was designed with a reduction of mass to a feasible value in mind, which included omitting any kinematic degrees of freedom. Using the CAD-model, the estimated mass is 9.5 g, in contrast to the 70 g of an off-the-shelf kinematic mount for rectangular optics (Thorlabs KM200S). Initially, the grating was attached to it by 4 screws. Pieces of rubber band were placed between the sides of the grating and the inner faces of the mount at the positions of the bores for the screws fixing the grating to ensure that the optics is not damaged by the screws.

The alignment was done with a fibre-coupled laser operating at 1549 nm (Optilab DFB-4-B-20) using a laser viewing card. A paddle fibre polarisation controller was placed in front of the setup, as the efficiency of the grating is sensitive to polarisation changes.

The compilation of the components was started with the placement of the OAPs, followed by the mounts holding the fibres. An iteration loop of collimating the beam and hitting the supposed grating centre, simulated with a laser viewing card, was carried out for both entrance and exit fibre/mirror section.

At first, the collimation of the OAPs was done by observing beam diameter and shape at different distances from the mirrors. But, as a consequence of the large

diameters of the beams, the rayleigh length $z_R = \pi \cdot w_0^2 / \lambda$ (where w_0 is half of the collimated beam diameter, Equ. 2.2) was quite large (approximately 120 m for the entrance-beam) which made this procedure inexpedient. Therefore, a shearing-interferometer was used which forced the usage of another wavelength (775 nm was chosen), as the diffuser-plate of this device was not suitable for 1550 nm. Still, the collimation using off-axis parabolic mirrors was found to be very intricate and a textbook example of perfect collimation could not be achieved.

Next, the PCBMotor with the attached grating was inserted. A multi mode fibre was inserted as exit and the power coupled into it was observed using a power meter (Thorlabs PM100D). To assure that after the focusing mirror the beam hits the face of the fibre at an angle as small as possible, the fibre was moved back and forth between two points using the freedom of the kinematic mount, and the coupling was optimized iteratively until the fibre could be moved along this axis without affecting the coupling ratio at these points.

After coupling nearly 100% of the light into the fibre, the pinhole was placed directly in front of it. Unfortunately, despite extensive efforts, no signal could be observed at all. This can probably be explained by manufacturing tolerances, namely the mispositioning between the core of the fibre and the centre of the pinhole, as is depicted in Fig. 3.2. Hence, the combination of multi mode fibre and pinhole was replaced with a SMF-28-J9 single mode fibre.

For a fixed rotational state of the grating, the coupling was optimized. Then, the grating was rotated using a few micro pulses and the coupling was optimized again. This procedure was repeated until a maximum was found.

The rotational degree of freedom offered by the 6-axis kinematic mounts was soon found to be quite unstable, leading to a reduction in throughput by approximately 25% in 30 minutes and a complete loss of coupling in one to two days. As this freedom was not necessary, the relevant part of the mounts were glued using two-

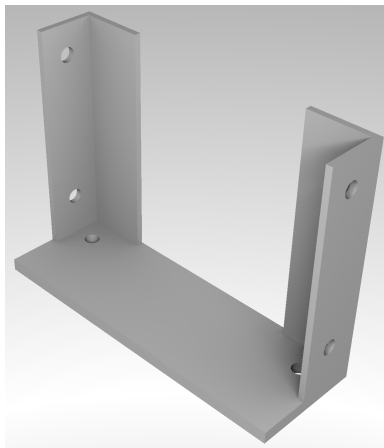


Figure 3.1: Rendered image of the grating mount.

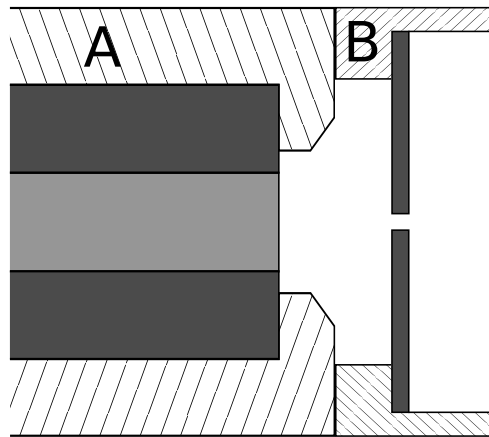


Figure 3.2: The first attempt of creating a fibre coupled exit slit: In a lens tube, a pinhole (with housing **B**) is placed right next to a multi mode-fibre (attached to a free-space adapter **A**). The relative positioning between the hole in the pinhole and the core of the fibre is crucial for the success of this concept.

part epoxy adhesive.

Using this procedure and setting the polarisation of the light entering the spectrometer in such a way that the observed power after the exit-fibre was maximal, an overall efficiency of the spectrometer of $\eta_{\text{tot}} = 0.3$ could be achieved at 1549 nm ($P_{\text{in}} = 7.03$ mW of power leaving the input fibre and $P_{\text{out}} = 2.13$ mW observed after the exit-fibre).

At this polarisation, and still using 1549 nm, the reflectivity of the OAPs was determined to be $R_{\text{OAP}} = 0.89$. Using the very same settings, the diffraction efficiency of the grating into the first order was measured to be $\eta_{\text{G}} = 0.85$. Thus the coupling efficiency into the exit-fibre was $\eta_c = \frac{P_{\text{out}}}{P_{\text{in}} R_{\text{OAP}}^2 \eta_{\text{G}}} = 0.45$. According to the theoretical estimation of the coupling efficiency obtained by using the procedure described in Sec. 1.1.4, a higher value was expected. Figure 3.3 depicts the theoretical coupling efficiency for this setup, which is wavelength-dependent because of a change in anamorphic magnification depending on the grating-rotation angle (see Fig. 3.4). At 1549 nm, this estimation predicts a coupling efficiency of 0.77. This gap may be explained by the non-perfect collimation and by aberrations introduced by the grating and the off-axis parabolic mirrors (due to imperfect alignment).

Finally, the setup was glued completely. First, both degrees of freedom on the KMS/M mounts holding the OAPs were glued. After the glue cured for one day, the coupling was optimized with the 6-axis mounts, and another one of their degrees of freedom was removed. Each day, another recoupling was carried out and the next

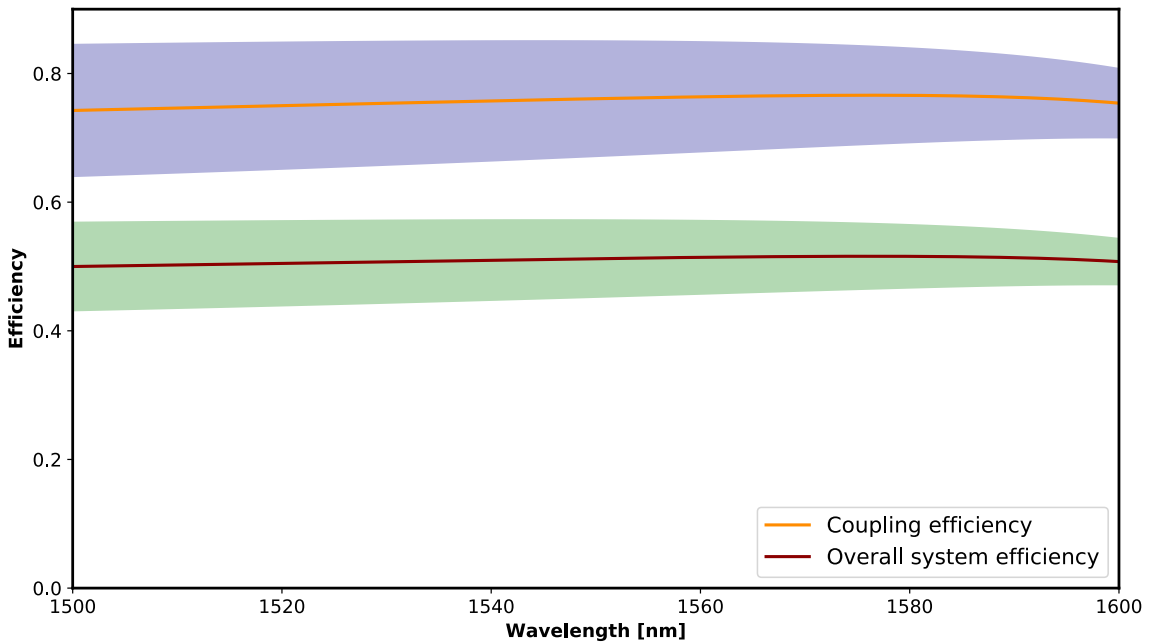


Figure 3.3: Using the overlap-integral theory and taking into account only the rotation- and thus wavelength-dependent anamorphic magnification of the beam gives the ideal maximal coupling efficiency (orange curve). Taking the reflectivities of the mirrors and the diffraction efficiency of the grating into the first order into account yields the estimated maximal overall system efficiency (red curve). The bands surrounding these curves take into account the uncertainty of the mode field diameter of both single-mode fibres and the uncertainty in the focal length of both mirrors.

degree of freedom was removed. During this process the overall efficiency dropped from 0.3 to about 0.27. On the last day of this procedure, the grating was glued into the mount as well. One day later the overall efficiency was determined to be 0.2. As this drop was not observed immediately after applying the glue but only after final curing, the most probable explanation for this behaviour is that the grating was tilted because of unbalanced shrinking of the adhesive and inhomogeneous application of the glue. In the following 4 weeks, the efficiency continuously decreased to 0, indicating that one of the degrees of freedom was still drifting.

Because of the inability to recouple, using apparently glued mounts, the mount of the output fibre was dismantled completely and this part of the spectrometer was rebuilt using another mount. This way, an efficiency of 0.35 was reached. This mount was left unglued to retain the ability to recouple if needed. Although the new mount is of the same type as the one it replaced, it is more stable. In the following 4 weeks, the efficiency only dropped to 0.2. However, owing to the unglued screws, it could be brought back to 0.32 by spending only a few minutes on recoupling.

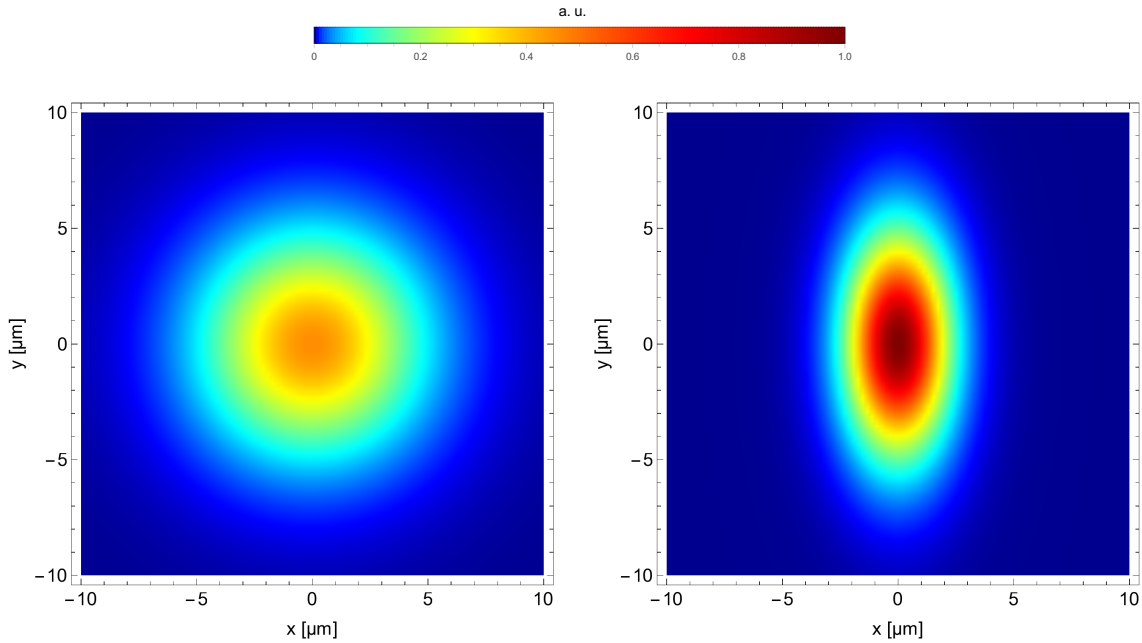


Figure 3.4: Left: Intensity profile of a gaussian beam with a waist of $10.4/2 \mu\text{m}$ at its waist, modelling the mode of a single mode fibre with mode field diameter $10.4 \mu\text{m}$. Right: Intensity profile of an elliptical gaussian beam at its waist (calculated with the actual values of components of the real setup), modelling the beam magnified through the grating and the mirrors.

Chapter 4

Software

To automatize the measurement process, a software-tool had to be developed whose main tasks are: communicating with the PCBMotor to rotate the grating; acquiring data from different detectors; and processing and displaying the data. The implementation was done using C++ and the Qt-framework¹, allowing for usage across several platforms (the tool was tested on GNU/Linux and Microsoft Windows). The software was given the name "dinspect", short for the detector independent spectrometer control tool.

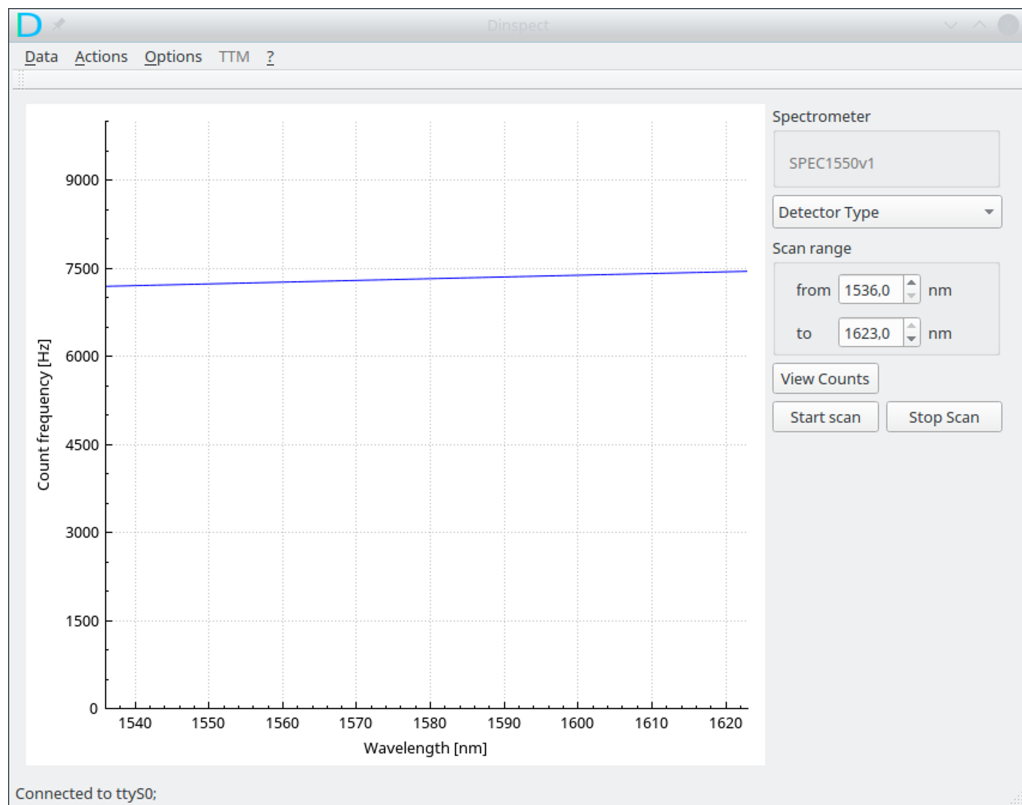


Figure 4.1: Screenshot of the main window of the control-software.

¹<https://www.qt.io/>. Version 5.6.1 was used for development.

4.1 Motor control

Communication with the controller of the PCBMotor is realized by a serial connection established using a USB-to-RS232 converter. This controller offers a few bytes of unused memory - two of which were used to store a unique ID. After initializing the connection, using this ID the spectrometer in use can be identified and an appropriate configuration file holding device-specific data like the grating groove density can be loaded (an example of such a configuration file will be given in appendix A to serve as a starting point for the integration of new spectrometers). This way, the software can be used for different PCBMotor-based spectrometers without altering the code.

4.2 Supported detectors

4.2.1 AIT TTM8000

The TTM8000 is an 8-channel time-to-digital converter for up to 25 MEvents/s with 82.3ps resolution in its continuous-measurement mode[30]. As it can be combined with any (single-photon) detector which provides a digital signal its use renders the spectrometer essentially detector-independent. The time tag data is transferred to a PC for post processing over an Ethernet connection using the User Data Protocol (UDP). Since the UDP-packages can be relayed to any other PC in the same Local Area Network (LAN), it is not necessary to connect the TTM8000 and the spectrometer to the same PC as long as a UDP/IP connection can be established between both PCs.

Parts of the very extensive sample application source code (licensed under the GNU GPLv2²) called *TTMViewer* shipped together with the device were used to incorporate the data acquisition and event frequency determination into the spectrometer control tool. For measurements with low count rates, the timespan during which data for each data point is acquired can be chosen according to one's needs.

In addition, it is possible to measure the spectral coincidences between two channels of the TTM8000, where an arbitrary coincidence window width can be chosen as well as a timing-offset between the two channels.

4.2.2 Thorlabs PM100D

The Thorlabs PM100D is a handheld optical power meter capable of operating with various photodiode sensors covering a wavelength range from 200 nm to 5500 nm and a power range from 100 pW up to 500 mW. Although it is not capable of single-photon detection, the afore-mentioned features and its relatively moderate price render it an expedient extension to the TTM8000 for cases in which single-photon sensitivity is not needed.

The power meter has to be connected to a USB-port of the same PC that controls the PCBMotor. The communication is realized by interfacing to Python and using the Python package ThorlabsPM100³, as this package already incorporates

²Available at <https://www.gnu.org/licenses/gpl-2.0.txt>.

³Developed by Pierre Cladé. Available at <http://pythonhosted.org/ThorlabsPM100/>. Distributed under the [BSD-License](#).

all necessary functionality. As communication protocol, it uses either the *Universal Serial Bus Test and Measurement Class* (USBTMC, the default for GNU/Linux) or the *Virtual Instrument Software Architecture* (VISA, the default for Windows). When a measurement is to be carried out, a Python interpreter is initialized, which executes a script that returns the current power reading of the power meter. As the data acquisition sporadically fails and returns either 0 or a negative error-code, the obtained value is checked for positivity and the measurement is redone in case the check fails.

4.3 The measurement procedure

To simplify matters, the nomenclature of wavelengths and angles, corresponding to specific rotational states of the grating, referred to in the following paragraphs is visualized in Fig. 4.2.

Each measurement starts with turning the PCBMotor to the mechanical stop position (fixed using the screw-handle mechanism described in Sec. 1.1.3, denoted with α_s in Fig. 4.2). As this point is far away from the position corresponding to about 1540 nm, the next step is to get to a reference position close to this point. This is done by first rotating by a specific number of steps, with 100 μ s waiting

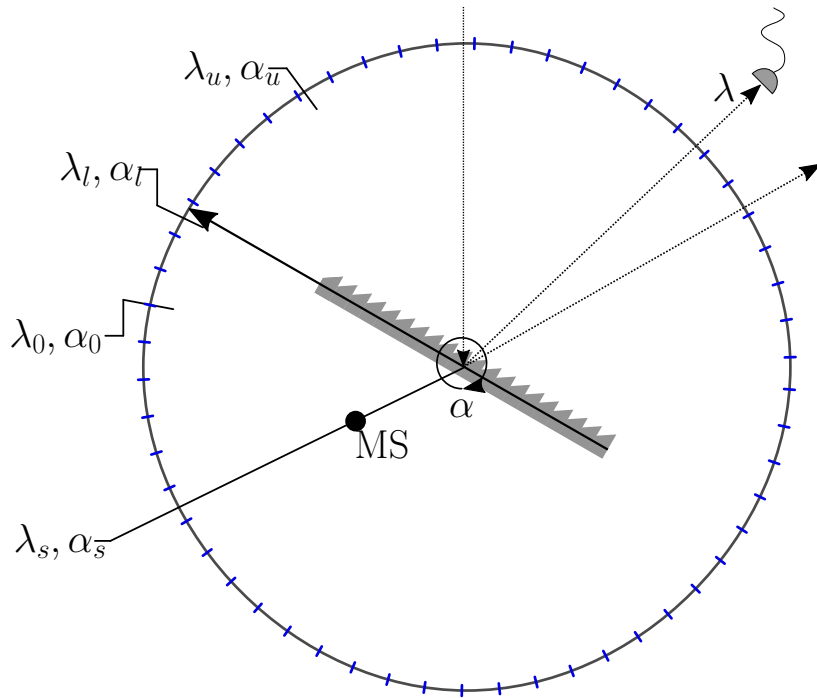


Figure 4.2: Schematic representation of certain rotational states of the grating. The blue strips on the circle represent the edges in the encoder-sensor-signal of the PCBMotor. λ_x is the selected wavelength at a certain rotation-angle α_x , where x is: **s** at the mechanical stop (MS), **0** at the reference starting point, **l/u** at the lower/upper limit of the wavelength range for a specific measurement. α_0 has to coincide with an edge to know λ_0 for sure. If there are n edges between α_0 and α_l , (n-1) of them can be skipped fast. One has to be skipped slowly for accurate positioning. The actual measurement starts at the last edge for which $\alpha < \alpha_l$ and ends at the first edge for which $\alpha > \alpha_u$.

time in between each two steps, for going all steps at once (> 500 for this specific spectrometer) was found to be unreliable since sometimes the motor overshoots its target by one step. The resulting position is still not achievable in a repeatable manner, hence the next edge in the encoder-sensor signal is found by applying micro pulses and observing the sensor data for a change. Again, this is implemented with waiting between applying the micro pulses as the controller built-in function meant to do this does not offer the desired repeatability. The state of rotation of the grating after this procedure is described by some angle α_0 , corresponding to a selected wavelength λ_0 .

Depending on the lower limit λ_l of the wavelength range $[\lambda_l, \lambda_u]$ selected for the currently intended measurement, it is possible to position the grating even closer to this desired starting point. As the detection of an edge in the signal of the encoder-sensor is necessary to determine the currently selected wavelength, this can only be done in steps of $1/16^\circ$ (the size of a half-step). Thus starting from λ_0 , the wavelengths λ_n corresponding to angles $\alpha_0 + n \cdot (1/16)^\circ, n \in \mathcal{N}$ are calculated until the first wavelength λ_k is found for which $\lambda_k > \lambda_l$. As λ_l shall be included in the measurement, the starting position will be set to an angle corresponding to λ_{k-1} . The 'skipping' is done using the built-in controller commands "rotate until the next edge is detected". As outlined before, this command does not deliver repeatable results. Therefore the last edge has to be found by using a sequence of micro pulses with waiting time in between. Therefore this procedure only saves time when $k \geq 3$ and is thus only carried out in those cases.

The last edge for which $\lambda < \lambda_l$ corresponds to a wavelength λ_{start} . There the actual measurement begins. A number of micro pulses specified in the configuration file is applied to the motor to achieve a rotation which will here be referred to as "measurement step". Afterwards, the acquisition of data is carried out, depending on the selected type of detector. The signal of the encoder sensor is observed at each of these measurement steps. At each edge the wavelengths for all positions occupied since the last edge are calculated, assuming the measurement steps are of constant size. If the wavelength at the current edge is greater than λ_u the measurement is stopped.

Due to the uncertainty of the step size, the data is initially recorded in the form of 'frequency at each step' ($f_{\text{step},i}$). The frequencies are only related to a wavelength (yielding f_{wl,λ_i}) at each edge, where the number of steps since the last edge is known as well as the angle between two edges, which is assumed to be exactly $(1/16)^\circ$. In addition, the current wavelength interval to be measured is divided into non uniform subintervals sized one bandpass: $(\lambda_{\text{start}}, \lambda_{\text{stop}}) = \bigcup_{\lambda_k=\lambda_{\text{start}}}^{\lambda_k<\lambda_{\text{stop}}} (\lambda_k, \lambda_k + \text{BP}(\lambda_k))$ with $\lambda_k + \text{BP}(\lambda_k) = \lambda_{k+1}$. For each subinterval, all f_{wl,λ_i} corresponding to it are averaged to a single value $f_{\text{BP},j} = \frac{1}{n_j} \sum_{\lambda_i \in (\lambda_j, \lambda_{j+1})} f_{\text{wl},\lambda_i}$, where n_j is the number of measurement points in a subinterval. These values $f_{\text{BP},j}$ are displayed in a graph and may later on be stored.

If an edge corresponding to a wavelength λ_{stop} is detected for which $\lambda_{\text{stop}} > \lambda_u$, the measurement stops.

4.4 Calibration

The quantities that need to be calibrated are the deviation angle D_v and the offset angle $\gamma = \alpha_0 - \alpha_x$, which is the angle between the position the grating is in when positioned such that the angle of light incident on the grating $\alpha = 0$ and the position of the reference starting point where each measurement begins (α_0 in Fig. 4.2). For the determination of D_v , Equ. 1.3a which relates the angle of light incident on the grating α with D_v and the wavelength of the light source λ (assuming that the groove density G is known), is used. By taking a measurement of a light source of known wavelength λ , a relation between λ and an angle of rotation of the motor ξ is found. This angle can be expressed as $\xi = \alpha + \varsigma$, where ς is the unknown angle between α_0 and the position of the motor where $\alpha = 0$. As ς is constant for a specific value of α_0 (the starting angle of a measurement), it can be eliminated by carrying out two measurements at wavelengths λ_1, λ_2 , which yield the angles ξ_1, ξ_2 . The difference between these angles is $\Delta\xi = \xi_1 - \xi_2 = \alpha_1 - \alpha_2 + \varsigma - \varsigma = \alpha_1 - \alpha_2 = \Delta\alpha$. Using Equ. 1.3a and w.l.o.g. examining the case $m = 1$, $\Delta\alpha$ can be expressed as

$$\Delta\alpha = \alpha_1 - \alpha_2 = \arcsin\left(\frac{G\lambda_1}{2\cos\left(\frac{D_v}{2}\right)}\right) - \arcsin\left(\frac{G\lambda_2}{2\cos\left(\frac{D_v}{2}\right)}\right). \quad (4.1)$$

Taking the sine of both sides of this equation and using $\sin(\arcsin(A) - \arcsin(B)) = A\sqrt{1-B^2} - B\sqrt{1-A^2}$, the following expression for D_v can be found:

$$D_v = 2 \operatorname{arcsec}\left(\frac{-2\sin(\Delta\alpha)}{G\sqrt{\lambda_1^2 + \lambda_2^2 - 2\lambda_1\lambda_2\cos(\Delta\alpha)}}\right) \quad (4.2)$$

This way, the deviation angle can be determined by carrying out two calibration measurements.

To determine γ , it is related to the angle of incidence α_3 corresponding to a known wavelength λ_3 through $\alpha_3 = \gamma + \chi$. As γ is the offset between $\alpha = 0$ and the reference starting position α_0 , the angle of rotation between the starting position and the occurrence of the central wavelength of a measurement χ can be acquired. Using Equ. 1.3a again,

$$\gamma = \alpha_3 - \chi = \arcsin\left(\frac{G \cdot \lambda_3}{2\cos\left(\frac{D_v}{2}\right)}\right) - \frac{D_v}{2} - \chi. \quad (4.3)$$

Eventually, the spectrometer can be calibrated by taking 3 measurements of light sources of known wavelength.

The calibration is carried out using dinspect. The three measurements can be taken right after each other, where the numerical value of the wavelength of the connected light source has to be entered before each single measurement starts. Another possibility is to carry out single calibration measurements and save the results. Once three measurements are saved, the calibration can be done using these files.

In case the light source used for calibration can be controlled using a text file, this process can be automated. Dinspect can write custom commands, specified in the spectrometer configuration file, to the text file and monitor it for changes. After placing the command for a wavelength-change, it waits for the content of the file to indicate that the desired wavelength was set and starts a measurement.

4.5 Additional Features

4.5.1 Automatisations

For cases in which the spectrum of a light source has to be acquired multiple times, a way to automatise the measurements may be beneficial. Some parts of *dinspects* functionality are controllable via customisable commands received via an UDP connection. That way it is possible to start and stop measurements, decide which detector (and, in case of using the TTM8000, which channels) is/are to be used and the wavelength range of the measurement.

Whenever a measurement is finished, UDP packages containing a custom message may be sent to inform other devices or applications about the progress, so that they may change the parameters of the light source and in turn instruct *dinspects* to start a new measurement.

Every measurement can be saved automatically on finish, named with a time tag.

4.5.2 Rough wavelength selection

As the efficiency of the grating (and thus of the spectrometer) is highly polarisation dependent, one might want to optimize the polarisation before starting the actual measurement. But to do so, the spectrometer needs to be brought into a state that selects a wavelength close to the central wavelength of the light source to be measured. The direct way to do so is to start a measurement, observe the output power and stop the measurement when a peak in the output power is observed. But due to the inconstant step size of the PCBMotor and the thus required careful (and slow) positioning procedure described in Sec. 4.3, this process can be quite time-consuming.

For this and similar cases in which the selection of a wavelength close to a specific wavelength is desired, but no exact knowledge of the selected wavelength is needed, an option to select such a state has been implemented. The corresponding positioning procedure skips the waiting time between consecutive steps and uses the faster but less precise motor control driver implementations of 'find the next edge on the encoder wheel'.

For additional fine tuning of the position, the most commonly used motor commands ('apply n micro pulses', 'go n steps', 'go to the mechanical stop' and 'find the next edge') can be accessed from within a dialogue. Any other command can be sent from there using the appropriate ASCII commands.

Chapter 5

Characterisation

5.1 Angular positioning uncertainty

As the positioning uncertainty of the PCBMotor depends on the loaded mass which could only be estimated at the time of the examination of the PCBMotor in Sec. 1.1.3, the actual setup has to be revised with this in mind. To do so, the motor was rotated and the data of the encoder wheel sensor was recorded. A total of 231 edges were detected, where the mean number of measurement steps between them was 41.4, with a standard deviation of 2.6. This can be translated into a measurement-step size of $(26.3 \pm 1.8) \mu\text{rad}$. Thus, the angular uncertainty is $74 \mu\text{rad}$ at most (after 41 steps, right before the next edge where the position is known again). As the resulting wavelength uncertainty is wavelength dependent (increasing with λ), a worst case scenario was picked with $\lambda = 1560 \text{ nm}$. Once the deviation angle of the actual setup is known (see the next section 5.2), the wavelength uncertainty at this point, introduced by the PCBMotor, can be determined to be 0.013 nm .

5.2 Calibration

The reference used for calibration was a locked infrared laser whose wavelength was measured using a wavelength meter (Toptica WS7-30). Calibration measurements were taken at 1548 nm , 1552 nm and 1556 nm .

The calibration yielded a deviation angle of $D_v = 13.31^\circ$ and an offset angle of $\gamma = 61.45^\circ$.

Verification measurements were performed at 1548 nm (result: $1547.92^{+0.11}_{-0.013} \text{ nm}$), 1550 nm (result: $1549.96^{+0.11}_{-0.013} \text{ nm}$) and 1552 nm (result: $1551.98^{+0.11}_{-0.013} \text{ nm}$). The errors stated account for the bandpass (see the next section 5.3) and the angular positioning uncertainty.

5.3 Bandpass

The bandpass of the spectrometer was determined by measuring the output of a tunable continuous wave (cw) laser (Keysight N7711A) at different wavelengths. The data was fitted to gaussian distributions and their FWHMs were calculated ($\text{FWHM} = 2\sqrt{2\ln 2}\sigma$ for gaussian distributions with a standard deviation σ). The FWHM is the same as the bandpass as the linewidth of this laser is much smaller than

the bandpass of the spectrometer. An average was calculated out of the different results.

The recorded data can be seen in Fig. 5.2, which also shows the FWHM for each single measurement (ranging from 0.094 nm to 0.108 nm). The averaged bandpass is $(0.101\,35 \pm 0.000\,15)$ nm, which is higher than the expected bandpass, whose wavelength dependence is pictured in Fig. 5.1.

This deviation can be explained by the fact that the bandpass equation 1.8 does only account for the sizes of the slits of the spectrometer and the beam magnification due to the mirrors and the grating, but not for a finite spectral bandwidth of the input light and aberrations. A more complete description of the bandpass is [31]

$$\text{BP}^{\text{real}} = \sqrt{\text{BP}^2 + \Delta\lambda_i^2 + \Delta R^2}, \quad (5.1)$$

where BP is the known ideal bandpass, $\Delta\lambda_i$ is the spectral bandwidth of the input light and ΔR is the limiting resolution of the spectrometer, determined by aberrations and diffraction effects.

Apart from ΔR , all quantities are known (the spectral bandwidth of the laser is 2 MHz, corresponding to 16 fm at 1550 nm). With the ideal bandpass at the same wavelength, (0.0728 ± 0.0036) nm and the measured bandpass as stated before, $\Delta R = (0.0705 \pm 0.0037)$ nm. Part of this quantity may be introduced by the off-axis parabolic mirrors as the alignment procedure of these components was complex, as described in Sec. 3.

5.4 Wavelength range and wavelength accuracy

Using the same tunable laser, wavelengths as low as 1539.8 nm could be measured (see Fig. 5.3, in which the wavelength the laser was set to is plotted against the

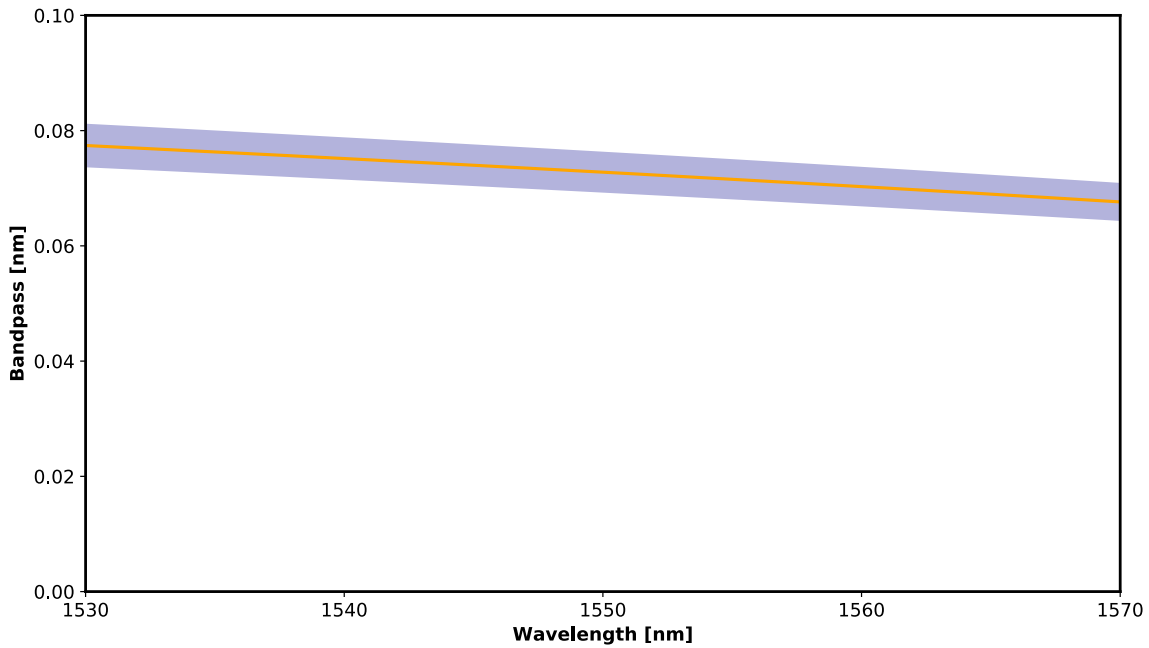


Figure 5.1: The expected bandpass for the actual setup, calculated using Equ. 1.8. The band accounts for the uncertainties in the focal lengths of the mirror and the uncertainty in the mode field diameter of the single-mode fibre.

wavelength measured with the spectrometer). Two things can be extracted from these data: For one, the spectrometer is, in principle, operational between 1540 nm to 1560 nm. However, as the efficiency of the spectrometer is wavelength dependent (not only because of the wavelength-dependent coupling efficiency and the grating diffraction efficiency, but also because of possible alignment issues), a more practical value may be obtained by finding the points in the efficiency-curve of the spectrometer at which the efficiency drops to a certain value.

To estimate the efficiency behaviour, the measurements using the Keysight N7711A laser to determine the bandpass can be analysed, assuming that the polarisation of the laser does not change when the wavelength is altered. Fig. 5.2 contains a gaussian fit of the maximal power values at the different wavelengths. The maximal efficiency was observed at (1549.34 ± 0.16) nm and the e^{-2} width is (10.26 ± 0.58) nm.

In addition, assuming that the tunable laser in use is exact (disregarding a constant offset between it and the laser used for calibration), the accuracy of the spectrometer can be obtained by determining the maximal deviation between the set and measured wavelengths. This was done by calculating the residuals of a linear fit to the data and determining the biggest one, which is 0.054 nm.

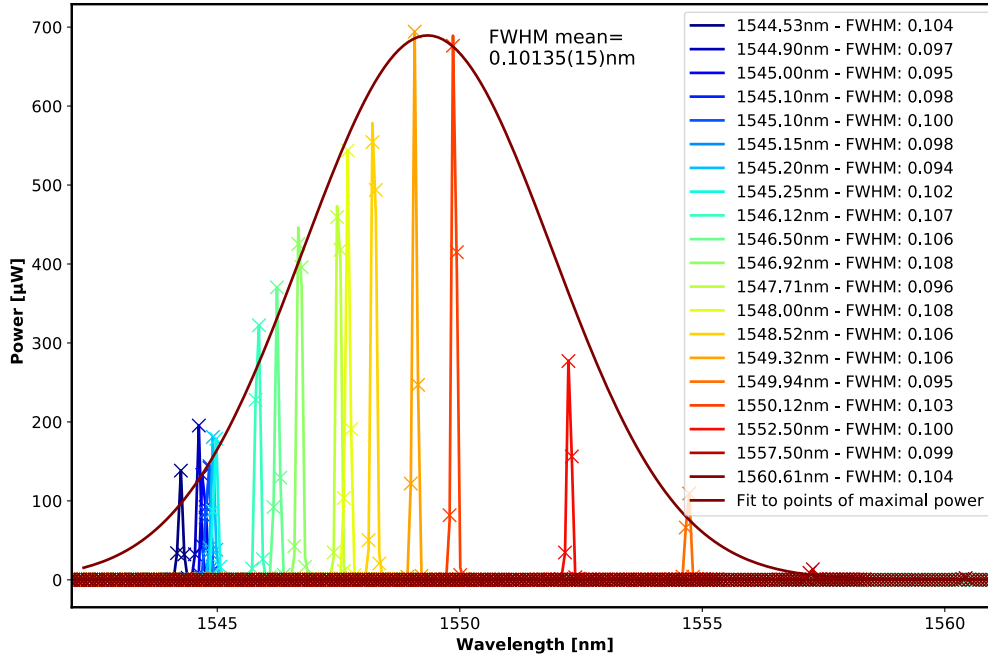


Figure 5.2: Points: data of 20 measurements recorded to determine the bandpass of the spectrometer. Solid curves: gaussian fits to the data. The measurements were done between 1544.5 nm and 1560.6 nm with a tunable cw laser (Keysight N7711A). The FWHMs of the single measurements range from 0.094 nm to 0.108 nm with an average of $(0.101\,35 \pm 0.000\,15)$ nm. Out of the fit to the points of maximal power for each measurement (darkest red), the wavelength range of the spectrometer can be determined.

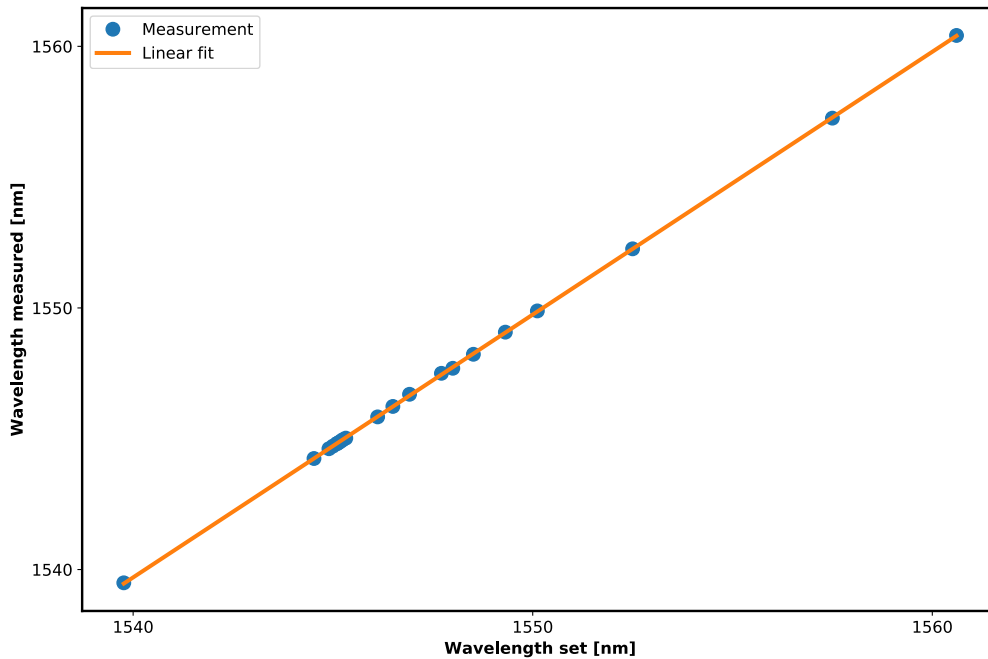


Figure 5.3: Wavelength measured by the spectrometer versus wavelength set with the tunable laser (Keysight N7711A), showing that the spectrometer is capable of recording data at least between 1540 nm and 1560 nm. The biggest residual of the linear fit can be considered to be the wavelength accuracy of the spectrometer, which is 0.054 nm.

Chapter 6

Applications

6.1 Spectral properties of a quasi phase matched collinear Type-II SPDC source in PPKTP around 1550nm

6.1.1 Background

The relation between the polarisation density $\mathbf{P}$ and the electrical field $\mathbf{E}$ in non linear crystals can be expressed as a power series expansion: using Einstein summation convention,

$$P_i(t) = \chi_{ij}^{(1)} E^j(t) + 2\chi_{ijk}^{(2)} E^j(t) E^k(t) + 4\chi_{ijkl}^{(3)} E^j(t) E^k(t) E^l(t) + \dots \quad (6.1)$$

where ϵ_0 is the vacuum permittivity and $\chi^{(n)}$ - tensors of rank $(n+1)$ - are the n -th order susceptibilities of the medium. The indices i, j, k, l refer to the cartesian coordinates. $\chi^{(n)}$ also contains information about the symmetries of the crystal, and the observed effective non-linearity will in general depend on the propagation direction of the light in the crystal.

Common methods to create entangled photon pairs are based on spontaneous parametric down-conversion (SPDC), a second-order non-linear optical process in which a pump photon with wavelength λ_p is converted into two photons - signal and idler - with wavelengths λ_s and λ_i . For this process to happen, energy- and momentum conservation have to be fulfilled:

$$\frac{1}{\lambda_p} = \frac{1}{\lambda_s} + \frac{1}{\lambda_i} \quad (6.2a)$$

$$\frac{n_p}{\lambda_p} = \frac{n_s}{\lambda_s} + \frac{n_i}{\lambda_i} \quad (6.2b)$$

where n_x is the refractive index of the material applicable for the respective photon.

In general, n_x depends on the temperature of the crystal, the wavelength of the photon, its polarisation and the angle between the optical axis of the crystal and the wave vector. Satisfying the second condition is called phase-matching, as a phase mismatch $\Delta k = (2\pi/c) \cdot (n_s/\lambda_s + n_i/\lambda_i - n_p/\lambda_p)$ will cause destructive interference between photons created in different parts of the crystal. This destructive interference will start after one coherence length $l_c = \pi/|\Delta k|$.

There are several ways of preventing this problem, one of them being the exploitation of the birefringence of the material and choosing polarisations and propagation directions such that $\Delta k = 0$. This method is called "birefringent phase-matching".

By choosing the propagation direction suitable to achieve phase-matching, it is, in general, not possible to simultaneously select the biggest component of the susceptibility tensor.

A possibility to overcome this limitation is called "quasi phase matching", which uses materials in which the non-linear properties of the medium are spatially modulated. One way of achieving this modulation is by periodic poling, a process in which the domain orientation of the non-linear crystal is reversed with a periodicity of $\Lambda = L/2$, which is called the poling period.

For periodically poled crystals, the phase matching conditions now read

$$\frac{n_p(\lambda_p, T)}{\lambda_p} = \frac{n_s(\lambda_s, T)}{\lambda_s} + \frac{n_i(\lambda_i, T)}{\lambda_i} + \frac{m}{\Lambda(T)} \quad (6.3a)$$

$$\frac{1}{\lambda_p} = \frac{1}{\lambda_s} + \frac{1}{\lambda_i} \quad (6.3b)$$

with the integer-valued quasi phase matching order m . Due to the temperature dependence of the refractive indices [32, 33] and the thermal expansion [34] of KTP (the material of the crystal used in the experiment at hand) it is, to some extent, possible to adjust signal and idler wavelengths for a fixed pump wavelength and a specific crystal.

An exemplary solution of these equations is, for a specific case, pictured in Fig. 6.1: a pump wavelength of 775 nm, a quasi phase matching order of (-1) and a poling period of 46.2 μm . Plots of this kind will be called *quasi phase matching temperature tuning curve*, or short *phase matching curve*. In this particular case at a certain temperature, the crossing point temperature, the wavelengths of signal and idler are degenerate.

The crossing point does shift when the pump wavelength is changed, as can be seen in Fig. 6.2, where phase matching curves for 5 different pump wavelengths between 771 nm and 779 nm, with the same poling period, are pictured.

The phase matching curve depends strongly on the poling period of the crystal, which can be seen in Fig. 6.3, where the difference between signal and idler wavelength is plotted against temperature for 5 poling periods in a range close to the value of the crystal at use in the experiment at hand. Changing Λ by 50 nm, which is a relative change of just above 10^{-3} , results in a change of the crossing point temperature of approximately 15 $^\circ\text{C}$ (for $\lambda_p = 775$ nm).

Usually, the probability of creating a two photon state with SPDC is rather small (in KTP, the largest entry of the second order susceptibility tensor is $\chi_{33}^{(2)} = 18.5 \text{ pV m}^{-1}$ [35]). Thus, the state describing signal and idler fields is mostly the vacuum state $|0\rangle$, but it also contains even-numbered photon contributions. The two-photon contribution can be expressed as

$$|\Psi\rangle = N \chi_{\text{eff}}^{(2)} L \int d\omega_s d\omega_i f(\omega_s, \omega_i) a_s^\dagger(\omega_s) a_i^\dagger(\omega_i) |0\rangle, \quad (6.4)$$

where the subscripts s, i stand for signal and idler, ω_x are the angular frequencies and $a_x^\dagger$ are the respective creation operators, the crystal length L , the accessible non-linearity of the crystal $\chi_{\text{eff}}^{(2)}$ and a proper normalisation constant N . The two

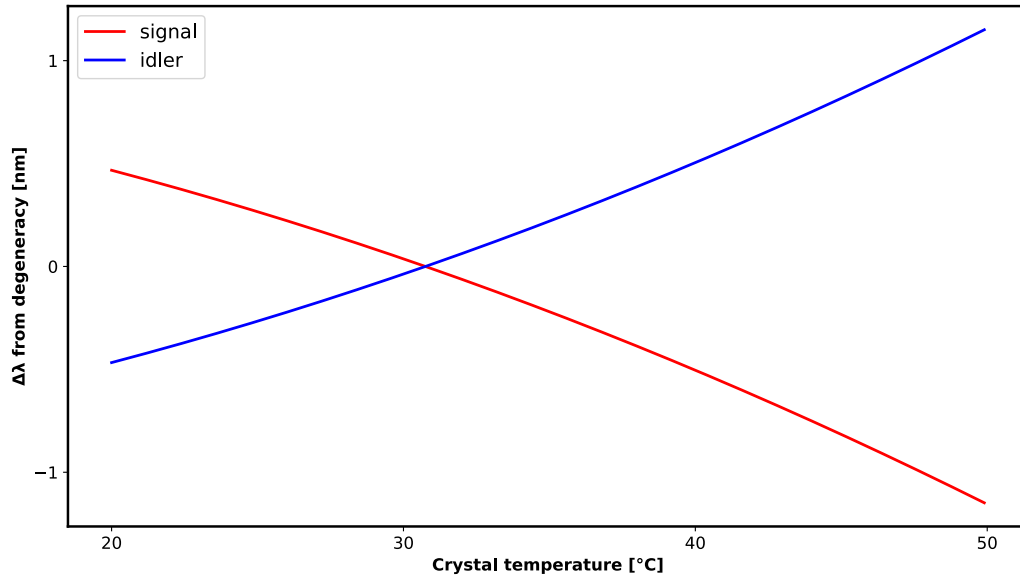


Figure 6.1: Phase matching curve: Temperature dependence of the wavelengths λ_s (red curve) and λ_i (blue curve) of the signal- and idler- photons for a pump wavelength of 775 nm, a poling period of $\Lambda(20^\circ\text{C}) = 46.2\ \mu\text{m}$ and quasi phase matching order $m = -1$. The crossing point temperature is expected to be 31.1°C .

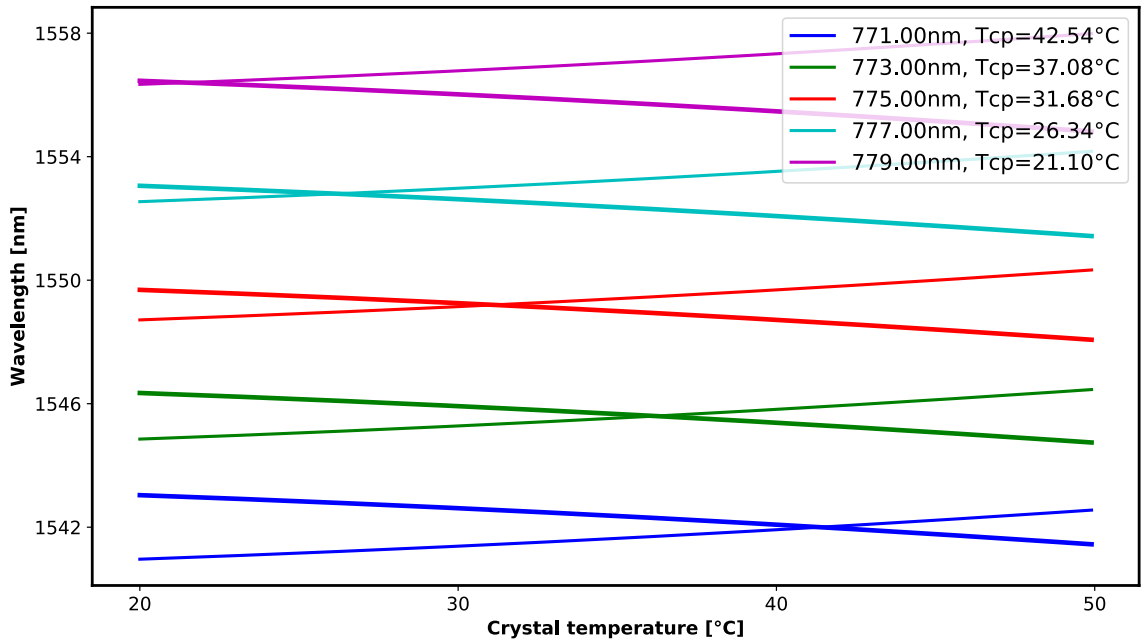


Figure 6.2: Phase-matching curves for different pump-wavelengths and a poling period of $\Lambda = 46.2\ \mu\text{m}$. The temperature T_{cp} at which signal- and idler-wavelength are degenerate decreases with increasing pump wavelength.

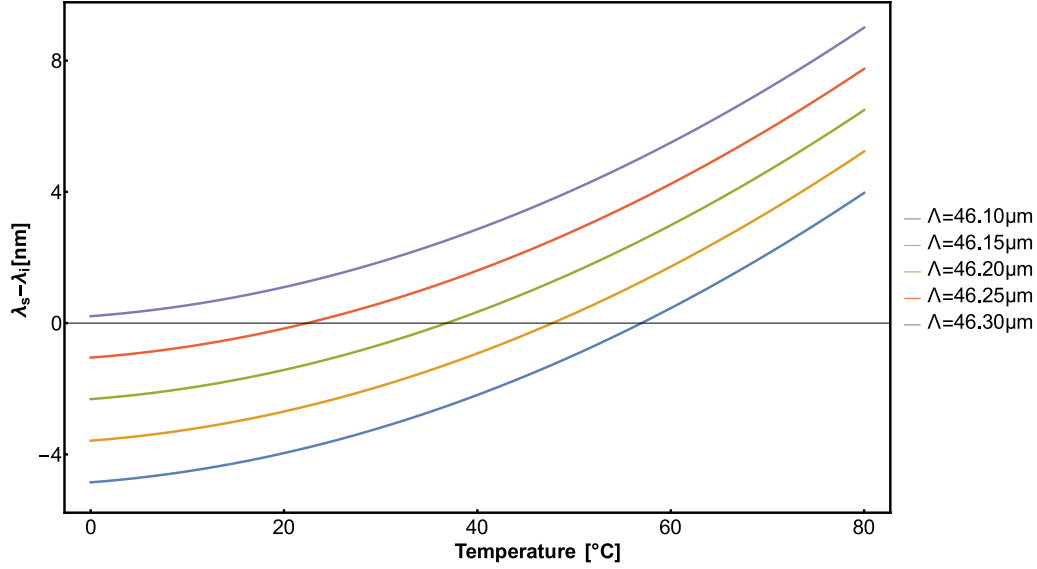


Figure 6.3: Temperature-dependence of the wavelength-gap $\lambda_s - \lambda_i$ for different poling periods at a pump wavelength of 775 nm, visualizing the influence of the poling period on the temperature at which signal- and idler-wavelengths are degenerate. A change in the poling period by 50 nm results, in the region of interest (close to the value stated by the manufacturer), in a temperature shift of more than 15 °C.

photon state is fully characterised by $f(\omega_s, \omega_i)$, which is called the joint spectral amplitude (JSA). It can be expressed as $f(\omega_s, \omega_i) = \alpha(\omega_s, \omega_i) \cdot \phi(\omega_s, \omega_i)$, with the pump envelope amplitude (PEA) $\alpha(\omega_s, \omega_i)$ (reflecting energy conservation) and the phase matching envelope amplitude (PMA) $\phi(\omega_s, \omega_i)$ (reflecting momentum conservation) as follows[36, 37]:

$$\alpha^g(\omega_s, \omega_i) \propto \exp\left(\left(\frac{\omega_s + \omega_i - \omega_p}{\sigma_p}\right)^2\right) \quad (6.5a)$$

$$\alpha^s(\omega_s, \omega_i) \propto \text{sech}\left((\omega_s + \omega_i - \omega_p) \cdot \frac{2 \text{arccosh}(\sqrt{2})}{\Delta\omega}\right) \quad (6.5b)$$

$$\phi(\omega_s, \omega_i) = \exp\left(\frac{i\Delta k L}{2}\right) \text{sinc}\left(\frac{\Delta k \cdot L}{2}\right) \quad (6.5c)$$

with $\Delta k = (1/c) \cdot (n_s \omega_s + n_i \omega_i - n_p \omega_p)$, the spectral bandwidth of the pump σ_p and the FWHM of the pump pulse frequency distribution $\Delta\omega_p$. Here, $\text{sinc}(x) = \frac{\sin(x)}{x}$. The pump envelope amplitude depends, of course, on the pump shape. α^g is applicable for a gaussian distribution, while α^s is valid for a sech^2 distribution. Exemplary plots for the respective intensities ($|\alpha^g(\omega_s, \omega_i)|^2$, $|\alpha^s(\omega_s, \omega_i)|^2$ and $|\phi(\omega_s, \omega_i)|^2$), are shown in Fig. 6.4. As can be seen in Equ. 6.5, the width of the PEA is proportional to the bandwidth of the pump and the width of the PMA is inverse proportional to the length of the crystal.

The pulsed sources under examination use a scheme which was first implemented in [38]: The non-linear crystal phase matched for collinear (when signal and idler photons are created, they travel parallel to the direction of the pump beam) Type II (signal and idler photon are created with orthogonal polarisation) down-conversion

resides in a sagnac interferometer in which it is pumped from both directions. The output photons are combined at a polarizing beam splitter. The crystal material is periodically poled potassium titanyl phosphate (KTiOPO₄), short PPKTP, with a poling period of $\Lambda = 46.2 \mu\text{m}$. A schematic representation of the source can be seen in Fig. 6.5. The pump beam is blocked with two edge-pass filters with a cut-on wavelength of 900 nm and 1300 nm while signal- and idler photons are collected using single mode fibres and detected with superconducting nanowire single photon detectors (SNSPDs). To allow for temperature tuning of the phase matching, the crystal is placed inside of an oven. Two sources were examined, both with a crystal of 30 mm length.

As pump laser, a mode locked Ti:Sapphire laser was used (Coherent Mira 900). Depending on the mode-locking scheme, the shape of the intensity distribution of such a laser may better be described by a secant hyperbolic squared (sech^2) distribution instead of a gaussian, where the secant hyperbolic is:

$$\text{sech}(x) = \frac{1}{\cosh(x)} = \frac{2}{e^x + e^{-x}}. \quad (6.6)$$

With a nomenclature for the parameters as in the following function (used to fit the spectra to sech^2 distributions)

$$y(x) = y_0 + A \cdot \left(\text{sech}^2(B(x - x_0)) \right), \quad (6.7)$$

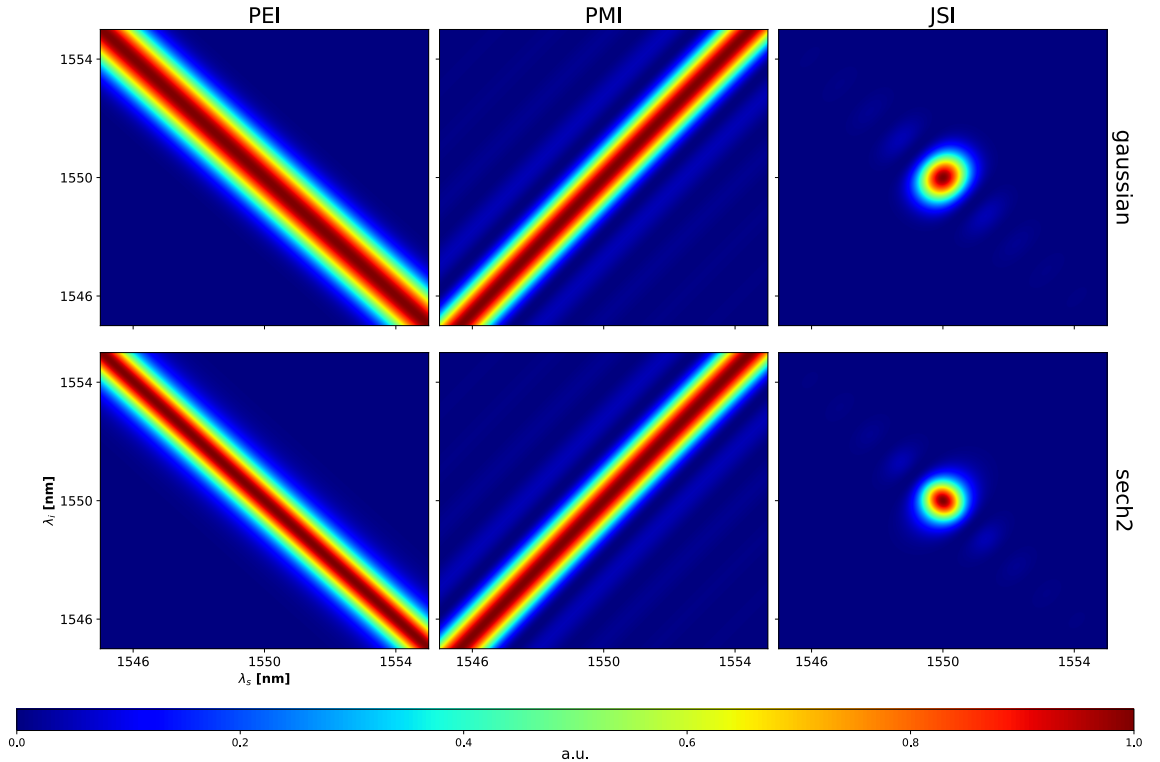


Figure 6.4: Pump envelope intensity (PEI), phase matching envelope intensity (PMI) and joint spectral intensity (JSI) for arbitrary pump- and crystal parameters, for both a gaussian and a sech^2 spectral distribution of the pump which share the same autocorrelator signal duration.

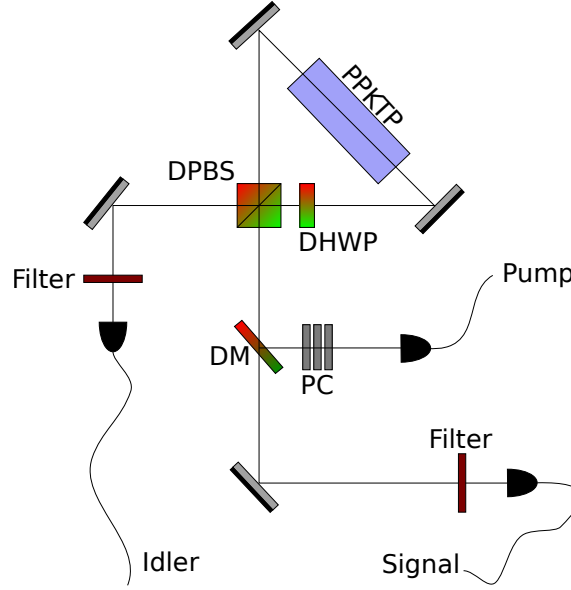


Figure 6.5: Schematic setup of the SPDC source under examination. PC: Polarisation controller. DM: Dichroic mirror. DPBS: Dichroic polarizing beam splitter. DHWP: Dichroic half wave plate. The PPKTP crystal is placed inside of an oven (not drawn). The filters are edge-pass, transmitting wavelengths $\lambda > 900$ nm and $\lambda > 1300$ nm to block the pump beam. Signal and idler photons are collected using single mode fibres.

the FWHM of this very same function is given by

$$\text{FWHM}_{\text{sech}^2} = \frac{2 \cosh^{-1}(\sqrt{2})}{B}. \quad (6.8)$$

When the temporal width of a pulsed source is measured using an intensity autocorrelator, the width of the recorded signal is bigger than the width of the input pulse, and needs to be corrected with a numerical factor f_τ [19], which is also called "deconvolution factor". For gaussian pulse shapes, it is $f_\tau^g = 1/\sqrt{2}$, and for sech^2 shaped pulses it is $f_\tau^s \approx 0.647$.

6.1.2 Measurements

For subsequent analysis it was necessary to check beforehand whether the assumption that the intensity distribution of signal- and idler photons is better described by a sech^2 than by a gaussian distribution is correct. An exemplary intensity distribution is pictured in Fig. 6.6, with fits to both gaussian and sech^2 distributions. For both, the sum of the squared differences between measured data and fit is given as a goodness of fit measure (denoted with χ^2). In fact, both the signal- and idler intensities are better described by a sech^2 distribution.

As the sources under examination are the building blocks of a multi-photon source in which interference visibility plays an important role, the temperature at which signal and idler photons are degenerate is of interest. To find it, a phase matching curve was acquired by measuring signal and idler spectra at different temperatures. The pump wavelength was set to 774.4 nm (measured with an APE pulse scope autocorrelator) and the spectra of signal and idler photons were recorded

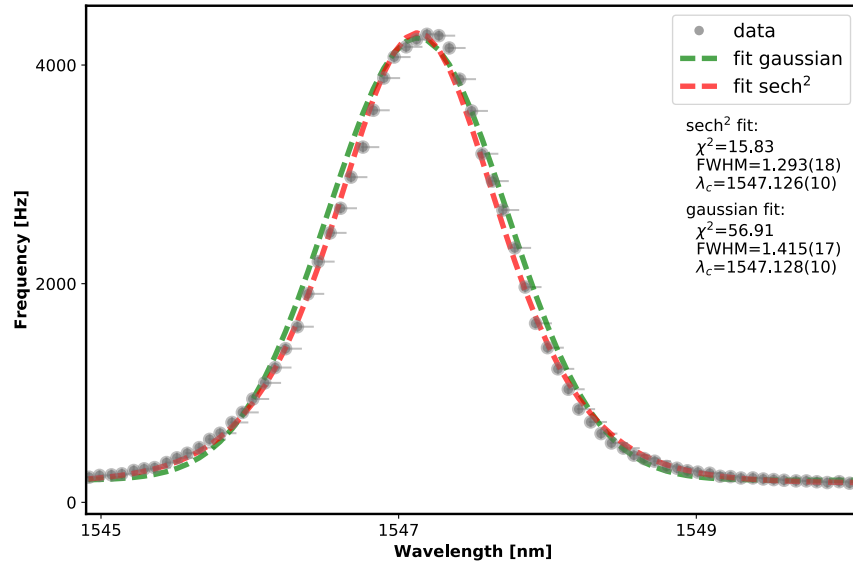


Figure 6.6: Spectrum of the idler photons of source 1, created with a pump of 773.1 nm and a pulse-width of 1.99 ps at a crystal temperature of 15 °C. The data (grey) is fitted to both a gaussian (green) and a sech^2 (red) distribution. The total sum of squared differences is given for both fits (χ^2). These measures indicate that the intensity distribution is better described by a sech^2 distribution than by a gaussian.

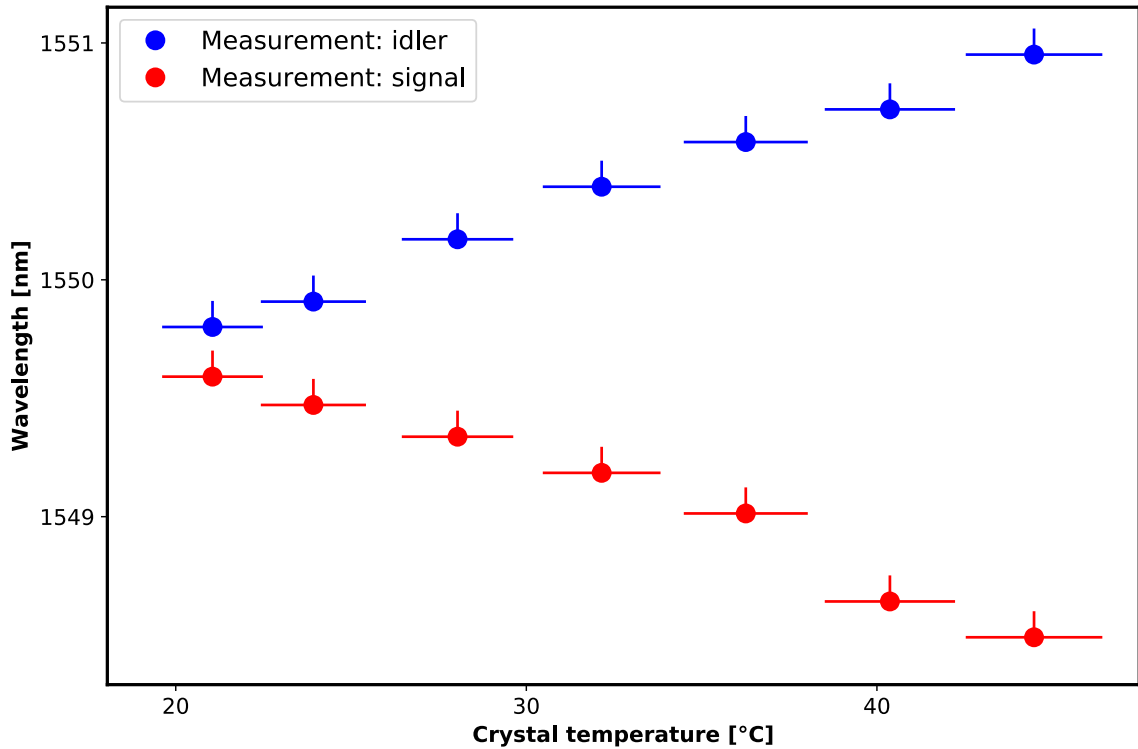


Figure 6.7: Phase-matching curve of source 2, with a pump wavelength of 774.4 nm and a pulse-width of 1.48 ps. The crossing point of idler (blue) and signal (red) wavelength was expected to be at 32.9 °C but could not be found there.

for varying crystal temperatures: from 25 °C to 50 °C in steps of 5 °C plus one measurement at 21.5 °C room temperature. The recorded phase matching curve for source 2 is depicted in Fig. 6.7.

According to the theory of quasi phase matching (Equ. 6.3), at this pump wavelength signal and idler photons should be degenerate at a temperature of 32.9 °C. As the experimental data clearly does match this expectation, another phase matching curve was recorded for source 1 to check if this behaviour is observable in both sources. The pump wavelength was lowered to 773.1 nm as this shifts the crossing point to higher temperatures, as illustrated in Fig 6.2. Figure 6.8 shows this data. The crossing point in this case was expected to be at 36.3 °C, but was found at (23.71 ± 1.53) °C.

As both sources show the same qualitative behaviour concerning the shift of the crossing point temperature, the possibility of an offset of one of the temperature sensors was ruled out. This assumption was backed up by setting one of the ovens to 25 °C and measuring its temperature with an additional temperature sensor (Thorlabs TSP-TH with an accuracy of ± 0.5 °C) taped onto the top of the oven. The temperature was recorded for 23 hours to be able to also observe potential daytime dependent effects. The mean of the measured temperature was (26.25 ± 0.12) °C with a peak to peak difference of 0.78 °C. Additionally, the difference between the readings of the two sensors was recorded at several oven-temperatures and found to be increasing with increasing temperature. For further analysis, the readings of the internal temperature sensor were corrected to match those of the external one. However, at a set temperature of 25 °C, the difference between the two sensors was only (1.25 ± 0.12) °C and thus did not explain the observed crossing point temperature.

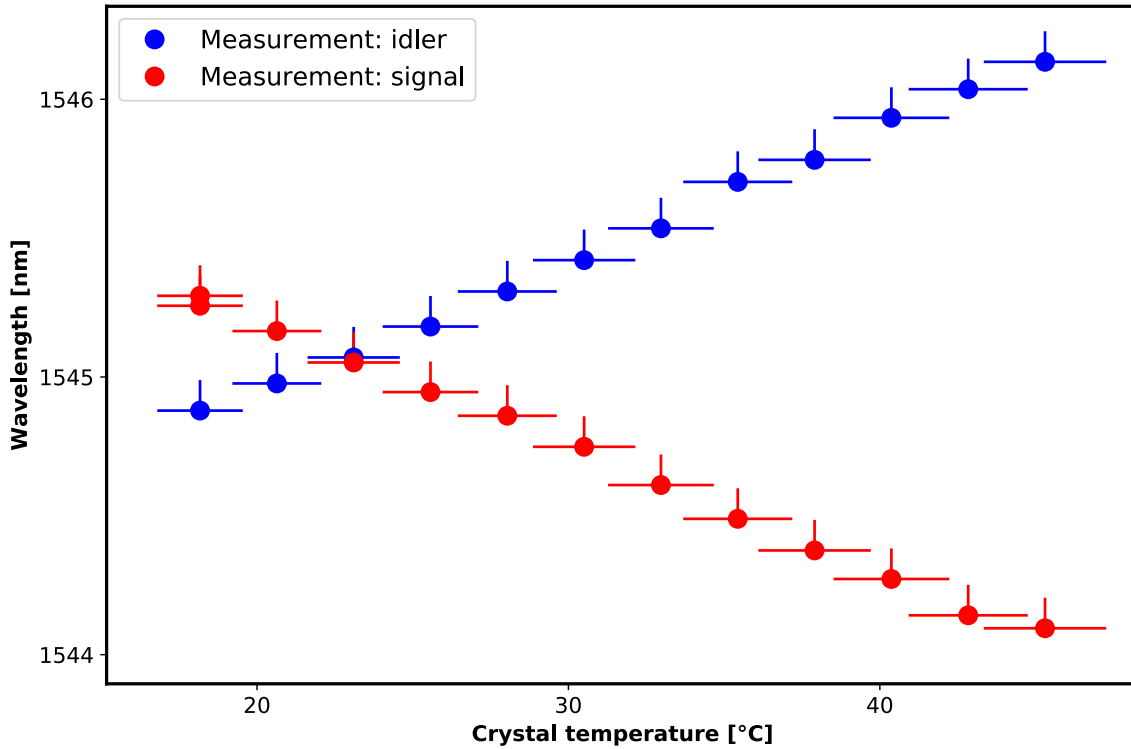


Figure 6.8: Phase-matching curve of source 1, with a pump wavelength of 773.1 nm and a pulse-width of 1.70 ps. The crossing point of idler (blue) and signal (red) wavelength is at (23.71 ± 1.53) °C, although it was expected to be at 36.3 °C.

Another possible explanation would be a concurrent error in the measurement of the pump wavelength and an offset in the spectrometer calibration. A crossing point temperature of 23.7 °C would be expected at a pump wavelength of 778 nm instead of 773.1 nm. This possibility could also be ruled out by using two additional different spectrometers (Ocean Optics USB2000 and HR4000) to check the pump wavelength.

The remaining and most likely reason for the findings seems to be a small deviation of the poling period of the crystal from the stated value. In Fig. 6.9 (Fig. 6.10), the measured phase matching curve for source 1 (source 2) with a pump wavelength of 773.1 nm is plotted together with the expected curves for both the stated poling period of $\Lambda_s = 46.2 \mu\text{m}$ and a slightly different poling period ($\Lambda_{\text{eff},S1}^{773.1} = (46.2496 \pm 0.0034) \mu\text{m}$ for source 1 and $\Lambda_{\text{eff},S2}^{773.1} = (46.2454 \pm 0.0039) \mu\text{m}$ for source 2) which leads to a correct prediction of the crossing point temperature. The relative difference between the stated and the effective poling periods is approximately 10^{-3} . Unfortunately, it was impossible to receive data about the fabrication uncertainties from the manufacturer (Raicol crystals ltd.).

The check of the pump-wavelength dependence of the crossing point temperature was pioneered by the recording of coarse phase matching curves for 5 different pump wavelengths, to have a rough approximation of where the crossing point would be. These curves can be seen in Fig. 6.11. For each phase matching curve, the data points for signal- and idler wavelengths were fitted hyperbolically.

The crossing points used in the analysis of the temperature dependence of the crossing point were determined at 5 different wavelengths, at which they were acces-

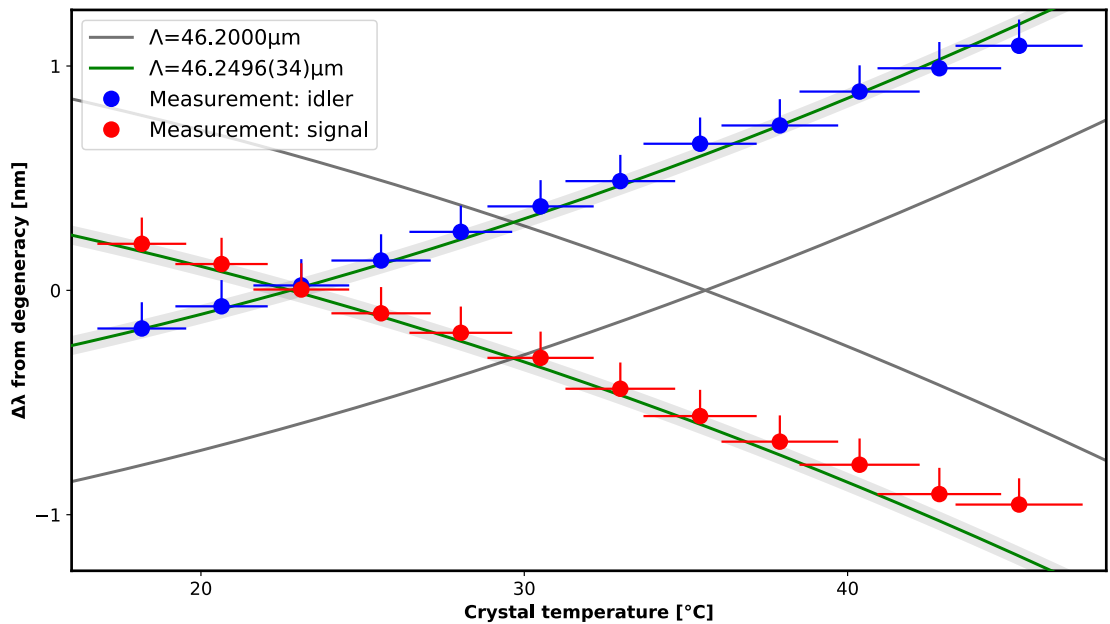


Figure 6.9: Phase-matching curve (measurement) of source 1, at a pump wavelength of 773.1 nm and a pulse-width of 1.74 ps (assuming a transformation limited pulse of sech^2 shape). The crossing point of idler (blue) and signal (red) wavelength is at $(22.77 \pm 1.05) ^\circ\text{C}$, while the expected phase-matching curve (grey) exhibits a crossing point at 36.3 °C. A possible explanation is a slightly different poling period of $\Lambda_{\text{eff},S1}^{773.1} = (46.2496 \pm 0.0034) \mu\text{m}$ instead of $\Lambda_s = 46.2 \mu\text{m}$. The phase-matching curve with the adjusted poling period is shown as well (green with grey uncertainty band).

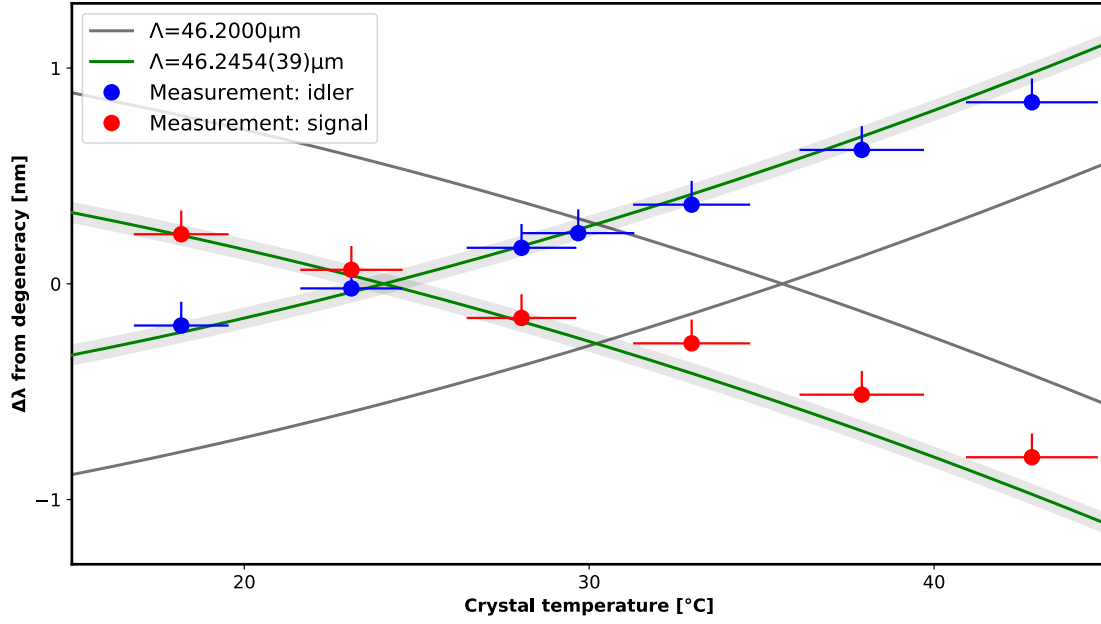


Figure 6.10: Phase-matching curve (measurement) of source 2, pump wavelength of 773.1 nm and a pulse-width of 1.59 ps (assuming a transformation limited pulse of sech^2 shape). The crossing point of idler (blue) and signal (red) wavelength is at $(24.05 \pm 1.17)^\circ\text{C}$, while the expected phase-matching curve (grey) exhibits a crossing point at 36.3°C . A possible explanation is a slightly different poling period of $\Lambda_{\text{eff},S2}^{773.1} = (46.2454 \pm 0.0039) \mu\text{m}$ instead of $\Lambda_s = 46.2 \mu\text{m}$. The phase-matching curve with the adjusted poling period is shown as well (green with grey uncertainty band).

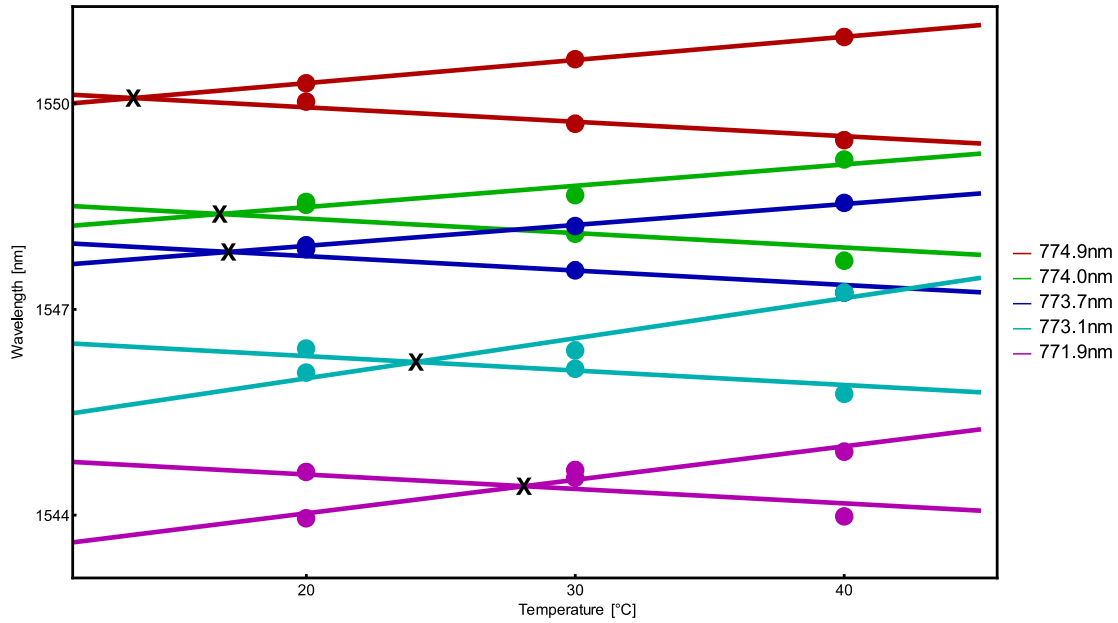


Figure 6.11: Phase-matching curves of source 1, for 5 different pump wavelengths between 771.9 nm and 774.9 nm. The temperature of degenerate signal- and idler wavelengths increases with decreasing pump-wavelength, according to the theoretical prediction (see Fig. 6.2). The data is fitted to hyperbolic functions. For each pump wavelength, the intersection between signal- and idler wavelength is marked with a blue point.

sible by the oven in use, by altering the temperature until signal- and idler spectra with coinciding central wavelength were found. The results can be seen in Fig. 6.12, with simulations showing the expected behaviour for both Λ_s and Λ_{eff} . For each data point, a poling period explaining its temperature best was calculated. These values coincide to within 6 nm and show no monotonic wavelength dependence. Therefore, a simulation of the wavelength dependence of the crossing point utilizing the mean of these values, $\Lambda_{\text{eff}} = (46.2462 \pm 0.0019) \mu\text{m}$, is included in Fig. 6.12 as well. The error shown is due to the uncertainty of the central wavelength determination.

The dependence of the FWHM of the spectral distributions of signal and idler photons was recorded for a pump wavelength of 733.1 nm at a temperature of 24 °C, the point of degenerate signal and idler wavelengths. The pulse width of the pump was measured with an autocorrelator and thus corrected with the appropriate deconvolution factor f_τ^s . The FWHM of signal and idler can be obtained by finding the curve at half maximum of the JSI and projecting the curve onto the respective axis[39]. As expected, the spectral FWHMs of signal and idler decrease with increasing temporal width of the pump pulses, as can be seen in Fig. 6.13(A) (for source 1) and Fig. 6.13(B) (for source 2).

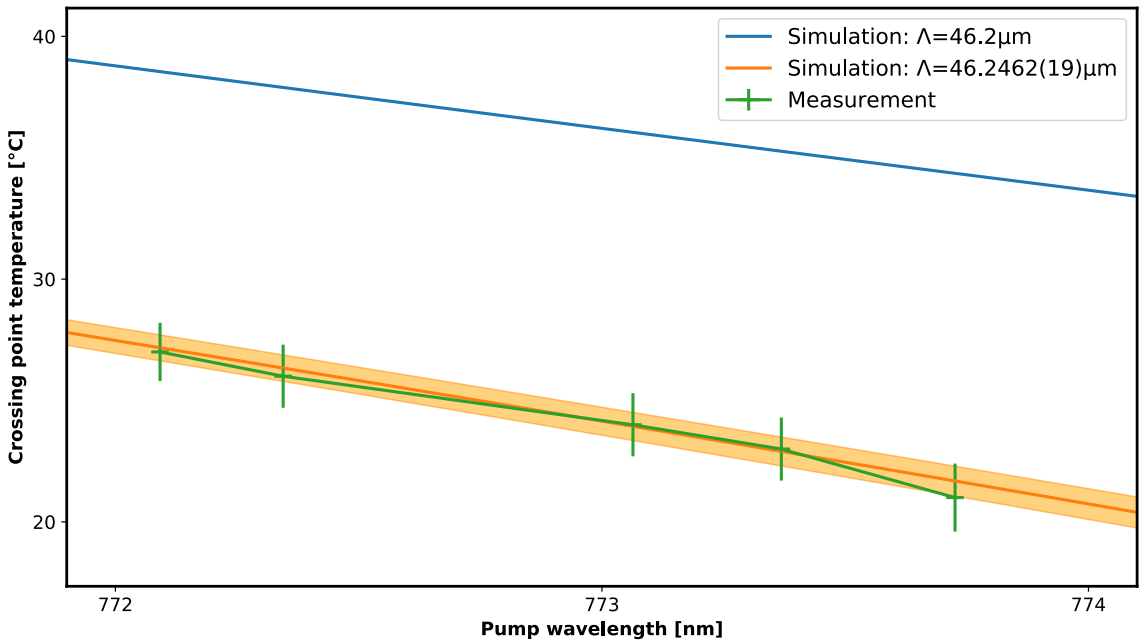


Figure 6.12: The simulation of the pump wavelength dependence of the crossing point temperature of source 1 for the stated poling period $\Lambda = 46.2 \mu\text{m}$ (blue) does not match the measured data points (green). For each measured crossing point temperature, an 'effective' poling period which does explain the corresponding temperature was determined. A simulation using the mean of these poling periods $\Lambda_{\text{eff}} = (46.2462 \pm 0.0019) \mu\text{m}$ is shown in orange.

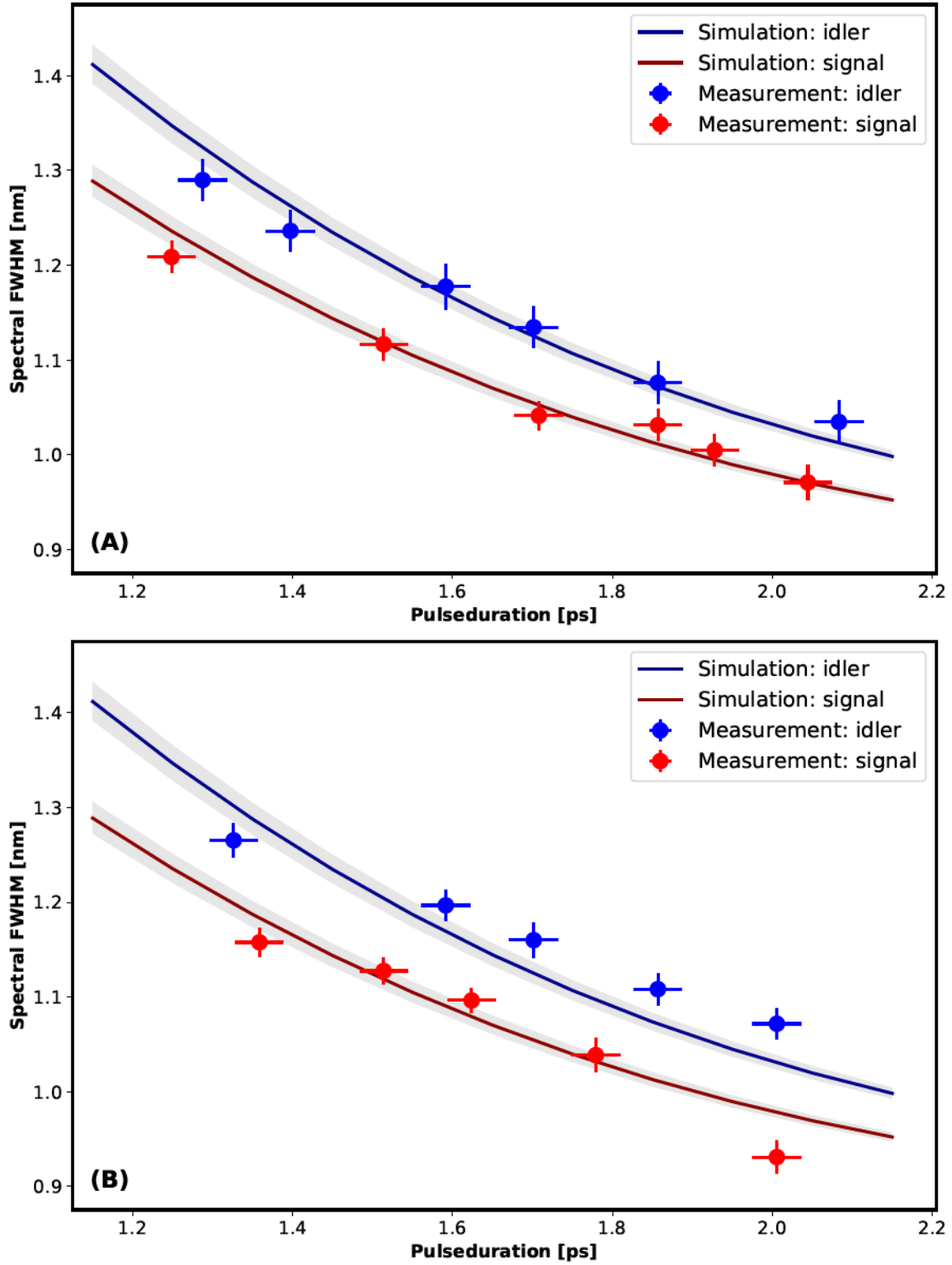


Figure 6.13: Measurement and simulation of the FWHM of the marginal spectra of source 1 (A) and source 2 (B). The spectral FWHMs of signal (red) and idler (blue) photons decrease with increasing temporal width of the pump pulses.

Chapter 7

Conclusion and Outlook

A fibre coupled, grating based spectrometer for wavelengths around 1550 nm with a resolution of 0.12 nm was built. Compared to commercial products the cost could be reduced (the price of the individual components was approximately € 2500) at the expense of speed and wavelength range. To quantify this statement, the cheapest offer found for a fibre-coupled monochromator with comparable specifications was about € 11000 for a *SOLAR Laser Systems M266-I* with a resolution of 0.126 nm. The manufacturer defines the wavelength range of this monochromator as the range in which the diffraction efficiency of the grating is greater than 40%. For this device, it is 660 nm-1800 nm. Its scan speed is 50 nm s^{-1} at max. It has to be mentioned, however, that at the single photon level, the resulting approximate $2.5 \mu\text{s}$ per resolution point at this speed might be too short to acquire enough photons to reconstruct their spectral distribution. Thus, this advantage in speed becomes less significant when dealing with low detection rates.

The characterisation measurements show that aberrations distinctly affect the achievable resolution. As the alignment of the off-axis parabolic mirrors was one of the trickiest parts of the setup and could therefore be considered the root of the problem, it might be worth to consider using achromatic lenses instead in a follow-up design. This would also reduce the actual costs even further.

Increasing the efficiency of the device may readily be done by using Au-coated mirrors instead of Al-coated ones at the expense of higher costs. In case future arrangements use achromatic lenses, the efficiency could be even further improved due to the possibility of using anti-reflection coatings for the lenses.

Since the anamorphic magnification of the grating reduces the coupling efficiency, investigating the advantages and drawbacks of the usage of beam shaping optics, like cylindrical lenses or an anamorphic prism pair, in order to reduce the ellipticity of the diffracted beam might prove beneficial as this would allow for improved mode matching with the single-mode fibres.

Bibliography

- [1] P. Fara, “Newton shows the light: a commentary on Newton (1672) ’A letter ... containing his new theory about light and colours ...’ ”, *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* **373** (2015) 10.1098/rsta.2014.0213, eprint: <http://rsta.royalsocietypublishing.org/content/373/2039/20140213.full.pdf>.
- [2] I. Newton, “A Letter of Mr. Isaac Newton, Professor of the Mathematicks in the University of Cambridge; Containing His New Theory about Light and Colours: Sent by the Author to the Publisher from Cambridge, Febr. 6. 1671/72; In Order to be Communicated to the R. Society”, *Philosophical Transactions* **6**, 3075–3087 (1671) 10.1098/rstl.1671.0072, eprint: <http://rstl.royalsocietypublishing.org/content/6/69-80/3075.full.pdf+html>.
- [3] I. Newton, *Opticks: or, a treatise of the reflexions, refractions, inflexions and colours of light. also two treatises of the species and magnitude of curvilinear figures*. Edited by p. f. S. S. London and B. Walford. (1704).
- [4] G. Kirchhoff, “Über das Verhältniss zwischen dem Emissionsvermögen und dem Absorptionsvermögen der Körper für Wärme und Licht”, *Annalen der Physik* **185**, 275–301 (1860) 10.1002/andp.18601850205.
- [5] G. Kirchhoff and R. Bunsen, “Chemische Analyse durch Spectralbeobachtungen”, *Annalen der Physik* **189**, 337–381 (1861) 10.1002/andp.18611890702.
- [6] H. Kragh, “The solar element: a reconsideration of helium’s early history”, *Annals of Science* **66**, 157–182 (2009) 10.1080/00033790902741633, eprint: <http://dx.doi.org/10.1080/00033790902741633>.
- [7] Y.-H. Kim and W. P. Grice, “Measurement of the spectral properties of the two-photon state generated via type II spontaneous parametric downconversion”, *Optics Letters* **30**, 908 (2005) 10.1364/ol.30.000908.
- [8] S. Devons, “Lewis Morris Rutherford 1816-1892”, *Appl. Opt.* **15**, 1731–1740 (1976) 10.1364/AO.15.001731.
- [9] D. R. F. Hopkinson, “An Optical Problem, Proposed by Mr. Hopkinson, and Solved by Mr. Rittenhouse”, *Transactions of the American Philosophical Society* **2**, 201–206 (1786).
- [10] J. v. Fraunhofer, “Neue Modifikation des Lichtes durch gegenseitige Einwirkung und Beugung der Strahlen, und Gesetze derselben”, *Denkschriften der königlichen Akademie der Wissenschaften zu München für das Jahr 1821 - Classe der Mathematik und Naturwissenschaften* **8**, 3–76 (1824) 10.3931/erara-4278.

-
- [11] J. v. Fraunhofer, “Kurzer Bericht von den Resultaten neuerer Versuche über die Gesetze des Lichtes, und die Theorie derselben”, *Annalen der Physik* **74**, Digitalisierte Fassung: http://www.mdz-nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:bvb:12-bsb10927396-8_337-378 (1823) 10.1002/andp.18230740802.
- [12] C. Palmer, *Diffraction grating handbook*, 7th ed. (2014).
- [13] R. G. Driggers, ed., *Encyclopedia of optical engineering*, Vol. Las-Pho (CRC Press, 2003).
- [14] J. E. Howard, “Imaging properties of off-axis parabolic mirrors”, *Appl. Opt.* **18**, 2714–2722 (1979) 10.1364/AO.18.002714.
- [15] K. Kataoka, “Estimation of coupling efficiency of optical fiber by far-field method”, *Optical Review* **17**, 476–480 (2010) 10.1007/s10043-010-0086-4.
- [16] E. Loewen and E. Popov, *Diffraction gratings and applications*, Optical Science and Engineering (Taylor & Francis, 1997).
- [17] M. Czerny and A. F. Turner, “Über den Astigmatismus bei Spiegelspektrometern”, *Zeitschrift für Physik* **61**, 792–797 (1930) 10.1007/BF01340206.
- [18] O. von Littrow, “Über eine neue Einrichtung des Spectralapparates”, *Sitzungsberichte der Kaiserlichen Akademie der Wissenschaften, mathematisch - naturwissenschaftliche Classe, Abteilung II Band* **47**, 26–33 (1863).
- [19] F. Träger, ed., *Springer handbook of lasers and optics* (Springer Nature, 2012), 10.1007/978-3-642-19409-2.
- [20] H. Ebert, “Zwei Formen von Spectrographen”, *Annalen der Physik* **274**, 489–493 (1889) 10.1002/andp.18892741113.
- [21] W. G. Fastie, “A small plane grating monochromator”, *J. Opt. Soc. Am.* **42**, 641–647 (1952) 10.1364/JOSA.42.000641.
- [22] W. G. Fastie, “Image forming properties of the ebert monochromator”, *J. Opt. Soc. Am.* **42**, 647–651 (1952) 10.1364/JOSA.42.000647.
- [23] A. H. C. P. Gillieson, “A new spectrographic diffraction grating mounting”, *J. Sci. Instrum.* **26**, 335 (1949).
- [24] G. S. Monk, “A mounting for the plane grating”, *Journal of the Optical Society of America* **17**, 358 (1928) 10.1364/josa.17.000358.
- [25] W. Welford, “VI- aberration theory of gratings and grating mountings”, in *Progress in optics*, Vol. 4, edited by E. Wolf, Progress in Optics (Elsevier BV, 1965), pp. 241–280, 10.1016/s0079-6638(08)70493-1.
- [26] R. C. M. Learner, “Spectrograph design, 1918-68”, *Journal of Physics E: Scientific Instruments* **1**, 589 (1968).
- [27] S. Ichimaru, H. Takenaka, K. Namikawa, E. M. Gullikson, M. Maruyama, and S. Oku, “Demonstration of the high collection efficiency of a broadband mo/si multilayer mirror with a graded multilayer coating on an ellipsoidal substrate”, *Review of Scientific Instruments* **86**, 093106 (2015) 10.1063/1.4929708.
- [28] H. Haber, “The torus grating”, *J. Opt. Soc. Am.* **40**, 153–165 (1950) 10.1364/JOSA.40.000153.
-

- [29] H. A. Rowland, “Preliminary notice of the results accomplished in the manufacture and theory of gratings for optical purposes”, *Nature* **29**, 211–213 (1882) [10.1038/026211b0](#).
- [30] R. Lasertechnik, *TTM8000 time tagging module with 8-channels*, Roithner Lasertechnik GmbH (Wiedner Hauptstraße 76 1040 Vienna, Austria, 2015).
- [31] J. M. Lerner, “Imaging spectrometer fundamentals for researchers in the biosciences - a tutorial”, *Cytometry Part A* **69A**, 712–734 (2006) [10.1002/cyto.a.20242](#).
- [32] F. König and F. N. C. Wong, “Extended phase matching of second-harmonic generation in periodically poled KTiOPO_4 with zero group-velocity mismatch”, *Applied Physics Letters* **84**, 1644–1646 (2004) [10.1063/1.1668320](#).
- [33] K. Fradkin, A. Arie, A. Skliar, and G. Rosenman, “Tunable midinfrared source by difference frequency generation in bulk periodically poled KTiOPO_4 ”, *Applied Physics Letters* **74**, 914–916 (1999) [10.1063/1.123408](#).
- [34] S. Emanuelli and A. Arie, “Temperature-dependent dispersion equations for KTiOPO_4 and KTiOAsO_4 ”, *Appl. Opt.* **42**, 6661–6665 (2003) [10.1364/AO.42.006661](#).
- [35] H. Vanherzeele and J. D. Bierlein, “Magnitude of the nonlinear-optical coefficients of KTiOPO_4 ”, *Optics Letters* **17**, 982 (1992) [10.1364/ol.17.000982](#).
- [36] N. Bruno, A. Martin, T. Guerreiro, B. Sanguinetti, and R. T. Thew, “Pulsed source of spectrally uncorrelated and indistinguishable photons at telecom wavelengths”, *Optics Express* **22**, 17246 (2014) [10.1364/oe.22.017246](#).
- [37] F. Laudenbach, H. Hübel, M. Hentschel, P. Walther, and A. Poppe, “Modelling parametric down-conversion yielding spectrally pure photon pairs”, *Optics Express* **24**, 2712 (2016) [10.1364/oe.24.002712](#).
- [38] T. Kim, M. Fiorentino, and F. N. C. Wong, “Phase-stable source of polarization-entangled photons using a polarization sagnac interferometer”, *Phys. Rev. A* **73**, 012316 (2006) [10.1103/PhysRevA.73.012316](#).
- [39] V. Ansari, “Correlations and correlation time of two-photon states”, MA thesis (University of Paderborn, 2013).

Appendices

Appendix A

Dinspect: Example spectrometer configuration file

A spectrometer configuration file contains properties of the spectrometer, such as the focal lengths of the lenses or mirrors, the slit sizes, and many more. It also contains the commands used to control the PCBMotor. This way, it is possible to operate different spectrometers without altering the control program. It is even possible to use a different rotation stage than the PCBMotor, as long as it can be controlled via ASCII commands sent over a serial connection, and supports all necessary commands.

Listing A.1 holds a minimal functional example of a configuration file. The information therein is sufficient to perform a calibration, which will store additional quantities in the file.

```
1 #Example of a minimal spectrometer configuration file
#
3 #Lines starting with a '#' will be treated as comments.
#The syntax for the assignment of values to variables is:
5 #VARIABLE=VALUE
#
7 #identification of spectrometer
Type=1550
9 Subversion=1
VersionString=1550v1
11
#
13 #Grating properties
#
15 #Grating constant in [lines/mm]
GratingConstant=1200.0
17 #Ruled width of the grating in [mm]
grating_width=50.0
19
#Fiber mode field diameters in [µm]
21 mfdin=10.40
mfdout=10.40
23
#Numerical aperture of in/out fibres
25 nain=0.140
naout=0.140
27
#Cut-off wavelength of fibres in [nm]
29 lcutoffin=1260.0
```

```

31 | lcutoffout=1260.0
33 | #mirror/lens focal lengths in [mm]
33 | #in: collimation; out: focusing
35 | fin=54.450
35 | fout=25.40
37 | #
37 | #motor properties
39 | #times are given in milliseconds
39 | #
41 | #number of steps between mechanical stop and measurement starting
41 | position
41 | offset_steps=518
43 | #time to wait for a single step to finish
43 | steptime=1000
45 | #command for movement to perform between taking to datapoints
45 | scanmove_command=u-2
47 | #command to go to next rising/falling edge
47 | nextrisingedge_command=UP-9999
49 | nextfallingedge_command=UN-9999
49 | #time to wait for nextrisingedge_command/nextfallingedge_command to
49 | finish
51 | nextedge_command_time=5000
51 | #Time to wait when going from arbitrary position to mechanical stop
53 | parkcommand_waittime=3000
53 | #Time it takes to go from mechanical stop 'offset_steps' to starting
53 | position
55 | gotostart_waittime=300000
55 | #Time between single steps the PCB Motor waits when it goes to the
55 | starting position
57 | gotostart_timebetweensteps=250
59 | #
59 | #END OF SPECTROMETER CONFIG FILE
61 | #

```

Listing A.1: ./specconf.d/1550v1.sconf

A minimal functional example of a spectrometer configuration file.

Appendix B

Dinspect: code documentation

In the following, a thorough documentation of the source code of the control software *dinspect* will be given, including a description of all used variables, functions and classes as well as their interplay. It is meant to offer an entry point for anyone who wants to alter the software.

In addition, the first pages of the documentation include a description of the GUI elements of the software and their usage.

Dinspect

The detector independent spectrometer control tool

Generated by Doxygen 1.8.14

Contents

1	Dinspect	1
1.1	Installation	1
1.1.1	Step 1: Prerequisites	1
1.2	Building	1
1.2.1	Windows	1
1.2.2	Linux	2
1.3	Usage	2
1.4	Fit spectra	8
2	Bug List	15
3	Namespace Index	17
3.1	Namespace List	17
4	Hierarchical Index	19
4.1	Class Hierarchy	19
5	Class Index	21
5.1	Class List	21
6	File Index	23
6.1	File List	23

7 Namespace Documentation	25
7.1 dmath Namespace Reference	25
7.1.1 Detailed Description	26
7.1.2 Function Documentation	26
7.1.2.1 calc_alpha()	26
7.1.2.2 calc_angle_offset()	27
7.1.2.3 calc_angle_offset_without_stepsize()	27
7.1.2.4 calc_bandpass()	28
7.1.2.5 calc_bandpasses_in_scanrange()	28
7.1.2.6 calc_beta()	29
7.1.2.7 calc_coupling_efficiency()	29
7.1.2.8 calc_deviationangle()	29
7.1.2.9 calc_deviationangle_without_stepsize()	30
7.1.2.10 calc_how_many_edges_to_skip()	31
7.1.2.11 calc_lambda_from_alpha()	31
7.1.2.12 calc_number_of_steps_in_scanrange()	32
7.1.2.13 calc_range()	32
7.1.2.14 calc_steps_between_edges()	32
7.1.2.15 calc_steps_in_BP()	33
7.1.2.16 calc_stepsize() [1/2]	33
7.1.2.17 calc_stepsize() [2/2]	34
7.1.2.18 calc_unneeded_stepsize()	34
7.1.2.19 find_max() [1/2]	35
7.1.2.20 find_max() [2/2]	35
7.1.2.21 fwhm_gauss()	35
7.1.2.22 normal_distribution()	36
7.1.2.23 recalc_steps_in_BP()	36
7.1.2.24 recalc_wavelength_at_step()	37
7.1.2.25 relate_maxpos_to_angle_difference()	37
7.2 helper Namespace Reference	38
7.2.1 Detailed Description	38
7.2.2 Function Documentation	38
7.2.2.1 error()	38
7.2.2.2 info()	38
7.2.2.3 message()	39
7.2.2.4 warning()	39

8	Class Documentation	41
8.1	AboutDialog Class Reference	41
8.1.1	Detailed Description	42
8.2	CalibrationDialog Class Reference	42
8.2.1	Detailed Description	44
8.2.2	Member Function Documentation	44
8.2.2.1	CalibrationDialogDone	44
8.2.2.2	dialogfinished	44
8.2.2.3	get_onlysingerun	45
8.2.2.4	get_security_angle	45
8.2.2.5	get_security_percentage	45
8.2.2.6	get_steps_to_go	46
8.2.2.7	get_steps_wo_measurement	46
8.2.2.8	ret_singerun	46
8.2.2.9	ret_understood	46
8.2.2.10	ret_wantnewscan	47
8.3	CoincCounter Class Reference	47
8.3.1	Detailed Description	48
8.4	ConfigDinspect Class Reference	48
8.4.1	Detailed Description	49
8.4.2	Member Function Documentation	49
8.4.2.1	LoadConfig()	49
8.4.2.2	SaveConfig()	50
8.5	Dinspect Class Reference	50
8.5.1	Detailed Description	60
8.5.2	Member Function Documentation	60
8.5.2.1	ActionAfterEdgeDetection()	60
8.5.2.2	CalibrationDialogFinished	60
8.5.2.3	end_scan	61
8.5.2.4	get_detector_data	61

8.5.2.5	GetFrequencyFromSNSPDcounter	61
8.5.2.6	GetPowerFromPM100D	61
8.5.2.7	getSpectrometerSubversion()	62
8.5.2.8	getSpectrometerType()	62
8.5.2.9	GraphSelectionChanged	62
8.5.2.10	initActionsConnections()	62
8.5.2.11	motor_is_at_starting_position	63
8.5.2.12	motorfoundfallingedge	63
8.5.2.13	motorfoundinitedge	63
8.5.2.14	motorfoundrisingedge	63
8.5.2.15	on_DetectorComboBox_activated	64
8.5.2.16	plot_savefile	64
8.5.2.17	recalc_with_dynamic_stepsize()	64
8.5.2.18	save()	65
8.5.2.19	saveAutomatically	65
8.5.2.20	SaveGraph	65
8.5.2.21	ScanRangeMinPositioning	65
8.5.2.22	setPM100D_AverageCount	66
8.5.2.23	setupGraph()	66
8.5.2.24	showActivityInfo	66
8.5.2.25	showCurrentMaxInfo	66
8.5.2.26	showCurrentWLInfo	67
8.5.2.27	showDataInfo	67
8.5.2.28	showPositionInfo	67
8.5.2.29	showStatusMessage	68
8.5.2.30	SNSPD_update_channel	68
8.5.2.31	SNSPD_update_coinc_channel	68
8.5.2.32	start_detectors()	68
8.5.2.33	TTM_IP_changed	69
8.5.2.34	TTM_Port_changed	69

8.5.2.35	UDPCOMsetdetector	69
8.5.2.36	UDPCOMsetscanrange	69
8.5.2.37	UDPCOMsetttmchannel	70
8.5.2.38	UDPCOMsetttmcoincidencechannel	70
8.5.2.39	update_spectrometer_info	70
8.5.2.40	updateIntegrationtime	71
8.6	readPM100Dlin.dinspectPM100D Class Reference	71
8.7	readPM100Dwin.dinspectPM100D Class Reference	71
8.8	ManualMotorControlDialog Class Reference	72
8.8.1	Detailed Description	74
8.8.2	Member Function Documentation	74
8.8.2.1	Connect_to_motor()	74
8.9	Motor Class Reference	75
8.9.1	Detailed Description	80
8.9.2	Member Function Documentation	80
8.9.2.1	command_microstep	80
8.9.2.2	command_nextedge	80
8.9.2.3	command_roughly_to_wl	81
8.9.2.4	command_scanmove	81
8.9.2.5	command_singlestep	81
8.9.2.6	command_step	81
8.9.2.7	command_to_start_pos	83
8.9.2.8	CreatePositionLabel()	83
8.9.2.9	edgewasfound	83
8.9.2.10	fallingedge	84
8.9.2.11	get_rough_wl	84
8.9.2.12	getdsettings	84
8.9.2.13	getSpectrometerSettings	85
8.9.2.14	gotonextedge	85
8.9.2.15	handleError	85

8.9.2.16	<code>isedge()</code>	86
8.9.2.17	<code>isfallingedge()</code>	86
8.9.2.18	<code>isopen</code>	86
8.9.2.19	<code>isrisingedge()</code>	86
8.9.2.20	<code>motorstatusmessage</code>	86
8.9.2.21	<code>risingedge</code>	87
8.9.2.22	<code>sendsignalfallingedge detected</code>	87
8.9.2.23	<code>sendsignalarisingedge detected</code>	87
8.9.2.24	<code>sensordata()</code>	88
8.9.2.25	<code>showStatusMessage</code>	88
8.9.2.26	<code>skip_edges</code>	89
8.9.2.27	<code>skip_edges_rough</code>	89
8.9.2.28	<code>spectrometer_info_updated</code>	89
8.9.2.29	<code>stop</code>	90
8.9.2.30	<code>updateAskmotor</code>	90
8.9.2.31	<code>updateIsFirstStep</code>	90
8.9.2.32	<code>updateScansIsRunning</code>	91
8.9.2.33	<code>write</code>	91
8.9.3	Member Data Documentation	91
8.9.3.1	<code>gotonextedgetimer</code>	91
8.9.3.2	<code>hometimer</code>	92
8.9.3.3	<code>roughwltimer</code>	92
8.9.3.4	<code>startpostimer</code>	92
8.10	MotorSettingsDialog::MotorSettings Struct Reference	92
8.10.1	Detailed Description	93
8.11	MotorSettingsDialog Class Reference	93
8.11.1	Detailed Description	95
8.11.2	Member Function Documentation	95
8.11.2.1	<code>apply</code>	95
8.11.3	Member Data Documentation	95

8.11.3.1	askmotor	95
8.12	PM100D Class Reference	96
8.12.1	Detailed Description	97
8.12.2	Member Function Documentation	97
8.12.2.1	getAverageCount()	97
8.12.2.2	getPower()	97
8.12.2.3	pyprint	97
8.12.2.4	setAverageCount()	98
8.12.2.5	setSerialNumber()	98
8.13	SNSPDcounter Class Reference	99
8.13.1	Detailed Description	99
8.14	Spectrometer Struct Reference	100
8.14.1	Detailed Description	102
8.15	spectrometeroptions Class Reference	102
8.15.1	Detailed Description	104
8.15.2	Member Function Documentation	104
8.15.2.1	initVars()	104
8.15.2.2	read_bp_data	104
8.15.2.3	read_ce_data	105
8.15.2.4	read_config	105
8.15.2.5	read_stepsinbp_data	105
8.15.2.6	read_wavelengthAtStep	106
8.15.2.7	return_single_spectrometer()	106
8.15.2.8	returnType()	106
8.15.2.9	write_config	107
8.16	tunablelaser Class Reference	107
8.16.1	Detailed Description	109
8.16.2	Member Function Documentation	109
8.16.2.1	configdir_changed	109
8.16.2.2	configfile_changed	110

8.16.2.3	processlasermg()	110
8.16.2.4	read()	110
8.16.2.5	setup()	111
8.16.2.6	setwl()	111
8.16.2.7	write()	111
8.17	UDPlistener Class Reference	111
8.17.1	Detailed Description	113
8.17.2	Constructor & Destructor Documentation	113
8.17.2.1	UDPlistener()	113
8.17.3	Member Function Documentation	114
8.17.3.1	bind	114
8.17.3.2	processCommands	114
8.17.3.3	setDetector	114
8.17.3.4	setScanrange	115
8.17.3.5	setTTMChannel	115
8.17.3.6	setTTMCoincidenceChannel	115
9	File Documentation	117
9.1	aboutdialog.h File Reference	117
9.2	calibrationdialog.h File Reference	118
9.3	coinccounter.h File Reference	119
9.4	configinspect.h File Reference	119
9.5	debug.h File Reference	120
9.5.1	Detailed Description	121
9.6	defines.h File Reference	121
9.6.1	Detailed Description	122
9.6.2	Macro Definition Documentation	122
9.6.2.1	ALLOW_UNCALIBRATED_MEASUREMENT	123
9.6.2.2	CALIBRATE_APPROXIMATELY_WITH_ARTIFICIAL_DATA	123
9.6.2.3	CALIBRATE_WITH_EDGE_DETECTION	123
9.6.2.4	CALIBRATION_RUNS_NEEDED	123

9.6.2.5	CALIBRATION_RUNS_NEEDED_ED	123
9.6.2.6	CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_SUBVER← SION	123
9.6.2.7	CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_TYPE . . .	123
9.6.2.8	CURRENT_SAVE_STRUCTURE_VERSION	124
9.6.2.9	DEBUG	124
9.6.2.10	DEBUGERROR	124
9.6.2.11	DEBUGINFO	124
9.6.2.12	DEBUGWARNING	124
9.6.2.13	DEGTORAD	124
9.6.2.14	EPS	125
9.6.2.15	PI	125
9.6.2.16	RADTODEG	125
9.7	dinspect.h File Reference	125
9.8	dmath.h File Reference	126
9.9	helper.h File Reference	128
9.10	manualmotorcontroldialog.h File Reference	130
9.11	motor.h File Reference	130
9.12	pm100d.h File Reference	131
9.13	snspsdcounter.h File Reference	132
9.14	spectrometeroptions.h File Reference	133
9.15	tunablelaser.h File Reference	134
9.16	udplistener.h File Reference	135
9.17	version.h File Reference	136
9.17.1	Detailed Description	137

Chapter 1

Dinspect

Dinspect, the **D**etector **i**ndependent **s**pectrometer control tool, is an application with graphical user interface to operate PCBMotor-based spectrometers. Its functionality will be described on the following pages

1.1 Installation

Dinspect does not need to be installed - just extract the archive to a folder and run it.

1.1.1 Step 1: Prerequisites

The following things are needed to run dinspect:

- Python 2.7
 1. with Pierre Cladés package ThorlabsPM100 (from <http://pythonhosted.org/ThorlabsPM100/>)
 2. Windows: the pyvisa package
- Windows: NI-VISA
- Windows: The Microsoft Visual C++ Redistributable Packages for Visual Studio 2013

1.2 Building

1.2.1 Windows

You will need:

- Microsoft Visual Studio (tested version: 2013)
- Qt (version 5.6)
- Python 2.7

To build, you can either:

- Run qmake followed by nmake in the source directory.
- Or open the qmake project file in QtCreator and build it in there.

After building dinspect, you need to copy the .py files, the LibTTM.dll, and the specconf.d folder to the build directory.

1.2.2 Linux

You will need:

- GCC (tested versions: 5 and 6) or LLVM/clang (tested versions: 3.6-3.9)
- Qt (tested versions: 5.6 to 5.8)
- Python 2.7

To build, you can either:

- Run qmake followed by make in the source directory.
- Or open the qmake project file in QtCreator and build it in there.

After building dinspect, you need to copy the .py files and the specconf.d folder to the build directory.

1.3 Usage

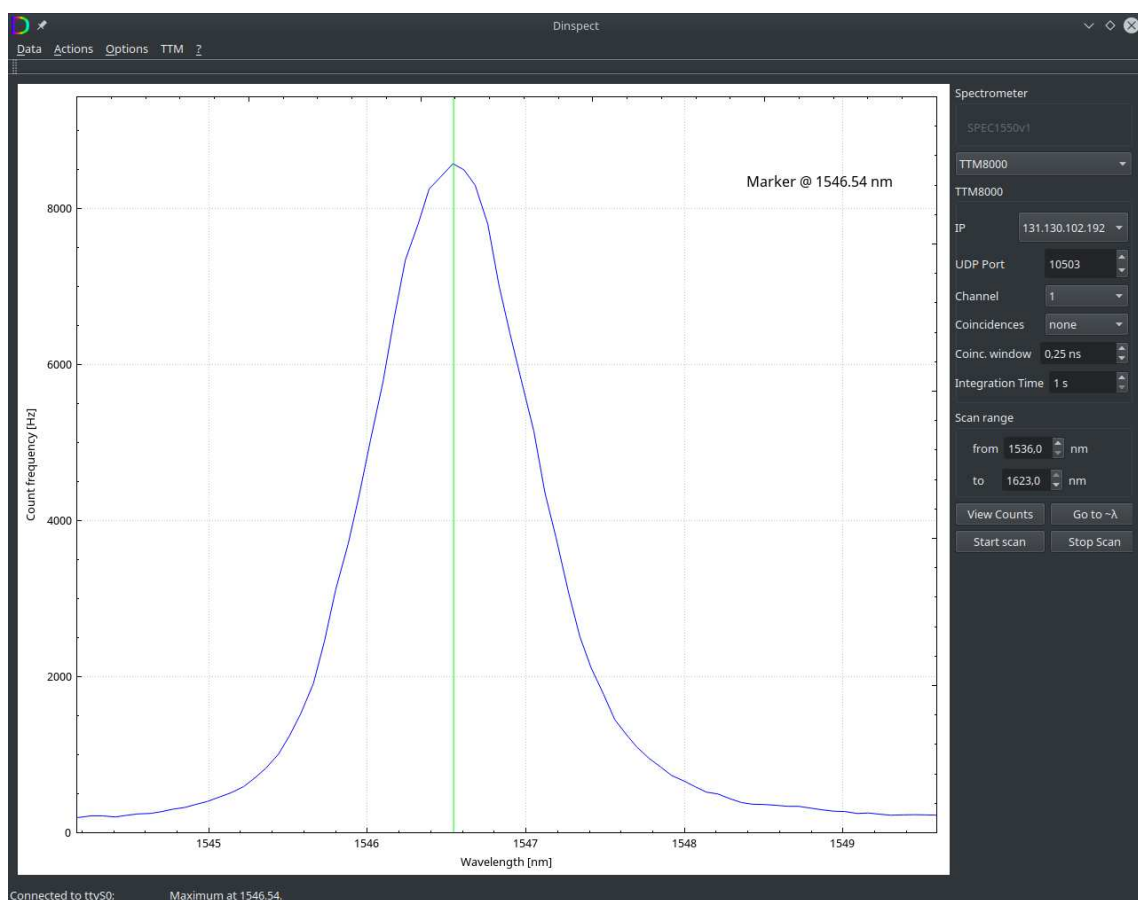


Figure 1.1 The main window.

The available controls are:

- **The detector combobox:** The detector can be set here.
- **Scanrange:** Specifies at which wavelengths the measurement is supposed to start and stop.
- **View counts:** When the TTM8000 was selected as a detector, opens a windows that displays the event count rate.
- **Go to $\sim \lambda$:** Opens a dialog to control the PCBMotor, described below.
- **Start scan:** Starts a measurement.
- **Stop scan:** Stops a measurement.

When the *TTM8000* was selected as a detector, the following options can be set:

- **IP:** The IP (of the computer running dinspect) at which dinspect listens for packages from the TTM8000.
- **UDP Port:** The port to listen to for packages from the TTM8000.
- **Channel:** The channel of the TTM8000 of which the event count rate will be aquired.
- **Coincidences:** If this is set to 'None', the single event count rate of the channel specified with the combobox 'Channel' will be aquired.
If it is set to a numeric value, this value and the channel 'Channel' will specify between which channels the coincidence rate will be aquired.
- **Coinc. window:** Sets the time window specifying which consecutive events will be considered to be a coincidence.
- **Integration time:** Sets the measurement time for each datapoint.

The graph can be controlled in the following way:

- A single click on one of the axes will select that axis.
- When an axis is selected, click and drag with the left mousebutton will shift the range of the corresponding axis.
- When an axis is selected, turning the mousewheel will zoom the range of the corresponding axis.
- A doubleclick will add a vertical line, positioned at the cursor, and a label showing at which wavelength this marker was positioned.

Spectra and graphs can be saved using the **Data** menu entry.



Figure 1.2 Managing the delay offsets per channel for counting coincidences with the TTM8000.

Using TTM->Edit Channel Offset, the timing offsets in between the channels of the TTM8000 can be set in nanoseconds. This is only necessary for coincidence - measurements.

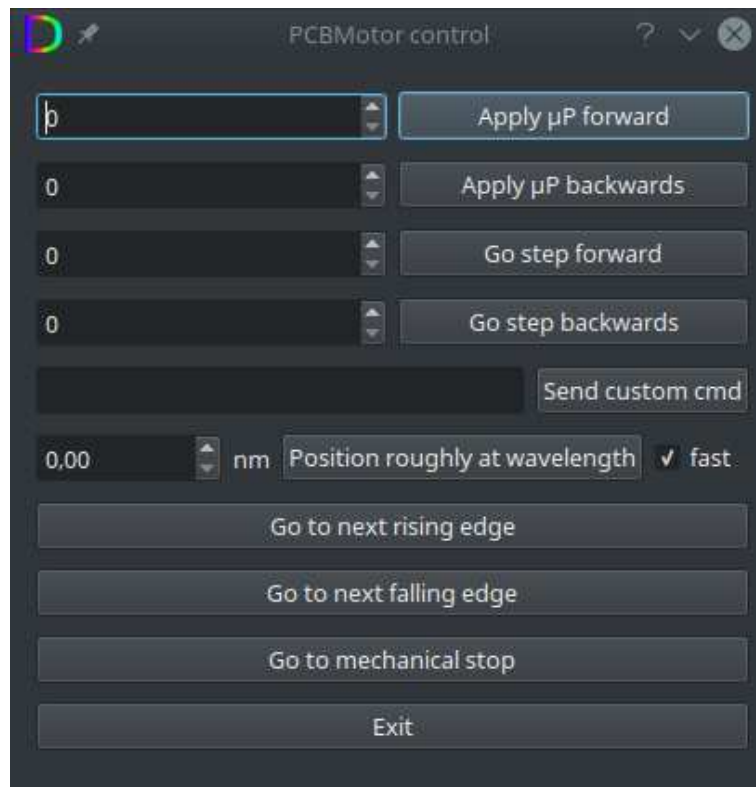


Figure 1.3 Controlling the PCBMotor from within dinspect.

Clicking on the button **Go to $\sim \lambda$** will open a dialog from which you can control the PCBMotor manually, including the possibility to position it close to, but not exactly at, a specific wavelength.

The controls are:

- **Apply μp forward:** Sends a command to the PCBMotor to apply a number of micropulses in direction of increasing wavelength.
- **Apply μp backward:** Sends a command to the PCBMotor to apply a number of micropulses in direction of decreasing wavelength.
- **Go step forward:** Sends a command to the PCBMotor to go a number of steps in direction of increasing wavelength.
- **Go step backward:** Sends a command to the PCBMotor to go a number of steps in direction of decreasing wavelength.
- **Send custom cmd:** Sends a custom command to the PCBMotor.
- **Position roughly at wavelength:** Rotates the PCBMotor such that the spectrometer will select a wavelength more or less close the specified wavelength.
- **Go to next rising edge:** Sends a command to the PCBMotor which then positions itself at the next rising edge in the PCBMotor encoder wheel sensor signal.
- **Go to next falling edge:** Sends a command to the PCBMotor which then positions itself at the next falling edge in the PCBMotor encoder wheel sensor signal.
- **Go to mechanical stop:** Sends a command to the PCBMotor to rotate to the mechanical stop.

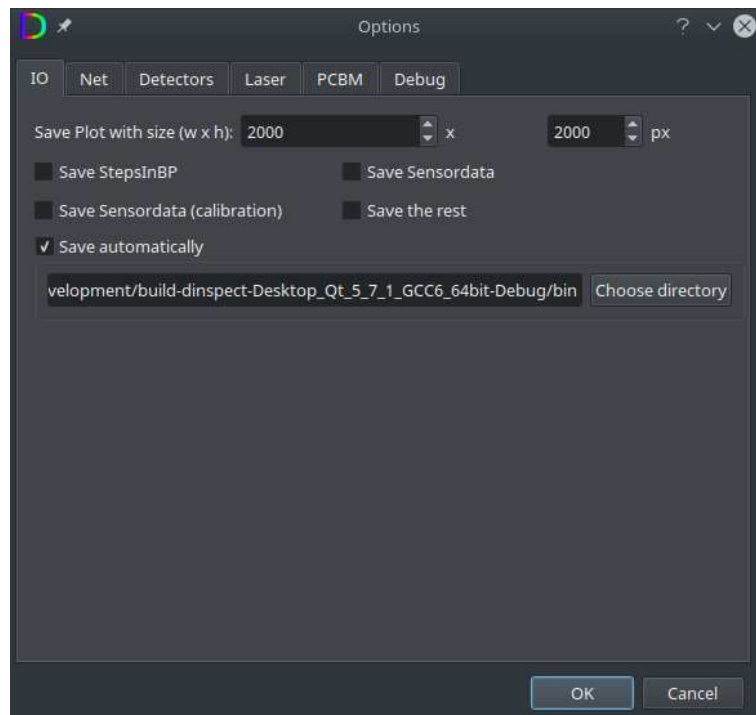


Figure 1.4 The input/output options dialog.

In the IO tab of the options dialog, the following settings can be adjusted:

- **Save plot width size (w x h):** When saving the plot of a spectrum, these two numbers will be the width and height of the image in pixels.
- **Save StepsInBP:** Saves the number of steps in each bandpass to an additional file.
- **Save Sensordata:** Saves the encoder wheel sensor data at each step to an additional file (only true for measurement scans).
- **Save Sensordata (calibration):** Saves the encoder wheel sensor data at each step to an additional file (only true for calibration scans).
- **Save the rest:** Saves several quantities useful for inspecting the behaviour of the spectrometer to an additional file.
- **Save automatically:** Saves the spectrum automatically when the measurement is over.
- **Choose directory:** Opens a dialog to choose the directory in which the spectra will be saved when automatic saving is enabled.

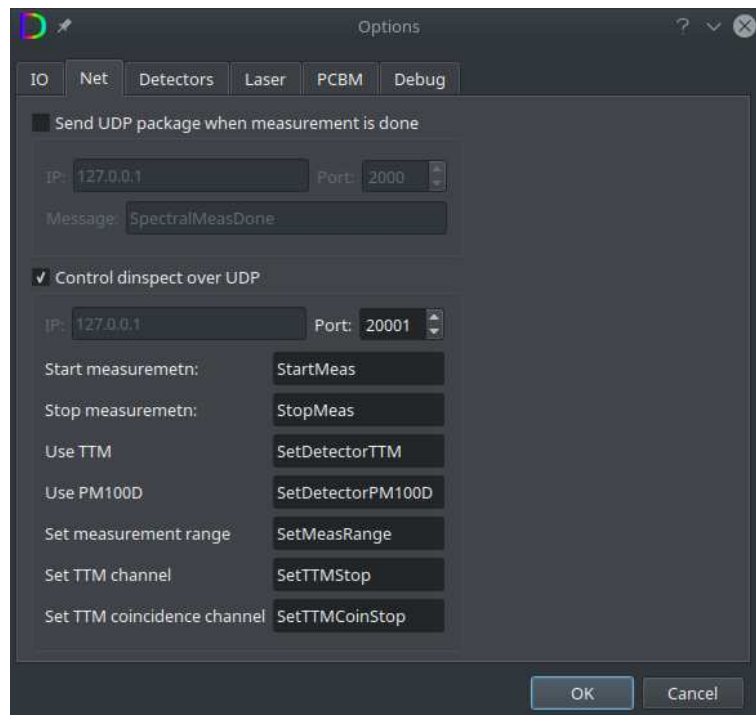


Figure 1.5 The network options dialog.

In the Net tab of the options dialog, the following settings can be adjusted:

- **Send UDP package when measurement is done:** If enabled, dinspect will send a UDP package containing a custom message to a specific IP:Port whenever a measurement is completed.
 1. **IP:** The IP to send the message to.
 2. **Port:** The Port to send the message to.
 3. **Message:** The message to send.
- **Control dinspect over UDP:** If enabled, dinspect will listen for packages arriving on a specifig UDP port. These packages will be checked for customizable commands which trigger various actions (see below). Potential arguments have to be provided in curved brackets. If no argument is required, no brackets shall be provided.
 1. **Port:** The port which dinspect will listen to.
 2. **Start measurement:** If this command is received, a measurement will be initiated (if none is running currently). Required arguments: None.
 3. **Stop measurement:** If this command is received, a currently running measurement will be stopped. Required arguments: None.
 4. **Use TTM:** If this command is received, the detector to use will be set to 'TTM8000'. Required arguments: None.
 5. **Use PM100D:** If this command is received, the detector to use will be set to 'Thorlabs PM100D'. Required arguments: None.
 6. **Set measurement range:** If this command is received, the lower and upper measurement range will be set accordingly. Required arguments: 2 (double, double). Example: *SetMeasRange(1545.5, 1552.7)*
 7. **Set TTM channel:** If this command is received, the channel for single event measurements of the TTM8000 will be set accordingly. Required arguments: 1 (int). Example: *SetTTMStop(3)*
 8. **Set TTM coincidence channel:** If this command is received, the channel of the TTM8000 will be set accordingly. Required arguments: 1 (int). Example: *SetTTMCoinStop(5)*

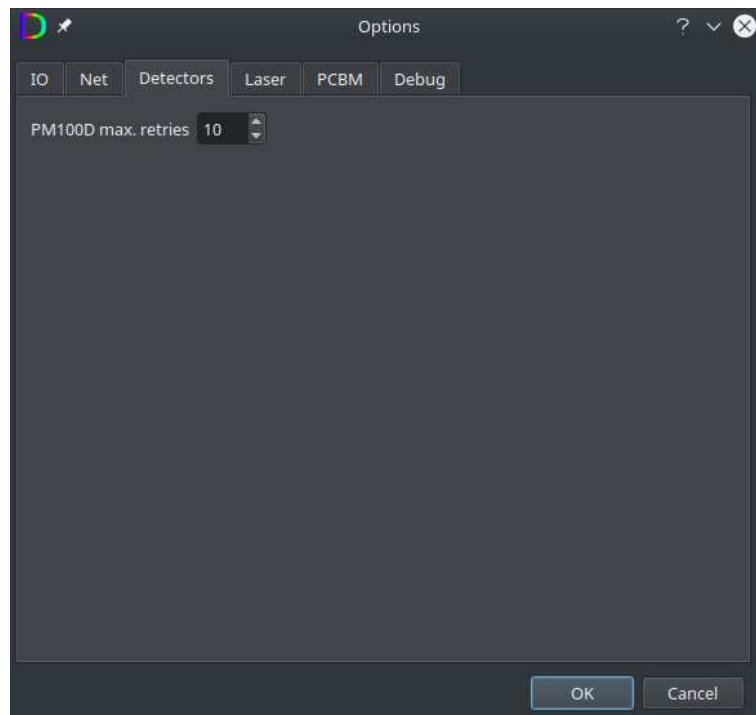


Figure 1.6 The detector options dialog.

In the Detectors tab of the options dialog, the following settings can be adjusted:

- **PM100D max retries:** Specifies how many times the data acquisition from the Thorlabs PM100D will be repeated in case of an error.

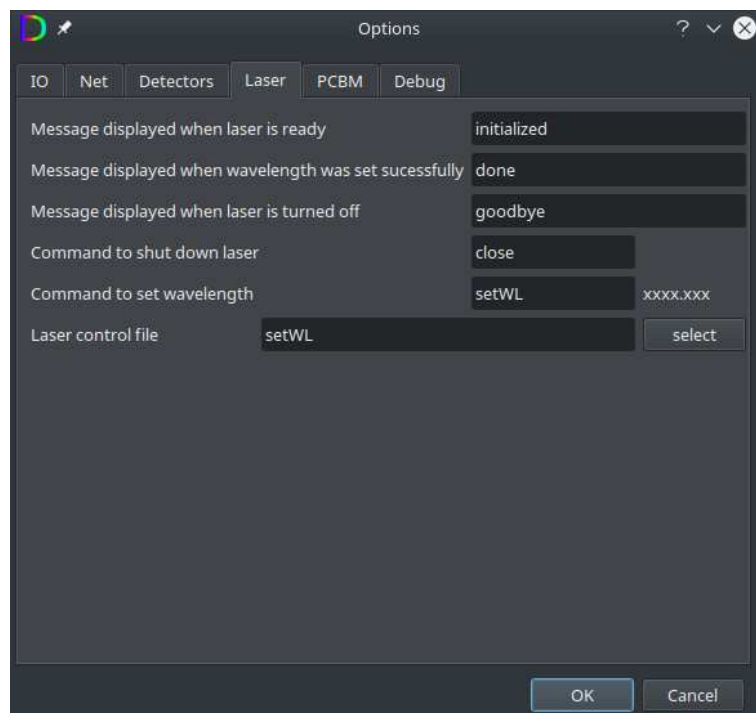


Figure 1.7 The options dialog to setup the tunable laser support.

In the Laser tab of the options dialog, the following settings can be adjusted:

- **Message displayed when laser is ready:** When this text is found in the laser control file, dinspect will know that the laser is ready to operate.
- **Message displayed when wavelength was set successfully:** When this text is found in the laser control file, dinspect will know that the wavelength change was successful.
- **Message displayed when laser is turned off:** When this text is found in the laser control file, dinspect will know that the laser has been turned off.
- **Command to shut down laser:** This is the text that will be written to the laser control file to shut down the laser.
- **Command to set wavelength:** This is the text that will be written to the laser control file to start a wavelength change. The wavelength will be appended directly (example: SetWL1552.123)
- **Laser control file:** Path to the laser control file.
- **Select** Opens a file dialog to select the laser control file.

1.4 Fit spectra

This python3 script shows one way how to process the spectra gathered with dinspect. It fits all spectra in a folder to gaussian and sech^2 distributions and outputs the fit parameters to separate files for further usage.

```
#!/usr/bin/python3
#
# Fit every *.spec file in the current directory and write the fit-parameters together with
# some other useful information into two CSV-files (one for idler, one for signal)
#
# This script uses the filename of the recorded spectra to gather information
# about the measurement.
# Things that need to be contained in the filename are:
# either the keyword 'signal' or 'idler'
# Xnm for the pump wavelength
# Xps for the pump pulse width
# _TX for the temperature
#
# Optional parameters in the filename:
# seedXnm for the seed wavelength
# xps for pump pulsewidth
# xMHz for the pump pulse repetition rate
#
# where X is a decimal number in all cases (use . as the decimal point)
#
# an example:
# '20161109_Hpump_55mW_773.1nm_2.2ps_1.6nm_T24_seed1556.701nm_signal.spec'
#
# The output will be sorted according to a column one can specify with the parameter 'sortidx' explained
# later in this file.
#
# For each file fitted, a pdf will be created showing fit and data together. This way one can check if the
# fits are reasonable.

#Correct for a (constant) wavelength-offset in the measurement data?
useoffset=False
offset=0
#Do gaussian fit?
usegaussian=False
#Do sech^2 fit?
usesech=True

#The gaussian distribution, to which the data is fitted
def gaussian(B, x):
    'Gaussian'
    #B[0] offset
    #B[1] Standard deviation
    #B[2] mean
    #B[3] scaling
    return B[0] + (B[3]/(B[1]*numpy.sqrt(2*numpy.pi)))*numpy.exp(-(numpy.power((x-B[2]),2)/(2*
numpy.power(B[1],2))))
```

```

def gaussian_fwhm(B):
    return 2*numpy.sqrt(2*numpy.log(2))*B[1]

def gaussian_fwhm_err(B,Berr):
    return 2*numpy.sqrt(2*numpy.log(2))*Berr[1]

#The sech^2 distribution, to which the data is fitted
def sechsq(B,x):
    'Secant hyperbolic squared distribution'
    #B[0] offset
    #B[1] 'thickness'
    #B[2] mean
    #B[3] scaling
    denom = numpy.cosh(B[1]*(x-B[2]))
    return B[0] + B[3]/(denom*denom)

def sechsq_fwhm(B):
    return 2*numpy.arccosh(numpy.sqrt(2))/B[1]

def sechsq_fwhm_err(B,Berr):
    return numpy.absolute(2*numpy.arccosh(numpy.sqrt(2))*Berr[1]/(B[1]*B[1]))

#function which returns one element of a list with a certain fixed index
#used to sort the produced datafiles
def getlistelem(item):
    return item[sortidx]

#sortidx holds the column redarding to which the output-files shall be sorted
#0: Sort by temperature
#1: Sort by pump pulsewidth
#2: Sort by pumpWL
#3: Sort by seedWL
sortidx=0

import os, array, numpy, scipy, scipy.odr, re
import matplotlib.pyplot as plt
import sys

#input data will be stored in these arrays
inwldata = array.array('d')
infreqdata = array.array('d')

#when reading a file, its contents will be stored here
readtmp = []

#index of first element not equal to zero
firstdataidx = 0
#index of max frequency
maxidx = 0

#gaussian parameters - estimate to start fitting
GParEst = [0, 0, 0, 0]
#gaussian parameters - returned by fitting function
GPar = [0, 0, 0, 0]
#standard errors of gaussian parameters
GParStd = [0, 0, 0, 0]

#sech2 parameters - estimate to start fitting
SParEst = [0, 0, 0, 0]
#sech2 parameters - returned by fitting function
SPar = [0, 0, 0, 0]
#sech2 errors of gaussian parameters
SParStd = [0, 0, 0, 0]

#id' of the current file when fitting multiple files at once
fileidx = 0

#filenames to write to
save_sig_gauss_fname = "python_fitparameters_gaussian_signal.csv"
save_idl_gauss_fname = "python_fitparameters_gaussian_idler.csv"
save_sig_sech_fname = "python_fitparameters_sech_signal.csv"
save_idl_sech_fname = "python_fitparameters_sech_idler.csv"
save_jsi_gauss_fname = "python_fitparameters_gaussian_jsi.csv"
save_jsi_sech_fname = "python_fitparameters_sech_jsi.csv"
#open these files and write header
if usegaussian:
    savefile_sig_gauss = open(save_sig_gauss_fname, "w")
    savefile_idl_gauss = open(save_idl_gauss_fname, "w")
    savefile_sig_gauss.write("Temperature[°C];PumpPulseWidth[ps];pumpWL[nm];SeedWL[nm];WL_sig[nm];"+"\\
    "
    WL_sig+[nm];WL_sig-[nm];FWHM_sig[nm];FWHM_sigErr[nm];RepetitionRate[MHz];fname\\n")
    savefile_idl_gauss.write("Temperature[°C];PumpPulseWidth[ps];pumpWL[nm];SeedWL[nm];WL_idl[nm];"+"\\
    "
    WL_idl+[nm];WL_idl-[nm];FWHM_idl[nm];FWHM_idlErr[nm];RepetitionRate[MHz];fname\\n")
if usessech:

```

```

savefile_sig_sech = open(save_sig_sech_fname, "w")
savefile_idl_sech = open(save_idl_sech_fname, "w")
savefile_sig_sech.write("Temperature[°C];PumpPulseWidth[ps];pumpWL[nm];SeedWL[nm];WL_sig[nm];"+\
    "
    WL_sig+[nm];WL_sig-[nm];FWHM_sig[nm];FWHM_sigErr[nm];RepetitionRate[MHz];fname\n")
savefile_idl_sech.write("Temperature[°C];PumpPulseWidth[ps];pumpWL[nm];SeedWL[nm];WL_idl[nm];"+\
    "
    WL_idl+[nm];WL_idl-[nm];FWHM_idl[nm];FWHM_idlErr[nm];RepetitionRate[MHz];fname\n")
#are there jsi-files?
for file in os.listdir(os.getcwd()):
    if 'jsi' in file.casefold():
        usejsi=True
    else:
        usejsi=False
if usejsi:
    savefile_jsi_gauss = open(save_jsi_gauss_fname, "w")
    savefile_jsi_sech = open(save_jsi_sech_fname, "w")
    savefile_jsi_gauss.write("
    Temperature[°C];PumpPulseWidth[ps];pumpWL[nm];SeedWL[nm];WL_idl[nm];WL_idl+[nm];"+\
    "
    WL_idl-[nm];FWHM_idl[nm];FWHM_idlErr[nm];AveragingTime[s];RepetitionRate[MHz];fname\n")
    savefile_jsi_sech.write("
    Temperature[°C];PumpPulseWidth[ps];pumpWL[nm];SeedWL[nm];WL_idl[nm];WL_idl+[nm];"+\
    "
    WL_idl-[nm];FWHM_idl[nm];FWHM_idlErr[nm];AveragingTime[s];RepetitionRate[MHz];fname\n")

#iterate through all '.spec'-files in the current directory
for file in os.listdir(os.getcwd()):
    if file.endswith(".spec"):
        print("Opening file: ", file)
        filestream = open(file, "r")
        readtmp = filestream.readlines()
        #skip the first 5 lines - they do not contain data
        for i in range(5, len(readtmp)):
            inwldata.append(float(readtmp[i].split()[1]))
            infreqdata.append(float(readtmp[i].split()[2]))

        #find first index of arrays where frequency is not zero
        for i in range(0, len(infreqdata)):
            if infreqdata[i]!=0:
                firstdataidx=i
                break

        #remove leading 0's
        inwldata = inwldata[firstdataidx:]
        infreqdata = infreqdata[firstdataidx:]

        if useoffset:
            inwldata=inwldata-numpy.ones(len(inwldata))*offset

        #set initial values for fitting
        minfreq = numpy.min(infreqdata)
        maxfreq = numpy.max(infreqdata)
        maxidx = numpy.argmax(infreqdata)
        if usegaussian:
            GParEst[0] = minfreq #offset = minimum
            GParEst[3] = maxfreq# - minfreq #scaling = max-min
            GParEst[2] = inwldata[maxidx] #WL of maximum
            GParEst[1] = 0.3

        #use this for sech squared
        if usessech:
            SParEst[0] = minfreq #offset = minimum
            SParEst[3] = maxfreq - minfreq #scaling = max-
            SParEst[2] = inwldata[maxidx] #WL of maximum
            SParEst[1] = 1

        #ODR fitting
        xerr=numpy.ones(len(inwldata))
        yerr=numpy.ones(len(inwldata))
        wl_err_plus = 0.11
        wl_err_min = 0.013
        xerrscalar=0.0615
        yerrscalar=100
        xerr=xerr*xerrscalar
        yerr=yerr*yerrscalar
        xerrlow=xerr*wl_err_min/xerrscalar
        xerrup=xerr*wl_err_plus/xerrscalar

        ChiSquStr=''
        if usegaussian:
            odrfitmodelG = scipy.odr.Model(gaussian)
            odrdataG = scipy.odr.RealData(inwldata, infreqdata, sx=xerr, sy=yerr)
            odrG = scipy.odr.ODR(odrdataG, odrfitmodelG, GParEst, maxit=10000)
            odroutG = odrG.run()
            GPar = odroutG.beta

```

```

GParStd = odroutG.sd_beta
GChiSqu = odroutG.sum_square
GParCovMx = odroutG.cov_beta
GParCov = numpy.sqrt(numpy.diag(GParCovMx))
print("Gaussian:")
print("[offset, thickness, mean, scale]")
print("GPar: ", GPar)
print("GParStd: ", GParStd)
print("GParCov: ", GParCov)
print("GChiSqu: ", GChiSqu)
#FWHM
gFWHM = gaussian_fwhm(GPar)
gFWHMerr = gaussian_fwhm_err(GPar,GParCov)
ChiSquStr = ChiSquStr+'$\chi^2$(gauss)=%.2f$\n'%(GChiSqu)
if usesech:
    odrfitmodels = scipy.odr.Model(sechsqu)
    odrdataS = scipy.odr.RealData(inwldata, infreqdata, sx=xerr, sy=yerr)
    odrS = scipy.odr.ODR(odrdataS, odrfitmodels, SParEst, maxit=10000)
    odroutS = odrS.run()
    SPar = odroutS.beta
    SParStd = odroutS.sd_beta
    SChiSqu = odroutS.sum_square
    SParCovMx = odroutS.cov_beta
    SParCov = numpy.sqrt(numpy.diag(SParCovMx))
    print("Sech:")
    print("[offset, thickness, mean, scale]")
    print("SPar: ", SPar)
    print("SParStd: ", SParStd)
    print("SParCov: ", SParCov)
    print("SChiSqu: ", SChiSqu)
    #FWHM
    sFWHM = sechsq_fwhm(SPar)
    sFWHMerr = sechsq_fwhm_err(SPar,SParCov)
    ChiSquStr = ChiSquStr+'$\chi^2$(sech^2)=%.2f$\n'%(SChiSqu)

#Plot fit
plt.title('Fit of '+file)
plt.plot(inwldata,infreqdata,"bo",label="data",ms=2)
plt.errorbar(inwldata,infreqdata,yerr=yerr,xerr=[xerrlow,xerrup],fmt='o',ms=1)
if usegaussian:
    plt.plot(inwldata,gaussian(GPar,inwldata), "g--", lw=1,label="fit gaussian")
if usesech:
    plt.plot(inwldata,sechsqu(SPar,inwldata), "r--", lw=1,label="fit sech2")
plt.legend()
plt.xlabel('Wavelength [nm]')
plt.ylabel('Frequency [Hz]')
plt.grid()

if usesech:
    plt.annotate(r'sech^2$ fit:'+n'+r'$\chi^2$='+("{:.2f}".format(SChiSqu)+"n"+r'
FWHM='+r"({0:.2f})$pm${1:.2f})nm".format(sFWHM,sFWHMerr)+"n"+\
r'$\lambda_c$='+r"({0:.2f})$pm${1:.2f})$nm".format(SPar[2],SParCov[2]),
xy=(0.7, 0.64), xycoords='axes fraction',fontSize=8)
if usegaussian:
    plt.annotate(r'gaussian fit:'+n'+r'$\chi^2$='+("{:.2f}".format(GChiSqu)+"n"+r'
FWHM='+r"({0:.2f})$pm${1:.2f})nm".format(gFWHM,gFWHMerr)+"n"+\
r'$\lambda_c$='+r"({0:.2f})$pm${1:.2f})$nm".format(GPar[2],GParCov[2]),
xy=(0.7, 0.475), xycoords='axes fraction',fontSize=8)

save_figure_fname = os.path.join(os.getcwd(), "fitplot_"+file+".pdf")
plt.savefig(save_figure_fname, format="pdf")
plt.close('all')
plt.gcf().clear()

#Get parameters from filename

#example filename:
# '20161109_Hpump_55mW_773.1nm_2.2ps_T24_1.6nm_seed1556.701nm.spec'
tempStr = re.findall('_T[+-]?[d*]\d+|_T\d+',file)
#['_T24']
if(len(tempStr)>0):
    temp = float(tempStr[0].replace("_T",""))
else:
    temp = 0

tauStr = re.findall('[+-]?[d*]\d+ps|\d+ps',file)
#['2.2ps']
if (len(tauStr)>0):
    tau = float(tauStr[0].replace("ps",""))
else:
    tau = 0

pwlStr = re.findall('[+-]?[d*]\d+nm|\d+nm',file)
#['773.1nm', '1.6nm', '1556.701nm']
#Up to three wavelengths in filename: PumpWL, (delta)PumpWL, SeedWL
#Store all of them in an array:
pwlList = []

```

```

for i in range(0,len(pwlStr)):
    pwllist.append(float(pwlStr[i].replace("nm","")))

#is seedWL written in the filename?
seedWL=0

#RepetitionRate
RR=0
if "mhz" in file.casefold():
    RRStr = re.findall('[+-]?\d*\.\d+mhz|\d+mhz',file.casefold())
    #['76MHz']
    if (len(RRStr)>0):
        RR = float(RRStr[0].replace("mhz",""))
    else:
        RR = 0

if "seed" in file:
    SeedWLStr = re.findall('seed[+-]?\d*\.\d+nm|seed\d+nm',s)
    seedWL = float(SeedWLStr[0].replace("seed","").replace("nm",""))
    #remove seedWL from list
    for i in range(0,len(pwlStr)):
        if pwlStr[i]==seedWL:
            del pwlStr[i]

#get pumpWL out of Wavelength-List (should be the highest WL now that seed is removed)
pwl = 0
for i in range(0,len(pwlStr)):
    pwl = max(pwl,float(pwlStr[i].replace("nm","")))

if "jsi" in file.casefold():
    #averaging time
    avgtStr = re.findall('[+-]?\d*\.\d+s|\d+s',file)
    #['60ps']
    if (len(avgtStr)>0):
        avgt = float(avgtStr[0].replace("s",""))
    else:
        avgt = 0

#Output file structure
#-----0-----1-----2-----3-----4-----
#Temperature[°C];PumpPulseWidth[ps];SeedWL[nm];WL[nm];WL+[nm];
#--5-----6-----7-----8-----9-----
#WL-[nm];FWHM[nm];FWHMerr[nm];RepetitionRate[MHz];fname
if "signal" in file.casefold():
    if usegaussian:
        savefile_sig_gauss.write(str(temp)+" "+
                                str(tau)+" "+
                                str(pwl)+" "+
                                str(seedWL)+" "+
                                str(GPar[2])+" "+
                                str(wl_err_plus)+" "+
                                str(wl_err_min)+" "+
                                str(gFWHM)+" "+
                                str(gFWHMerr)+" "+
                                str(RR)+" "+
                                file+
                                "\n")
    if usesech:
        savefile_sig_sech.write(str(temp)+" "+
                                str(tau)+" "+
                                str(pwl)+" "+
                                str(seedWL)+" "+
                                str(SPar[2])+" "+
                                str(wl_err_plus)+" "+
                                str(wl_err_min)+" "+
                                str(sFWHM)+" "+
                                str(sFWHMerr)+" "+
                                str(RR)+" "+
                                file+
                                "\n")

elif "idler" in file.casefold():
    if usegaussian:
        savefile_idl_gauss.write(str(temp)+" "+
                                str(tau)+" "+
                                str(pwl)+" "+
                                str(seedWL)+" "+
                                str(GPar[2])+" "+
                                str(wl_err_plus)+" "+
                                str(wl_err_min)+" "+
                                str(gFWHM)+" "+
                                str(gFWHMerr)+" "+
                                str(RR)+" "+
                                file+
                                "\n")
    if usesech:

```

```

        savefile_idl_sech.write(str(temp)+";"+"
                                str(tau)+";"+"
                                str(pwl)+";"+"
                                str(seedWL)+";"+"
                                str(SPar[2])+";"+"
                                str(wl_err_plus)+";"+"
                                str(wl_err_min)+";"+"
                                str(sFWHM)+";"+"
                                str(sFWHMerr)+";"+"
                                str(RR)+";"+"
                                file+
                                "\n")

#Output file structure
#-----0-----1-----2-----3-----4-----
#Temperature[°C];PumpPulseWidth[ps];SeedWL[nm];WL[nm];WL+[nm];
#--5-----6-----7-----8-----9-----10
#WL-[nm];FWHM[nm];FWHMerr[nm];AveragingTime[s];RepetitionRate[MHz];fname
elif "jsi" in file.casefold():
    if usegaussian:
        savefile_jsi_gauss.write(str(temp)+";"+"
                                str(tau)+";"+"
                                str(pwl)+";"+"
                                str(seedWL)+";"+"
                                str(GPar[2])+";"+"
                                str(wl_err_plus)+";"+"
                                str(wl_err_min)+";"+"
                                str(gFWHM)+";"+"
                                str(gFWHMerr)+";"+"
                                str(avgt)+";"+"
                                str(RR)+";"+"
                                file+
                                "\n")

    if usessech:
        print('writing')
        savefile_jsi_sech.write(str(temp)+";"+"
                                str(tau)+";"+"
                                str(pwl)+";"+"
                                str(seedWL)+";"+"
                                str(SPar[2])+";"+"
                                str(wl_err_plus)+";"+"
                                str(wl_err_min)+";"+"
                                str(sFWHM)+";"+"
                                str(sFWHMerr)+";"+"
                                str(avgt)+";"+"
                                str(RR)+";"+"
                                file+
                                "\n")

    else:
        print("ERROR: Please add either signal or idler to filename")
        break

#prepare for next file
print("\n")
filestream.close()
#del odrG
del odrS
#del odrdataG
del odrdataS
#del odrfitmodelG
del odrfitmodelS
inwldata = array.array('d')
infreqdata = array.array('d')
fileidx = fileidx + 1

#savefile.close()
if usegaussian:
    savefile_sig_gauss.close()
    savefile_idl_gauss.close()
if usessech:
    savefile_sig_sech.close()
    savefile_idl_sech.close()
if usejsi:
    savefile_jsi_gauss.close()
    savefile_jsi_sech.close()

#open savefiles and sort them
if usegaussian:
    with open(save_sig_gauss_fname, "r+") as sfile:
        print('saving gauss/signal')
        readtmp = sfile.readlines()
        data = [[0 for x in range(0,len(readtmp[0].split(';')))] for y in range(1,len(readtmp))]
        header=readtmp[0]

    for i in range(1,len(readtmp)):
        for j in range(0,len(readtmp[0].split(';'))):
            data[i-1][j] = readtmp[i].split(';')[j]

```

```

        #sort data
        data=sorted(data, key=getlistelement)

        sfile.seek(0)

        sfile.write(header)

        #write data to file
        for i in range(0,len(data)):#rows
            sfile.write(":".join(data[i]))

        sfile.close()

with open(save_idl_gauss_fname, "r+") as sfile:
    print('saving gauss/idler')
    readtmp = sfile.readlines()
    data = [[0 for x in range(0,len(readtmp[0].split(';')))] for y in range(1,len(readtmp))]
    header=readtmp[0]

    for i in range(1,len(readtmp)):
        for j in range(0,len(readtmp[0].split(';'))):
            data[i-1][j] = readtmp[i].split(';')[j]

    #sort data
    data=sorted(data, key=getlistelement)

    #prepare for rewriting the file
    sfile.seek(0)
    sfile.write(header)

    #write data to file
    for i in range(0,len(data)):#rows
        sfile.write(":".join(data[i]))

    sfile.close()

with open(save_sig_sech_fname, "r+") as sfile:
    print('saving sech/signal')
    readtmp = sfile.readlines()
    data = [[0 for x in range(0,len(readtmp[0].split(';')))] for y in range(1,len(readtmp))]
    header=readtmp[0]

    for i in range(1,len(readtmp)):
        for j in range(0,len(readtmp[0].split(';'))):
            data[i-1][j] = readtmp[i].split(';')[j]

    #sort data
    data=sorted(data, key=getlistelement)

    sfile.seek(0)

    sfile.write(header)

    #write data to file
    for i in range(0,len(data)):#rows
        sfile.write(":".join(data[i]))

    sfile.close()

with open(save_idl_sech_fname, "r+") as sfile:
    print('saving sech/idler')
    readtmp = sfile.readlines()
    data = [[0 for x in range(0,len(readtmp[0].split(';')))] for y in range(1,len(readtmp))]
    header=readtmp[0]

    for i in range(1,len(readtmp)):
        for j in range(0,len(readtmp[0].split(';'))):
            data[i-1][j] = readtmp[i].split(';')[j]

    #sort data
    data=sorted(data, key=getlistelement)

    #prepare for rewriting the file
    sfile.seek(0)
    sfile.write(header)

    #write data to file
    for i in range(0,len(data)):#rows
        sfile.write(":".join(data[i]))

    sfile.close()

```

Chapter 2

Bug List

Class [AboutDialog](#)

There are no known bugs.

Class [CalibrationDialog](#)

There are no known bugs.

Class [CoincCounter](#)

There are no known bugs.

Class [ConfigDinspect](#)

There are no known bugs.

File [debug.h](#)

Printing to console does not work on Windows. Workaround: Redirect stderr to stdout and redirect stdout to a file.

File [defines.h](#)

There are no known bugs.

Class [Dinspect](#)

Randomly, reading data from the TTM socket fails. It seems that the TTMDDataPacket_t is not available. This has only been observed on windows.

Namespace [dmath](#)

There are no known bugs.

Namespace [helper](#)

There are no known bugs.

Class [ManualMotorControlDialog](#)

There are no known bugs.

Class [Motor](#)

There are no known bugs.

Class [MotorSettingsDialog](#)

There are no known bugs.

Class [PM100D](#)

On Windows, getPower sometimes fails when using visa. Maybe a timing issue.

Class [SNSPDcounter](#)

There are no known bugs.

Class [Spectrometer](#)

There are no known bugs.

Class [spectrometeroptions](#)

There are no known bugs.

Class [tunablelaser](#)

There are no known bugs.

Class [UDPlistener](#)

There are no known bugs

File [version.h](#)

There are no known bugs.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

dmath		25
helper	Small functions to display messages	38

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

readPM100Dlin.dinspectPM100D	71
readPM100Dwin.dinspectPM100D	71
MotorSettingsDialog::MotorSettings	92
QDialog	
AboutDialog	41
CalibrationDialog	42
ConfigDinspect	48
ManualMotorControlDialog	72
MotorSettingsDialog	93
QMainWindow	
Dinspect	50
QObject	
CoincCounter	47
Motor	75
PM100D	96
SNSPDcounter	99
spectrometeroptions	102
tunablelaser	107
UDPlistener	111
Spectrometer	100

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AboutDialog	
Just used to show the about-dialog	41
CalibrationDialog	
Handles userinput to configure the calibration-procedure	42
CoincCounter	
Aquires coincidence frequencies from a TM8000	47
ConfigDinspect	
Handles user input to configure the behaviour of dinspect	48
Dinspect	
Main class of this application	50
readPM100Dlin.dinspectPM100D	71
readPM100Dwin.dinspectPM100D	71
ManualMotorControlDialog	
Offers the ability to manually operate the PCBMotor	72
Motor	
Operates the PCB-motor	75
MotorSettingsDialog::MotorSettings	
The MotorSettings struct contains contains the information to successfully establish a serial connection to the PCBMotor	92
MotorSettingsDialog	
Handles user-input to configure the connection to the PCB-motor	93
PM100D	
Operates the Thorlabs PM100D powermeter	96
SNSPDcounter	
Aquires single event frequencies from a TM8000	99
Spectrometer	
The Spectrometer struct holds spectrometer-specific variables	100
spectrometeroptions	
Handles spectrometer-configurationfiles	102
tunablelaser	
Able to communicate with a tunable laser using a file	107
UDPlistener	
Used to control dinspect with UDP packages	111

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

aboutdialog.h	117
calibrationdialog.h	118
coincounter.h	119
configdinspect.h	119
debug.h	
Debug macros	120
defines.h	
Various compile-time definitions	121
dinspect.h	125
dmath.h	126
helper.h	128
manualmotorcontroldialog.h	130
motor.h	130
pm100d.h	131
snspdcounter.h	132
spectrometeroptions.h	133
tunablelaser.h	134
udplistener.h	135
version.h	
This file contains information about the code version	136

Chapter 7

Namespace Documentation

7.1 dmath Namespace Reference

Functions

- int [find_max](#) (QVector< double > vec)
find_max determines the biggest entry in a vector and returns its position
- std::pair< double, double > [find_max](#) (QVector< double > frequ, QVector< double > wl)
find_max determines the biggest entry in a vector and returns the corresponding entry of both this and another vector
- double [normal_distribution](#) (double A, double sigma, double mu, double x)
normal_distribution evaluates the normal distribution with given parameters at a given point
- double [fwhm_gauss](#) (double s)
fwhm_gauss calculates the FWHM of a gaussian out of its standard deviation
- double [calc_stepsize](#) (QVector< double > wl, QVector< int > mpos)
calc_stepsize calculates the scanmove-stepsize of the spectrometer when edge-detection is not used
- double [calc_stepsize](#) (double l1, double l2, double l3, double l4, double ds12, double ds34)
calc_stepsize calculates the scanmove-stepsize of the spectrometer when edge-detection is not used
- double [calc_unneeded_stepsize](#) (Spectrometer *spectrometer, QVector< bool > SensorDataAtStep)
calc_unneeded_stepsize calculates the mean scanmove-stepsize of the spectrometer
- double [calc_deviationangle](#) (double G, double l1, double l2, double da)
calc_deviationangle calculates the deviation angle of the spectrometer
- double [calc_deviationangle_without_stepsize](#) (double gratingconstant, QVector< double > wl, QVector< double > mangle)
calc_deviationangle_without_stepsize calculates the deviation angle of the spectrometer (used in calibration). Does not rely on a fixed stepsize.
- double [calc_bandpass](#) (double G, double fin, double win, double alpha)
calc_bandpass the size of a bandpass
- std::pair< uint32_t, uint32_t > [calc_range](#) (Spectrometer *spectrometer, double sec_angle, double sec_angle ← percentage)
calc_range calculates the wavelength range accessible by the spectrometer
- double [calc_angle_offset](#) (Spectrometer *spectrometer, double l, int mpos)
calc_angle_offset calculates the offset-angle (used in calibration).
- double [calc_angle_offset_without_stepsize](#) (Spectrometer *spectrometer, QVector< double > wl, QVector< double > mangle)
calc_angle_offset_without_stepsize calculates the offset-angle (used in calibration). Does not rely on a fixed stepsize.

- double [relate_maxpos_to_angle_difference](#) ([Spectrometer](#) *spectrometer, QVector< bool > SensorData, AtStep, int mpos, bool calibrated_with_edge_detection)
relate_maxpos_to_angle_difference finds the angle of rotation at which the maximum was measured
- double [calc_alpha](#) (double G, double l, double dv)
calc_alpha calculates the angle of light incident on the grating at a given wavelength
- double [calc_beta](#) (double G, double l, double dv)
calc_beta calculates the angle of light reflected from the grating at a given wavelength
- double [calc_lambda_from_alpha](#) (double G, double a, double dv)
calc_lambda_from_alpha calculates the wavelength selected by a spectrometer at a rotation angle a
- void [calc_coupling_efficiency](#) ([Spectrometer](#) *spectrometer, QVector< double > *EfficiencyAtBP, QVector< double > *WavelengthAtBP)
calc_coupling_efficiency calculates the maximal coupling efficiency at each bandpass
- void [calc_steps_in_BP](#) (QVector< double > *WavelengthAtStep, QVector< double > *WavelengthAtBP, QVector< int > *StepsInBP)
calc_steps_in_BP calculates the number of steps in each bandpass
- std::pair< uint32_t, uint32_t > [calc_bandpasses_in_scanrange](#) (std::pair< double, double > scanrange, QVector< double > *WavelengthAtBP)
calc_bandpasses_in_scanrange calculates the number of bandpasses covered by the scanrange of the current measurement
- uint32_t [calc_number_of_steps_in_scanrange](#) (std::pair< uint32_t, uint32_t > bppos, QVector< int > *StepsInBP)
calc_number_of_steps_in_scanrange calculates the estimated number of steps between two bandpasses
- double [recalc_wavelength_at_step](#) ([Spectrometer](#) *spectrometer, double newstepsize, uint64_t minstep, uint64_t maxstep, QVector< double > *WavelengthAtStep_EC, double oldangle)
recalc_wavelength_at_step recalculates the wavelength at each measurement step already carried out, knowing the number of steps between two edges (which fix the angle of rotation)
- uint16_t [calc_steps_between_edges](#) (QVector< uint64_t > *EdgeAtStep)
calc_steps_between_edges calculates how many steps lie between
- std::pair< int, uint64_t > [recalc_steps_in_BP](#) (QVector< double > *WavelengthAtStep, QVector< double > *WavelengthAtBP, QVector< int > *StepsInBP, uint64_t minpos, uint64_t maxpos, std::pair< int, uint64_t > oldstate)
recalc_steps_in_BP calculates the number of datapoints found in each bandpass
- uint16_t [calc_how_many_edges_to_skip](#) ([Spectrometer](#) *spectrometer, double wl)
calc_how_many_edges_to_skip calculates how many edges have to be skipped to position as close as possible to a certain wavelength

7.1.1 Detailed Description

contains functions to evaluate equations and carry out algorithms

Bug There are no known bugs.

7.1.2 Function Documentation

7.1.2.1 calc_alpha()

```
double dmath::calc_alpha (
    double G,
    double l,
    double dv )
```

[calc_alpha](#) calculates the angle of light incident on the grating at a given wavelength

Parameters

<i>G</i>	The grating constant G in 1/mm
<i>l</i>	The wavelength
<i>dv</i>	The deviation angle in Rad

Returns

The angle of light incident on the grating

7.1.2.2 calc_angle_offset()

```
double dmath::calc_angle_offset (
    Spectrometer * spectrometer,
    double l,
    int mpos )
```

calc_angle_offset calculates the offset-angle (used in calibration).

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>l</i>	the wavelength
<i>mpos</i>	the position of the maximum

Returns

the offset angle

7.1.2.3 calc_angle_offset_without_stepsize()

```
double dmath::calc_angle_offset_without_stepsize (
    Spectrometer * spectrometer,
    QVector< double > wl,
    QVector< double > mangle )
```

calc_angle_offset_without_stepsize calculates the offset-angle (used in calibration). Does not rely on a fixed step-size.

Parameters

<i>spectrometer</i>	Structure containing parameters of the Spectrometer
<i>wl</i>	QVector containing wavelengths at which the calibration scans have been performed
<i>mangle</i>	QVector containing Angles (relative to offset-angle, which can thus still be unknown) at which maxima have been measured

Returns

Offset-angle: angle between 'Starting position' and 'angle at which light is reflected back into input direction'

7.1.2.4 calc_bandpass()

```
double dmath::calc_bandpass (
    double G,
    double fin,
    double win,
    double alpha )
```

calc_bandpass the size of a bandpass

Parameters

<i>G</i>	The grating constant G in 1/mm
<i>fin</i>	The focal length of the input mirror in mm
<i>win</i>	The width of the input slit (or the MFD of the input fibre)
<i>alpha</i>	The angle of incidence on the grating in Rad

Returns**7.1.2.5 calc_bandpasses_in_scanrange()**

```
std::pair< uint32_t, uint32_t > dmath::calc_bandpasses_in_scanrange (
    std::pair< double, double > scanrange,
    QVector< double > * WavelengthAtBP )
```

calc_bandpasses_in_scanrange calculates the number of bandpasses covered by the scanrange of the current measurement

Parameters

<i>scanrange</i>	the scanrange of the current measurement
<i>WavelengthAtBP</i>	the wavelength at each bandpass

Returns

7.1.2.6 calc_beta()

```
double dmath::calc_beta (
    double G,
    double l,
    double dv )
```

calc_beta calculates the angle of light reflected from the grating at a given wavelength

Parameters

<i>G</i>	The grating constant G in 1/mm
<i>l</i>	The wavelength
<i>dv</i>	The deviation angle in Rad

Returns

the angle of light reflected from the grating

7.1.2.7 calc_coupling_efficiency()

```
void dmath::calc_coupling_efficiency (
    Spectrometer * spectrometer,
    QVector< double > * EfficiencyAtBP,
    QVector< double > * WavelengthAtBP )
```

calc_coupling_efficiency calculates the maximal coupling efficiency at each bandpass

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>EfficiencyAtBP</i>	the maximal coupling efficiency at each bandpass
<i>WavelengthAtBP</i>	the wavelength at each bandpass

7.1.2.8 calc_deviationangle()

```
double dmath::calc_deviationangle (
    double G,
    double l1,
    double l2,
    double da )
```

calc_deviationangle calculates the deviation angle of the spectrometer

Parameters

<i>G</i>	The grating constant G in 1/mm
<i>l1</i>	A wavelength
<i>l2</i>	A wavelength
<i>da</i>	angle between positions at which the two wavelengths l1 and l2 occur

Returns

The deviation angle in Rad

The angle of incidence on the grating is

$$\alpha = \arcsin \left(\frac{G \cdot \lambda}{2 \cdot \cos(dv/2)} \right) - dv/2$$

So, the angle between two wavelengths is

$$d\alpha = \alpha_2 - \alpha_1 = \arcsin \left(\frac{G \cdot \lambda_2}{2 \cdot \cos(dv/2)} \right) - \arcsin \left(\frac{G \cdot \lambda_1}{2 \cdot \cos(dv/2)} \right)$$

In general:

$$\sin(\arcsin(a) - \arcsin(b)) = a \cdot \sqrt{1 - b^2} - b \cdot \sqrt{1 - a^2}$$

This can be transformed into an equation for the deviation angle:

$$dv = 2 \arccos \left(\frac{G \sqrt{\lambda_1^2 - 2 \sqrt{1 - (\alpha_2 - \alpha_1)^2} \lambda_1 \lambda_2 + \lambda_2^2}}{2c(\alpha_2 - \alpha_1)} \right)$$

7.1.2.9 calc_deviationangle_without_stepsize()

```
double dmath::calc_deviationangle_without_stepsize (
    double gratingconstant,
    QVector< double > wl,
    QVector< double > mangle )
```

calc_deviationangle_without_stepsize calculates the deviation angle of the spectrometer (used in calibration). Does not rely on a fixed stepsize.

Parameters

<i>gratingconstant</i>	The grating constant G in 1/mm
<i>wl</i>	The wavelength
<i>mangle</i>	The rotation angle at which the maximum appears

Returns

the deviationangle

Implements

$$d_v = 2 \sec^{-1} \left(\frac{-2 \sin(d\alpha_1 - d\alpha_2)}{G \sqrt{\lambda_1^2 + \lambda_2^2 - 2\lambda_1 \lambda_2 \cos(d\alpha_1 - d\alpha_2)}} \right)$$

7.1.2.10 calc_how_many_edges_to_skip()

```
uint16_t dmath::calc_how_many_edges_to_skip (
    Spectrometer * spectrometer,
    double wl )
```

calc_how_many_edges_to_skip calculates how many edges have to be skipped to position as close as possible to a certain wavelength

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>wl</i>	Input: wavelength to position to

Returns

the number of steps to skip

7.1.2.11 calc_lambda_from_alpha()

```
double dmath::calc_lambda_from_alpha (
    double G,
    double a,
    double dv )
```

calc_lambda_from_alpha calculates the wavelength selected by a spectrometer at a rotation angle a

Parameters

<i>G</i>	The grating constant G in 1/mm
<i>a</i>	The rotation angle
<i>dv</i>	The deviation angle in Rad

Returns

the wavelength

7.1.2.12 `calc_number_of_steps_in_scanrange()`

```
uint32_t dmath::calc_number_of_steps_in_scanrange (
    std::pair< uint32_t, uint32_t > bppos,
    QVector< int > * StepsInBP )
```

`calc_number_of_steps_in_scanrange` calculates the estimated number of steps between two bandpasses

Parameters

<i>bppos</i>	two indices specifying which bandpasses to compare
<i>StepsInBP</i>	number of steps in each bandpass

Returns

the number of steps between the two bandpasses

7.1.2.13 `calc_range()`

```
std::pair< uint32_t, uint32_t > dmath::calc_range (
    Spectrometer * spectrometer,
    double sec_angle,
    double sec_percentage )
```

`calc_range` calculates the wavelength range accessible by the spectrometer

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>sec_angle</i>	buffer angle (offset)
<i>sec_percentage</i>	buffer angle (factor)

Returns

7.1.2.14 `calc_steps_between_edges()`

```
uint16_t dmath::calc_steps_between_edges (
    QVector< uint64_t > * EdgeAtStep )
```

`calc_steps_between_edges` calculates how many steps lie between

Parameters

<i>EdgeAtStep</i>	list of steps where edges were found
-------------------	--------------------------------------

Returns

7.1.2.15 calc_steps_in_BP()

```
void dmath::calc_steps_in_BP (
    QVector< double > * WavelengthAtStep,
    QVector< double > * WavelengthAtBP,
    QVector< int > * StepsInBP )
```

calc_steps_in_BP calculates the number of steps in each bandpass

Parameters

<i>WavelengthAtStep</i>	the wavelength at each step
<i>WavelengthAtBP</i>	the wavelength at each bandpass
<i>StepsInBP</i>	the number of steps in each bandpass

7.1.2.16 calc_stepsize() [1/2]

```
double dmath::calc_stepsize (
    QVector< double > wl,
    QVector< int > mpos )
```

calc_stepsize calculates the scanmove-stepsize of the spectrometer when edge-detection is not used

Parameters

<i>wl</i>	a vector containing wavelengths in nm
<i>mpos</i>	a vector containing integers indicating at which step the corresponding wavelength in wl occurs

Returns

the mean scanmove-stepsize in μ Rad

This is an overloaded function.

See also

[calc_stepsize\(double l1, double l2, double l3, double l4, double ds12, double ds34\)](#)

7.1.2.17 calc_stepsize() [2/2]

```
double dmath::calc_stepsize (
    double l1,
    double l2,
    double l3,
    double l4,
    double ds12,
    double ds34 )
```

`calc_stepsize` calculates the scanmove-stepsizes of the spectrometer when edge-detection is not used

Parameters

<i>l1</i>	Wavelength of maximum 1
<i>l2</i>	Wavelength of maximum 2
<i>l3</i>	Wavelength of maximum 3
<i>l4</i>	Wavelength of maximum 4
<i>ds12</i>	The number of steps between wavelength 1 and 2
<i>ds34</i>	The number of steps between wavelength 3 and 4

Returns

the stepsizes of the spectrometer

This is an overloaded function.

See also

[calc_stepsize\(QVector<double> wl, QVector<int> mpos\)](#)

7.1.2.18 calc_unneeded_stepsize()

```
double dmath::calc_unneeded_stepsize (
    Spectrometer * spectrometer,
    QVector< bool > SensorDataAtStep )
```

`calc_unneeded_stepsize` calculates the mean scanmove-stepsizes of the spectrometer

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>SensorDataAtStep</i>	A vector containing the PCB-motor sensor-data at each step

Returns

the mean scanmove-stepsizes in μ Rad

7.1.2.19 find_max() [1/2]

```
int dmath::find_max (
    QVector< double > vec )
```

find_max determines the biggest entry in a vector and returns its position

Parameters

<i>vec</i>	a QVector of type double
------------	--------------------------

Returns

the index i at which vec.at(i) is maximal

7.1.2.20 find_max() [2/2]

```
std::pair< double, double > dmath::find_max (
    QVector< double > frequ,
    QVector< double > wl )
```

find_max determines the biggest entry in a vector and returns the corresponding entry of both this and another vector

Parameters

<i>frequ</i>	the vector of which the maximum will be determined returned
<i>wl</i>	the vector of which an entry will be returned

Returns

first entry: the maximum of vector frequ.

second entry: an entry with index i of the vector wl, where i is such that frequ has its maximal entry at index i

7.1.2.21 fwhm_gauss()

```
double dmath::fwhm_gauss (
    double s )
```

fwhm_gauss calculates the FWHM of a gaussian out of its standard deviation

Parameters

<i>s</i>	the standard deviation of the gaussian
----------	--

Returns

th FWHM of the gaussian

7.1.2.22 normal_distribution()

```
double dmath::normal_distribution (
    double A,
    double sigma,
    double mu,
    double x )
```

normal_distribution evaluates the normal distribution with given parameters at a given point

Evaluates the normal distribution with probability density function

$$y = \frac{A}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

at with a given parameterset at a given point.

Parameters

<i>A</i>	Factor <i>A</i>
<i>sigma</i>	The squareroot σ of the variance
<i>mu</i>	The mean μ
<i>x</i>	Point <i>x</i> at which the function is evaluated

Returns

the value of the normal distribution

7.1.2.23 recalc_steps_in_BP()

```
std::pair< int, uint64_t > dmath::recalc_steps_in_BP (
    QVector< double > * WavelengthAtStep,
    QVector< double > * WavelengthAtBP,
    QVector< int > * StepsInBP,
    uint64_t minpos,
    uint64_t maxpos,
    std::pair< int, uint64_t > oldstate )
```

recalc_steps_in_BP calculates the number of datapoints found in each bandpass

Parameters

<i>WavelengthAtStep</i>	the wavelengths of each datapoint
<i>WavelengthAtBP</i>	the wavelengths where each new bandpass starts
<i>StepsInBP</i>	the number of datapoints in each bandpass
<i>minpos</i>	start recalculation at this index
<i>maxpos</i>	stop recalculation at this index
<i>oldstate</i>	

Returns

7.1.2.24 recalc_wavelength_at_step()

```
double dmath::recalc_wavelength_at_step (
    Spectrometer * spectrometer,
    double newstepsize,
    uint64_t minstep,
    uint64_t maxstep,
    QVector< double > * WavelengthAtStep_EC,
    double oldangle )
```

`recalc_wavelength_at_step` recalculates the wavelength at each measurement step already carried out, knowing the number of steps between two edges (which fix the angle of rotation)

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>newstepsize</i>	the mean stepsize between the last two edges
<i>minstep</i>	start recalculation at this index
<i>maxstep</i>	stop recalculation at this index
<i>WavelengthAtStep_EC</i>	the 'edge-corrected' wavelength at each step
<i>oldangle</i>	the angle of rotation at which the last recalculation-iteration stopped

Returns

7.1.2.25 relate_maxpos_to_angle_difference()

```
double dmath::relate_maxpos_to_angle_difference (
    Spectrometer * spectrometer,
    QVector< bool > SensorDataAtStep,
    int mpos,
    bool calibrated_with_edge_detection = true )
```

`relate_maxpos_to_angle_difference` finds the angle of rotation at which the maximum was measured

Parameters

<i>spectrometer</i>	A spectrometer-structure containing device-specific variables and settings
<i>SensorDataAtStep</i>	A vector containing the PCB-motor sensor-data at each step
<i>mpos</i>	position of the maximum
<i>calibrated_with_edge_detection</i>	

Returns

the offset angle

7.2 helper Namespace Reference

contains small functions to display messages

Functions

- void [message](#) (QString msg)
message displays a message box
- void [error](#) (QString msg)
error displays an error-messagebox and writes a debug_error message to stdout
- void [warning](#) (QString msg)
warning displays warning-messagebox and writes a debug_warning message to stdout
- void [info](#) (QString msg)
info displays an info-messagebox and writes a debug_info message to stdout

7.2.1 Detailed Description

contains small functions to display messages

Bug There are no known bugs.

7.2.2 Function Documentation

7.2.2.1 error()

```
void helper::error (  
    QString msg )
```

error displays an error-messagebox and writes a debug_error message to stdout

Parameters

<i>msg</i>	the message to be displayed
------------	-----------------------------

7.2.2.2 info()

```
void helper::info (
```

```
QString msg )
```

info displays an info-messagebox and writes a debug_info message to stdout

Parameters

<i>msg</i>	the message to be displayed
------------	-----------------------------

7.2.2.3 message()

```
void helper::message (  
    QString msg )
```

message displays a message box

Parameters

<i>msg</i>	the message to be displayed
------------	-----------------------------

7.2.2.4 warning()

```
void helper::warning (  
    QString msg )
```

warning displays warning-messagebox and writes a debug_warning message to stdout

Parameters

<i>msg</i>	the message to be displayed
------------	-----------------------------

Chapter 8

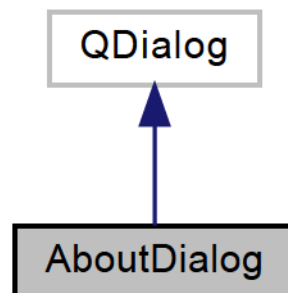
Class Documentation

8.1 AboutDialog Class Reference

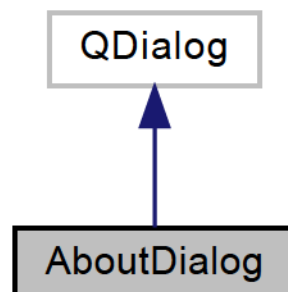
The [AboutDialog](#) class is just used to show the about-dialog.

```
#include "aboutdialog.h"
```

Inheritance diagram for AboutDialog:



Collaboration diagram for AboutDialog:



Public Member Functions

- **AboutDialog** (QWidget *parent=0)

Private Attributes

- `Ui::AboutDialog * ui`

8.1.1 Detailed Description

The [AboutDialog](#) class is just used to show the about-dialog.

Bug There are no known bugs.

The documentation for this class was generated from the following files:

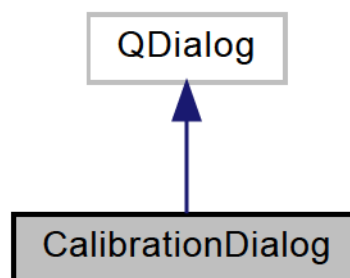
- [aboutdialog.h](#)
- [aboutdialog.cpp](#)

8.2 CalibrationDialog Class Reference

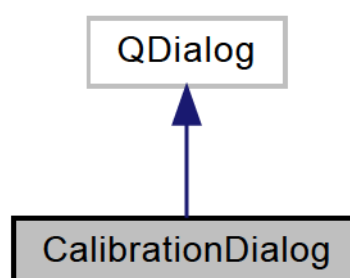
The [CalibrationDialog](#) class handles userinput to configure the calibration-procedure.

```
#include <calibrationdialog.h>
```

Inheritance diagram for CalibrationDialog:



Collaboration diagram for CalibrationDialog:



Public Slots

- bool [ret_understood](#) ()
ret_understood check if user states that he/she knows what she/he's doing
- bool [ret_wantnewscan](#) ()
ret_wantnewscan check if new scans shall be carried out
- bool [ret_singlerun](#) ()
ret_singlerun check if only a single calibration run shall be carried out
- void [reset](#) ()
reset resets the controls in the GUI to standard values

Signals

- void [CalibrationDialogDone](#) (bool singlerun, bool newscan, int [steps_to_go](#), int [steps_wo_measurement](#), double [sec_angle](#), double [sec_percentage](#))
CalibrationDialogDone emitted when the calibration dialog was accepted.

Public Member Functions

- **CalibrationDialog** (QWidget *parent=0)

Public Attributes

- bool [understood](#)
True if the user ticked 'I understand that this is dangerous'.
- bool [wantnewscan](#)
True if user wants to perform the calibration with new scans.
- bool [onlysinglerun](#)
True if user wants to perform only a single calibration-measurement.
- int [steps_to_go](#)
Total number of steps to go in the current calibration-scan.
- int [steps_wo_measurement](#)
Number of steps to go before measurement begins in calibration scan. Depending on the integration time set, this may save a lot of time.
- double [sec_angle](#)
The accesible angle-range will be shortened by this fixed offset angle on both sides.
- double [sec_percentage](#)
The accesible angle-range will be shortened by this factor on both sides.

Private Slots

- void [dialogfinished](#) (int result)
dialogfinished starts a calibration measurement if the dialog was accepted and the necessary boxes were ticked
- void [get_security_angle](#) (double angle)
get_security_angle reads from the ui: a constant offset to both ends of the final accessible angle-range
- void [get_security_percentage](#) (double percentage)
get_security_percentage reads from the ui: a factor to shorten to both ends of the final accessible angle-range
- void [get_steps_to_go](#) (int stg)
get_steps_to_go reads from the ui: the number of steps to carry out in the current calibration measurement
- void [get_steps_wo_measurement](#) (int swom)
get_steps_wo_measurement reads from the ui: the number of steps to go before the calibration measurement starts (to save time by not waiting for the detectors)
- void [get_onlysinglerun](#) (bool single)
get_onlysinglerun check if only a single calibration run shall be carried out

Private Attributes

- `Ui::CalibrationDialog * ui`

8.2.1 Detailed Description

The `CalibrationDialog` class handles userinput to configure the calibration-procedure.

Bug There are no known bugs.

8.2.2 Member Function Documentation

8.2.2.1 CalibrationDialogDone

```
void CalibrationDialog::CalibrationDialogDone (
    bool singlerun,
    bool newscan,
    int steps_to_go,
    int steps_wo_measurement,
    double sec_angle,
    double sec_percentage ) [signal]
```

CalibrationDialogDone emitted when the calibration dialog was accepted.

Parameters

<i>singlerun</i>	True if only a single calibration measurement has to be performed
<i>newscan</i>	True if a measurement is carried out (instead of loading saved calibration spectra)
<i>steps_to_go</i>	The number of steps to take data at during the measurement
<i>steps_wo_measurement</i>	The number of steps to go without taking data (before <i>steps_to_go</i> datapoints will be aquired)
<i>sec_angle</i>	The accesible angle-range will be shortened by this fixed offset angle on both sides
<i>sec_percentage</i>	The accesible angle-range will be shortened by this factor on both sides

When the calibration dialog was accepted (with the 'OK' button) and all necessary boxes were checked, this signal is emitted to transmit all necessary data to the dinspect class, which then performs the calibration or a calibration measurement (depending on which option was chosen in the dialog).

8.2.2.2 dialogfinished

```
void CalibrationDialog::dialogfinished (
    int result ) [private], [slot]
```

dialogfinished starts a calibration measurement if the dialog was accepted and the necessary boxes were ticked

Parameters

<i>result</i>	indicating if dialog was accepted or rejected (if the 'ok' or the 'abort' button was clicked)
---------------	---

8.2.2.3 get_onlysinglerun

```
void CalibrationDialog::get_onlysinglerun (  
    bool single ) [private], [slot]
```

get_onlysinglerun check if only a single calibration run shall be carried out

Parameters

<i>single</i>	true if it is so
---------------	------------------

8.2.2.4 get_security_angle

```
void CalibrationDialog::get_security_angle (  
    double angle ) [private], [slot]
```

get_security_angle reads from the ui: a constant offset to both ends of the final accessible angle-range

Parameters

<i>angle</i>	the offset angle
--------------	------------------

8.2.2.5 get_security_percentage

```
void CalibrationDialog::get_security_percentage (  
    double percentage ) [private], [slot]
```

get_security_percentage reads from the ui: a factor to shorten to both ends of the final accessible angle-range

Parameters

<i>percentage</i>	the factor
-------------------	------------

8.2.2.6 get_steps_to_go

```
void CalibrationDialog::get_steps_to_go (
    int stg ) [private], [slot]
```

`get_steps_to_go` reads from the ui: the number of steps to carry out in the current calibration measurement

Parameters

<i>stg</i>	number of steps to go
------------	-----------------------

8.2.2.7 get_steps_wo_measurement

```
void CalibrationDialog::get_steps_wo_measurement (
    int swom ) [private], [slot]
```

`get_steps_wo_measurement` reads from the ui: the number of steps to go before the calibration measurement starts (to save time by not waiting for the detectors)

Parameters

<i>swom</i>	number of steps without measurement
-------------	-------------------------------------

8.2.2.8 ret_singlerun

```
bool CalibrationDialog::ret_singlerun ( ) [slot]
```

`ret_singlerun` check if only a single calibration run shall be carried out

Returns

the state of the corresponding checkbox

8.2.2.9 ret_understood

```
bool CalibrationDialog::ret_understood ( ) [slot]
```

`ret_understood` check if user states that he/she knows what she/he's doing

Returns

the state of the corresponding checkbox

8.2.2.10 ret_wantnewscan

```
bool CalibrationDialog::ret_wantnewscan ( ) [slot]
```

ret_wantnewscan check if new scans shall be carried out

Returns

the state of the corresponding checkbox

The documentation for this class was generated from the following files:

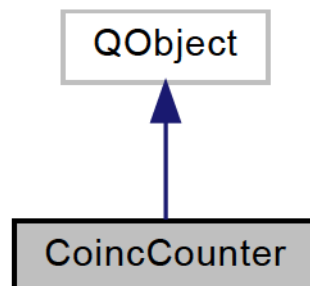
- [calibrationdialog.h](#)
- [calibrationdialog.cpp](#)

8.3 CoincCounter Class Reference

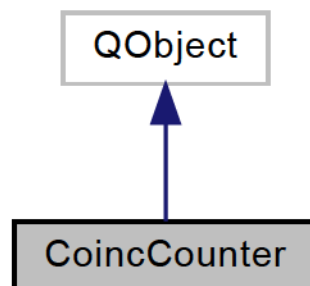
The [CoincCounter](#) class acquires coincidence frequencies from a TM8000.

```
#include <coinccounter.h>
```

Inheritance diagram for CoincCounter:



Collaboration diagram for CoincCounter:



Public Member Functions

- **CoincCounter** (DataSource *gMainWnd, TTMViewOptions *gOptions, QWidget *parent=0)

8.3.1 Detailed Description

The [CoincCounter](#) class acquires coincidence frequencies from a TM8000.

This class is similar to the 'MultiCoincidences' class from the AIT sample program TTMViewer, but a lot of unnecessary functionality was removed. This class is no longer a QWidget and all GUI-parts were removed. Functions were added to interface with dinspect. The coincidence determination was altered to only calculate coincidences between two stops.

Author

Roland Lieger

Bug There are no known bugs.

The documentation for this class was generated from the following files:

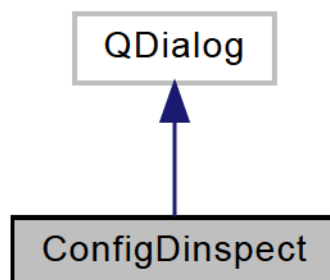
- [coinccounter.h](#)
- [coinccounter.cpp](#)

8.4 ConfigDinspect Class Reference

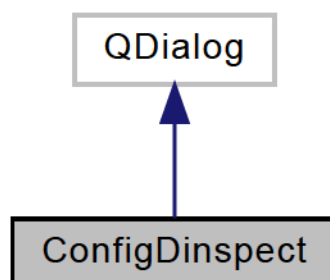
The [ConfigDinspect](#) class handles user input to configure the behaviour of dinspect.

```
#include <configdinspect.h>
```

Inheritance diagram for ConfigDinspect:



Collaboration diagram for ConfigDinspect:



Public Member Functions

- **ConfigDinspect** (QSettings *getsettings, QWidget *parent=0)

Private Member Functions

- void [LoadConfig](#) ()
LoadConfig fills the elements of the dialog.
- void [SaveConfig](#) ()
SaveConfig stores the contents of the dialog.
- void [ChooseAutosaveDir](#) ()
ChooseAutosaveDir opens a dialog to set the folder which is used to autosave spectra.

Private Attributes

- Ui::ConfigDinspect * **ui**
- QSettings * [settings](#)
An object used permanently store settings.

8.4.1 Detailed Description

The [ConfigDinspect](#) class handles user input to configure the behaviour of dinspect.

Bug There are no known bugs.

The user input is handled via a dialog. The settings are stored in a QSettings object.

8.4.2 Member Function Documentation

8.4.2.1 LoadConfig()

```
void ConfigDinspect::LoadConfig ( ) [private]
```

LoadConfig fills the elements of the dialog.

The dialog is set up with values already stored in the QSettings object. If a specific quantity does not exist there, it is set to a standard value.

Public Slots

- void [showStatusMessage](#) (const QString &message)
showStatusMessage updates a label showing the current status of the serial connection (Connected, disconnected, errorcode, ...).
- void [showPositionInfo](#) (const QString &message)
showPositionInfo updates a label stating what the spectrometer is currently doing (Aquiring data? Stopping? Positioning? Waiting for something?...).
- void [showDataInfo](#) (const QString &message)
showDataInfo updates a label showing the newest aquired frequency/power.
- void [showActivityInfo](#) (const QString &message)
showActivityInfo updates a label in the UI.
- void [showCurrentWLInfo](#) (double wl)
showCurrentWLInfo updates a label stating up to which wavelength the scan progressed, so far.
- void [showCurrentMaxInfo](#) (double maxwl)
showCurrentMaxInfo updates a label stating where the maximum was found
- void [update_spectrometer_info](#) (uint16_t type, uint16_t subversion)
update_spectrometer_info receives the major and minor version of the connected spectrometer from the motor class and updates the UI to show it.
- void [GetFrequencyFromSNSPDcounter](#) (double frequency)
GetFrequencyFromSNSPDcounter receives the current frequency from the TTM based counter class and stores it in the appropriate vector.
- void [GetPowerFromPM100D](#) ()
GetPowerFromPM100D reads the current power value from the [PM100D](#), checks it and stores it in the appropriate vector.
- void [setPM100D_AverageCount](#) (int newAvCnt)
setPM100D_AverageCount calls the function to set the number of samples the powermeter averages over
- void [CalibrationDialogFinished](#) (bool singlerun, bool newscan, int steps_to_go, int steps_wo_measurement, double sec_angle, double sec_percentage)
CalibrationDialogFinished receives parameters of the calibrationdialog and, depending on those parameters, either starts the calibration or performs (one or more) calibration measurements.
- void [motorfoundfallingedge](#) ()
motorfoundfallingedge called when a falling edge is detected in the PCBMotor encoder wheel sensor data during a measurement (not during initial positioning)
- void [motorfoundrisingedge](#) ()
motorfoundrisingedge called when a rising edge is detected in the PCBMotor encoder wheel sensor data during a measurement (not during initial positioning)
- void [motor_skipped_init_edges](#) ()
motor_skipped_init_edges gets called when the initial positioning is done to continue the measurement
- void [motor_is_at_starting_position](#) ()
motor_is_at_starting_position
- void [slot_update_ui](#) ()
Can be called to initiate an update of graphicsal user interface.

Public Member Functions

- **Dinspect** (QString *TTMofn, QString *TTMorfn, TTMViewOptions *TTMo, QWidget *parent=0)
- void [setupGraph](#) (QCustomPlot *graph)
setupGraph Sets up appearance and behaviour of the graph.

Private Slots

- void [on_DetectorComboBox_activated](#) (const QString &arg1)
on_DetectorComboBox_activated Detects which Detector the user wants to use.
- void [on_Button_OpenMotor_clicked](#) ()
on_Button_OpenMotor_clicked tries to establish a serial connection
- void [offsetdialog](#) ()
offsetdialog creates a dialog to adjust the TTM channel offsets for coincidence measurement
- void [initMotor](#) ()
initMotor opens a serial connection and tries to find out if it is a spectrometer
- void [TTM_IP_changed](#) (QString ip)
TTM_IP_changed changes the IP which is used to listen for ttm-packages.
- void [TTM_Port_changed](#) (int port)
TTM_Port_changed changes the port which is used to listen for ttm-packages.
- void [TTMcoincWindowChanged](#) ()
TTMcoincWindowChanged gets coincidence window from GUI.
- void [updateIntegrationtime](#) (int newinttime)
updateIntegrationtime gets the time over which the frequency is acquired (using TTM-based counters)
- void [allocate_arrays](#) ()
allocate_arrays allocates necessary arrays with the right size
- void [SaveGraph](#) (bool automatic=false)
SaveGraph saves a plot.
- void [send_data_to_plot](#) ()
send_data_to_plot updates the plot with new data. Resizes the plot range (frequency-wise) automatically.
- void [clear_plot](#) ()
clear_plot resets the plot
- void [update_graph_labels](#) ()
update_graph_labels sets labels of graph according to whether measurement is done with powermeter or with TTM (and there whether it is singles- or coincidences)
- void [plot_savefile](#) (uint16_t savefileversion)
plot_savefile plots data from a saved spec-file
- void [get_detector_data](#) (double freq)
get_detector_data is a slot that acquires the frequency from the TTM-based counters
- void [start_counting](#) ()
start_counting Tells the TTM-based counters to start a frequency determination or acquires the frequency from the PM100D.
- void [scan_loop](#) ()
scan_loop iterates between data acquisition and repositioning the motor.
- void [end_scan](#) ()
end_scan stops finalizes the measurement process.
- void [SNSPD_update_channel](#) (int getchannel)
SNSPD_update_channel called when the TTM channel was changed. Updates the snspsdcounter and the coincidencecounter with the new channel.
- void [SNSPD_update_coinc_channel](#) (int getchannel)
SNSPD_update_coinc_channel called when the TTM coincidence channel was changed. Updates the snspsdcounter and the coincidencecounter with the new channel.
- void [ScanRangeMinPositioning](#) ()
ScanRangeMinPositioning positions the motor to select a wavelength below or at the lower scan range bound.
- void [calibrate](#) ()
calibrate starts the calibration process.
- void [calibrate_wo_stepsize](#) ()
calibrate_wo_stepsize starts the calibration process. Does not rely on a fixed stepsize.

- void [motorfoundinitedge](#) (bool rising, int stepstoedge)
motorfoundinitedge called during initial positioning when the motor found an edge
- void [GraphMouseWheel](#) ()
GraphMouseWheel specifies what happens when the mouse wheel is turned over the plot.
- void [GraphMousePress](#) ()
GraphMousePress specifies what happens when a mouse button is pressed over the plot.
- void [GraphSelectionChanged](#) ()
GraphSelectionChanged sets the actions for different events based on which part of the graph is selected.
- void [GraphDoubleClick](#) (QMouseEvent *event)
GraphDoubleClick handles doubleclicks on the graph. Adds a wavelength-marker and a label showing the wavelength the marker is positioned at.
- void [GraphAxisrangechanged](#) (QCPRange newRange)
GraphAxisrangechanged takes care of non-negativity of the x-axis when zooming/dragging.
- void [GraphYaxisrangechanged](#) (QCPRange newRange)
GraphYaxisrangechanged takes care of non-negativity of the y-axis when zooming/dragging.
- void [UDPCOMstartmeasurement](#) ()
UDPCOMstartmeasurement starts measurement if UDP-control is enabled and no measurement is currently in progress. Called when the [UDPlistener](#) receives the command to start a measurement.
- void [UDPCOMsetdetector](#) (QString det)
UDPCOMsetdetector changes the detector for the measurement. . Called when the [UDPlistener](#) receives the command to change the detector.
- void [UDPCOMstopmeasurement](#) ()
UDPCOMstopmeasurement Stops the measurement. . Called when the [UDPlistener](#) receives the command to stop the measurement.
- void [UDPCOMsetscanrange](#) (double low, double high)
UDPCOMsetscanrange Sets the scanrange. Called when the [UDPlistener](#) receives the command to change the scanrange.
- void [UDPCOMsetttmcoincidencechannel](#) (int channel)
UDPCOMsetttmcoincidencechannel Changes the TTM coincidence channel. Called when the [UDPlistener](#) receives the command to change the TTM coincidence channel in use.
- void [UDPCOMsetttmchannel](#) (int channel)
UDPCOMsetttmchannel Changes the TTM channel. Called when the [UDPlistener](#) receives the command to change the TTM channel in use.
- void [saveAutomatically](#) ()
saveAutomatically saves spectrum and plot automatically when a measurement is finished

Private Member Functions

- void [showOptionDialog](#) ()
showOptionDialog Opens a dialog where some functionality of the application can be controlled.
- void [showCalibrationDialog](#) ()
showCalibrationDialog Shows the dialog with which the mode of calibration is to be selected.
- void [calibrate_with_existing_data](#) ()
calibrate_with_existing_data Starts the calibration with data of already performed calibration measurements.
- void [calibrate_with_new_scans](#) ()
calibrate_with_new_scans Starts a number of calibration measurements, when all necessary measurements for calibration shall be carried out.
- void [single_calibration_run](#) ()
single_calibration_run Starts a calibration measurements, when only a single one is to be carried out.
- void [discretize_measurement_points](#) ()
discretize_measurement_points Splits accessible wavelengthrange into resolvable bandpasses
- void [recalc_with_dynamic_stepsize](#) ()

- recalc_with_dynamic_stepsize* recalculates several quantities once an edge was detected.
- void [initActionsConnections](#) ()
 - initActionsConnections* establishes the Qt signal-slot connections
- void [GraphConnections](#) ()
 - GraphConnections* sets up connections related to the graph/plot.
- void [initVars](#) ()
 - initVars* initializes all variables.
- void [initUI](#) ()
 - initUI* initializes the user interface
- void [about](#) ()
 - about* opens the about dialog
- void [opencontrolmotordialog](#) ()
 - opencontrolmotordialog* opens the dialog to manually control the motor and position roughly to a specific wavelength
- void [save](#) (bool automatic=false)
 - save* saves the data aquired during a measurement to a file
- void [save_calibration_run](#) ()
 - save_calibration_run* saves the data aquired during a calibration measurement to a file
- void [load](#) ()
 - load* opens the open/load dialog
- void [help](#) ()
 - help* opens the documentation pdf
- void [full_scan](#) ()
 - full_scan* initializes a measurement scan.
- void [stop_scan](#) ()
 - stop_scan* stops a scan if possible.
- void [start_detectors](#) ()
 - start_detectors* starts the detectors
- void [full_calibration_scan](#) ()
 - full_calibration_scan* initializes a calibration scan
- void [updateScanrange](#) ()
 - updateScanrange* updates the corresponding variables when the scanrange was edited in the GUI or the appropriate control command was received over the network.
- void [getSpectrometerType](#) ()
 - getSpectrometerType* starts the determination process of the spectrometer version
- void [getSpectrometerSubversion](#) ()
 - getSpectrometerSubversion* starts the determination process of the spectrometer subversion
- void [ActionAfterEdgeDetection](#) ()
 - ActionAfterEdgeDetection* (re)calculates various quantities once an edge was detected.
- void [updateUI](#) ()
 - updateUI* fills the Graphics User Interface with information.

Private Attributes

- Ui::Dinspect * [ui](#)
- QLabel * [status](#)
 - A label displaying status messages.*
- QLabel * [PositionInfo](#)
 - A label displaying the current und estimated final step.*
- QLabel * [DataInfo](#)
 - A label displaying the current frequency/power.*

- `QLabel` * [ActivityInfo](#)
A label displaying which action is currently performed.
- `QLabel` * [CurrentWLInfo](#)
A label displaying up to which wavelength the measurement has been performed.
- `QLabel` * [CurrentMaxInfo](#)
A label displaying at which wavelength the maximal frequency/power was detected up to now.
- `Motor` * [motor](#)
A class handling the serial communication with the PCBMotor.
- `TTMViewOptions` * [TTMOptions](#)
A class handling preferences and settings for all TTM8000 related calls.
- `DataSource` * [DataSrc](#)
The class responsible for the communication with the TTM8000.
- `QString` [TTMOptionsFileName](#)
- `QString` [TTMOptionsRestartFileName](#)
- `SNSPDcounter` * [snspsdcounter](#)
A class used to acquire the event rate of a single channel of a TTM8000.
- `CoincCounter` * [coinccounter](#)
A class used to acquire the coincidence rate between two channels of a TTM8000.
- `spectrometeroptions` * [SpecOptions](#)
A class handling access to the spectrometer configuration files.
- `Spectrometer` [spectrometer](#)
A structure containing all necessary information about the currently connected spectrometer.
- `tunablelaser` * [laser](#)
A class handling a tunable laser, controlled by a textfile.
- `PM100D` * [pm100d](#)
A class handling communication with the Thorlabs [PM100D](#) powermeter.
- `UDPlistener` * [udplistener](#)
Listens for commands on a UDP port.
- `ConfigDinspect` * [dopts](#)
A class handling reading and writing application settings.
- `QSettings` * [dsettings](#)
An object in which settings can be permanently stored.
- `AboutDialog` * [aboutdialog](#)
A dialog showing the version of dinspect.
- `ManualMotorControlDialog` * [manualmotorcontroldialog](#)
A dialog used to let the user send commands directly to the PCBMotor. Also used to roughly position at a given wavelength.
- `CalibrationDialog` * [caldiag](#)
A dialog used to specify how the calibration has to be carried out.
- `QCPIItemText` * [GraphTextMaxPos](#)
Text displaying wavelength at which maximum appears.
- `QCPIItemStraightLine` * [WLMarker](#)
Vertical line in graph.
- `QCPIItemText` * [WLMarkerText](#)
Text showing wavelength the marker is at.
- `bool` [wantnewscan](#)
True if user wants to perform the calibration with new scans. Set in calibration dialog.
- `bool` [understood](#)
True if the user ticked 'I understand that this is dangerous' in the calibration dialog.
- `bool` [cal_singlerun](#)
True if user wants to perform only a single calibration-measurement. Set in calibration dialog.

- int [cal_steps_to_go](#)
Total number of to go in the current calibration-scan. Set in calibration dialog.
- int [cal_steps_wo_measurement](#)
Number of steps to go before measurement begins in calibration scan. Depending on the integration time set, this may save a lot of time. Set in calibration dialog.
- double [cal_sec_angle](#)
The accesible angle-range will be shortened by this fixed offset angle on both sides.
- double [cal_sec_percentage](#)
The accesible angle-range will be shortened by this factor on both sides.
- uint8_t [calibration_loaded_datasets](#)
Number of datasets loaded for calibration purpose.
- bool [calibration_in_progress](#)
True if calibration is in progress.
- bool [calibrationscan](#)
True if a calibration scan is performed.
- QVector< int > [calibration_max_pos](#)
Vector containing the positions of the maxima of the calibration-scans.
- QVector< double > [calibration_max_angle](#)
Vector containing the angles th which the maxima of the calibration-scans appear.
- QVector< double > [calibration_wavelength](#)
Vector containing the wavelengths at which calibration-scans were taken. User-set.
- uint16_t [savefile_version](#)
Savefile version read from savefile. Used to determine how to read data.
- int [askmotor](#)
Specifies which information shall be extracted out of the text transmitted by the PCBMotor.
- uint8_t [failed_serial_connection_tries](#)
Stores failed attempts to establish serial connection to the PCBMotor.
- int [snspd_channel](#)
The channel of the TTM8000 from which data is read.
- int [snspd_coinc_channel](#)
Coincidences between snspsd_channel and snspsd_coinc_channel will be read from TTM8000.
- bool [measure_coincidences](#)
True if coincidences shall be counted.
- bool [measure_singles](#)
True if singles shall be counted.
- uint8_t [calibration_run](#)
When performing a calibration with new scans, this variable stores the number of the current iteration of the scans.
- uint8_t [calibration_runs_needed](#)
States how many datasets are needed to calibrate the spectrometer.
- uint32_t [SpectrometerType](#)
Used to identify the connected spectrometer.
- uint32_t [SpectrometerSubversion](#)
Used to identify the connected spectrometer.
- uint32_t [integrationtime](#)
When using TTM8000, this is the timespan over which data is aquired in seconds.
- uint32_t [steptime](#)
Time to wait between scanloop-iterations in msec.
- uint8_t [full_scan_iteration](#)
Used to iterate through the different tasks of a measurement.
- uint8_t [calibration_scan_iteration](#)
Used to iterate through the different tasks of a calibration measurement.

- `uint8_t calibration_singlescan_iteration`
Used to iterate through the different tasks of a calibration measurement.
- `uint8_t calibrate_with_new_scans_iteration`
Used to iterate through the different tasks of a calibration, consisting out of several calibration measurements.
- `uint32_t stepsToScanRangeMin`
Number of steps to go from initial (lowest accessible) starting position to the wavelength at which the measurement shall begin.
- `uint32_t currentBP`
Counter to access bandpass the current step corresponds to (approximately, without recalculation using edges).
- `uint32_t current_EC_BP`
Counter to access the bandpass up to which the recalculation (using edges) has been done.
- `uint32_t currstepsInBP`
Single value out of StepsInBP, which the current wavelength corresponds to.
- `uint16_t frequenciesWrittenIntoBP`
States how many times a frequency has been written into the appropriate entry in the vector FrequencyAtBandpass(currentBP). Needed to realize when to change of current FrequencyAtBandpass - entry.
- `uint64_t current_step`
Step which is currently measured.
- `uint64_t steps_needed`
Steps which should be measured in total. In a measurement scan, this value is approximate since the stepsize is unknown.
- `double TTMcoincWindow`
TTM coincidence window width in ns.
- `std::pair< double, double > max`
Maximal frequency (or power) and wavelength at which the maximum of the scan occurs. Calculated after the scan has finished.
- `std::pair< double, double > current_max`
Holds the Frequency/Power as well as the wavelength at the current maximum of the 'original' curve (variable to update during scan).
- `std::pair< double, double > current_max_EC`
Holds the Frequency/Power as well as the wavelength at the current maximum of the EdgeDetection-corrected curve (variable to update during scan).
- `QVector< double > FrequencyAtStep`
Vector containing the frequency at each step.
- `QVector< double > WavelengthAtStep`
Vector containing the wavelength at each step.
- `QVector< double > WavelengthAtStep_EC`
Vector containing the wavelength at each step, corrected with stepsize calculated through edgedetection.
- `QVector< double > WavelengthAtBP`
Vector containing the fwavelength at each bandpass.
- `QVector< double > FrequencyAtBP`
Vector containing the frequency at each bandpass.
- `QVector< double > FrequencyAtBP_EC`
Vector containing the frequency at each bandpass, corrected with stepsize calculated through edgedetection.
- `QVector< double > WavelengthAtBPbak`
Vector containing the wavelength at each bandpass, used for backup when loading spectrum.
- `QVector< double > FrequencyAtStepbak`
Vector containing the frequency at each bandpass, used for backup when loading spectrum.
- `QVector< double > EfficiencyAtBP`
Vector containing the couplingefficiency at each bandpass.
- `QVector< int > StepsInBP`
Vector containing the number of steps in each bandpass.

- QVector< int > [StepsInBP_EC](#)
Vector containing the number of steps in each bandpass, corrected with stepsize calculated through edgedetection.
- QVector< double > [Step](#)
Vector containing the number of steps. Only double because QCustomPlot requires QVector<double> for plotting.
- QVector< bool > [SensordataAtStep](#)
Vector containing the sensordata of the PCB-Motor at each step. 0: low. 1: high.
- QVector< uint64_t > [EdgeAtStep](#)
Vector containing the steps at which edges appear.
- QVector< bool > [EdgeType](#)
Vector containing the type of the edges in EdgeAtStep. 0:falling, 1:rising.
- int [stepstofirstedge](#)
Holds the number of movements that were done before the first edge was found.
- bool [lookforinitedge](#)
True if we're looking for an edge for the very first time.
- double [current_stepsize](#)
The mean stepsize since the last edge, recalculated when an edge was detected.
- int [bps_to_skip_before_scanrangemin](#)
Number of bandpasses between the lowest accessible wavelength and the measurement starting wavelength.
- std::pair< int, uint64_t > [recalc_SIBP_state](#)
Used to recalculate the number of steps in each Bandpass. Saves the datapoint up to which the recalculation has already been carried out for overlapping iterations.
- std::pair< uint64_t, int > [recalc_FIBP_state](#)
Used to recalculate the frequency/power in each Bandpass. Saves datapoint up to which the recalculation has already been carried out for overlapping iterations.
- double [recalc_WAS_angle](#)
Grating rotation angle up to which which the wavelength at each step was recalculated using the angle-information aquired from edge-detection.
- uint64_t [io_num_datapoints](#)
number of datapoints in a savefile.
- int [saveimagewidth](#)
Save plot with this width (in pixels).
- int [saveimageheigth](#)
Save plot with this heigth (in pixels).
- bool [detector_is_ttm](#)
True if the detector is a TTM8000.
- bool [detector_is_pm100d](#)
True if the detector is a PM100D.
- bool [detector_is_fpga](#)
True if the detector is an FPGA.
- bool [detector_is_powermeter](#)
True if the detector is a powermeter.
- bool [spectrometerisnew](#)
True if a new spectrometer was connected, used to update UI and configuration.
- bool [motorready](#)
True if the PCBMotor is ready for action.
- bool [scanisrunning](#)
True during measurement/calibration.
- bool [stopbutton_clicked](#)
True if 'Stop scan' button clicked.
- bool [consecutive_measurement](#)

Set to true after the first measurement has been taken to decide which things to reset when another measurement is started.

- bool [LOOPrepositioning](#)
True if 'scanmove' shall be done.
- bool [LOOPdetecting](#)
True if grating is positioned and frequency can be measured.
- bool [reached_scan_range_max](#)
True if last (Edge-corrected) calculated Bandpass is above the selected wavelength up to which the user wants to scan.
- bool [realignment](#)
True if realignment has to be done.
- bool [savedfileisopen](#)
True if savefile is open currently.
- bool [calibrationisbackedup](#)
True if backup of WavelengthAtBP was created, needed for functionality after opening savefile.
- bool [savesensordata](#)
True if the sensordata at each step should be saved to a file.
- bool [last_edge_is_rising](#)
1 if last edge was rising, 0 if last edge was falling
- bool [PlotFrequencyAgainstStep](#)
Frequency against step is plotted in calibration run (instead of Frequency against Wavelength in a measurement run)
- double [scanrangeMin](#)
The lower wavelength bound of the range which shall be scanned during the measurement.
- double [scanrangeMax](#)
The upper wavelength bound of the range which shall be scanned during the measurement.
- QTimer [GetSpectrometerTypeTimer](#)
Used to start the aquisition process of the spectrometer version.
- QTimer [GetSpectrometerSubversionTimer](#)
Used to start the aquisition process of the spectrometer subversion.
- QTimer [UpdateUITimer](#)
Used to update the Graphical User Interface.
- QTimer [Scanlooptimer](#)
Used to iterate through the different steps of a measurement.
- QTimer [ScanRangeMinPositionTimer](#)
Used to start the process of positioning to the lower bound of the measurement wavelength range.
- QTimer [CalibrationWithNewScanTimer](#)
Used to start a calibration measurement.
- QTimer [fullscantimer](#)
Used to iterate through the different steps of the preparation of a measurement.
- QTimer [calibrationscantimer](#)
Used to iterate through the different steps of the preparation of a calibration measurement.
- bool [pm100d_success](#)
Indicating if the aquisition of data from the [PM100D](#) was successful.
- int [pm100d_retry_count](#)
The number of already failed tries to get data from the [PM100D](#) for ONE datapoint. Reset on success.
- int [pm100d_retry_count_max](#)
The maximum number of tries to aquire data from the [PM100D](#) before the power value for this point is set to 0.

8.5.1 Detailed Description

The [Dinspect](#) class is the main class of this application.

Bug Randomly, reading data from the TTM socket fails. It seems that the TTMDatPacket_t is not available. This has only been observed on windows.

8.5.2 Member Function Documentation

8.5.2.1 ActionAfterEdgeDetection()

```
void Dinspect::ActionAfterEdgeDetection ( ) [private]
```

ActionAfterEdgeDetection (re)calculates various quantities once an edge was detected.

The stepsize of the PCBMotor is not constant, but the angle between two edges in the motor control driver encoder wheel signal is. Once an edge is detected, the current state of rotation is known and we are able to calculate back which wavelength was selected at the steps carried out between the last edge and the current one.

This function calculates the average stepsize since the last edge, calls the recalculation routines, and updates the plot.

See also

[recalc_with_dynamic_stepsize\(\)](#)

8.5.2.2 CalibrationDialogFinished

```
void Dinspect::CalibrationDialogFinished (
    bool singlerun,
    bool newscan,
    int steps_to_go,
    int steps_wo_measurement,
    double sec_angle,
    double sec_percentage ) [slot]
```

CalibrationDialogFinished receives parameters of the calibrationdialog and, depending on those parameters, either starts the calibration or performs (one or more) calibration measurements.

Parameters

<i>singlerun</i>	True if only a single calibration measurement has to be performed
<i>newscan</i>	True if a measurement is carried out (instead of loading saved calibration spectra)
<i>steps_to_go</i>	The number of steps to take data at during the measurement
<i>steps_wo_measurement</i>	The number of steps to go without taking data (before steps_to_go datapoints will be aquired)
<i>sec_angle</i>	The accesible angle-range will be shortened by this fixed offset angle on both sides
<i>sec_percentage</i>	The accesible angle-range will be shortened by this factor on both sides

8.5.2.3 end_scan

```
void Dinspect::end_scan ( ) [private], [slot]
```

end_scan stops finalizes the measurement process.

Stops the data acquisition from the detectors, resets variables, if option was set saves spectra and plots, if option was set sends UDP package

8.5.2.4 get_detector_data

```
void Dinspect::get_detector_data (
    double freq ) [private], [slot]
```

get_detector_data is a slot that acquires the frequency from the TTM-based counters

Parameters

<i>freq</i>	the acquired frequency
-------------	------------------------

PM100D-frequency is already acquired in [start_counting\(\)](#)

See also

[start_counting\(\)](#)

8.5.2.5 GetFrequencyFromSNSPDcounter

```
void Dinspect::GetFrequencyFromSNSPDcounter (
    double frequency ) [slot]
```

GetFrequencyFromSNSPDcounter receives the current frequency from the TTM based counter class and stores it in the appropriate vector.

Parameters

<i>frequency</i>	The event frequency of the current datapoint.
------------------	---

8.5.2.6 GetPowerFromPM100D

```
void Dinspect::GetPowerFromPM100D ( ) [slot]
```

GetPowerFromPM100D reads the current power value from the [PM100D](#), checks it and stores it in the appropriate vector.

In case the power value returned from the pm100d class is negative (an error value), a variable will be set to indicate the failure and open possibilities for counteractions (e.g. stop the scan and retry the acquisition).

See also

[scan_loop\(\)](#)
[PM100D::getPower\(\)](#)

8.5.2.7 getSpectrometerSubversion()

```
void Dinspect::getSpectrometerSubversion ( ) [private]
```

getSpectrometerSubversion starts the determination process of the spectrometer subversion

Tells the motor class to listen for the spectrometer subversion and sends the command to return the content of the corresponding memory position of the PCBMotor controller over the serial connection.

8.5.2.8 getSpectrometerType()

```
void Dinspect::getSpectrometerType ( ) [private]
```

getSpectrometerType starts the determination process of the spectrometer version

Tells the motor class to listen for the spectrometer version and sends the command to return the content of the corresponding memory position of the PCBMotor controller over the serial connection.

8.5.2.9 GraphSelectionChanged

```
void Dinspect::GraphSelectionChanged ( ) [private], [slot]
```

GraphSelectionChanged sets the actions for different events based on which part of the graph is selected.

For example, if the x-Axis is selected, then the mousewheel controls zooming of the x-Axis. If, however, the y-Axis is selected, then the mousewheel controls zooming of the y-Axis. And so on.

8.5.2.10 initActionsConnections()

```
void Dinspect::initActionsConnections ( ) [private]
```

initActionsConnections establishes the Qt signal-slot connections

These connections are used e.g. to specify what the buttons/other controls of the GUI do and to establish interfaces between classes.

8.5.2.11 motor_is_at_starting_position

```
void Dinspect::motor_is_at_starting_position ( ) [slot]
```

motor_is_at_starting_position

Receives a signal from the motor if it is positioned at the (offset) starting position. Depending on whether it is a calibration- of measurement scan, it will continue appropriately.

8.5.2.12 motorfoundfallingedge

```
void Dinspect::motorfoundfallingedge ( ) [slot]
```

motorfoundfallingedge called when a falling edge is detected in the PCBMotor encoder wheel sensor data during a measurement (not during initial positioning)

If the appropriate option is set, it gathers the type of the edge and at which step it appeared. In every case, it starts the (re)calculation of several quantities (stepsize, wavelength)

See also

[motorfoundrisingedge\(\)](#)
[ActionAfterEdgeDetection\(\)](#)

8.5.2.13 motorfoundinitedge

```
void Dinspect::motorfoundinitedge (
    bool rising,
    int stepstoedge ) [private], [slot]
```

motorfoundinitedge called during initial positioning when the motor found an edge

Parameters

<i>rising</i>	
<i>stepstoedge</i>	It check whether the type of the edge which was found (rising/falling) matches the type which shall be used (rising/falling/both). Depending on the type of scan (calibration/measurement), it calls the corresponding scan-loops.

8.5.2.14 motorfoundrisingedge

```
void Dinspect::motorfoundrisingedge ( ) [slot]
```

motorfoundrisingedge called when a rising edge is detected in the PCBMotor encoder wheel sensor data during a measurement (not during initial positioning)

If the appropriate option is set, it gathers the type of the edge and at which step it appeared. In every case, it starts the (re)calculation of several quantities (stepsize, wavelength)

See also

[motorfoundfallingedge\(\)](#)
[ActionAfterEdgeDetection\(\)](#)

8.5.2.15 on_DetectorComboBox_activated

```
void Dinspect::on_DetectorComboBox_activated (
    const QString & arg1 ) [private], [slot]
```

on_DetectorComboBox_activated Detects which Detector the user wants to use.

Parameters

<i>arg1</i>	a string containing an identifies of the detector (e.g. "TTM8000")
-------------	--

8.5.2.16 plot_savefile

```
void Dinspect::plot_savefile (
    uint16_t savefileversion ) [private], [slot]
```

plot_savefile plots data from a saved spec-file

Parameters

<i>savefileversion</i>	the version of the savefile structure
------------------------	---------------------------------------

8.5.2.17 recalc_with_dynamic_stepsize()

```
void Dinspect::recalc_with_dynamic_stepsize ( ) [private]
```

recalc_with_dynamic_stepsize recalculates several quantities once an edge was detected.

During a measurement, after an edge in the PCBMotor encoder wheel sensor data was found, recalculates affected quantities in this order:

Recalculates the wavelength at a given step (using the average stepsize since the last edge).

Then, recalculates how many steps are in each bandpass.

Finally, recalculates the frequency in each bandpass.

See also

[ActionAfterEdgeDetection\(\)](#)

8.5.2.18 save()

```
void Dinspect::save (
    bool automatic = false ) [private]
```

save saves the data acquired during a measurement to a file

Parameters

<i>automatic</i>	does not open a file-dialog but determines filename automatically if true
------------------	---

8.5.2.19 saveAutomatically

```
void Dinspect::saveAutomatically ( ) [private], [slot]
```

saveAutomatically saves spectrum and plot automatically when a measurement is finished

The automatic save happens both in the case when the upper wavelength bound was reached as well as when a measurement was stopped.

8.5.2.20 SaveGraph

```
void Dinspect::SaveGraph (
    bool automatic = false ) [private], [slot]
```

SaveGraph saves a plot.

Parameters

<i>automatic</i>	does not open a file-dialog but determines filename automatically if true (used for autosave)
------------------	---

8.5.2.21 ScanRangeMinPositioning

```
void Dinspect::ScanRangeMinPositioning ( ) [private], [slot]
```

ScanRangeMinPositioning positions the motor to select a wavelength below or at the lower scan range bound.

As the stepsize is not constant, the number of steps to a given wavelength can't be predicted But something close to that can be done:

Guess if the desired wavelength will be between starting position (sp) and (sp+ 1/8 °), (or 1/16 °, respectively) (one edge or two edges (halfstep/fullstep)) if yes: scan from the beginning if no:

- skip edges
- guess if the desired wavelength will be between the current position and (+1/8 ° or +1/16 °)
- if yes scan; if no go to next edge and repeat

8.5.2.22 setPM100D_AverageCount

```
void Dinspect::setPM100D_AverageCount (
    int newAvCnt ) [slot]
```

setPM100D_AverageCount calls the function to set the number of samples the powermeter averages over

Parameters

<i>newAvCnt</i>	the number of samples to average over
-----------------	---------------------------------------

The [PM100D](#) is able to internally average over a specific number of 'samples', where one sample is a measurement that takes approximately 3ms.

8.5.2.23 setupGraph()

```
void Dinspect::setupGraph (
    QCustomPlot * graph )
```

setupGraph Sets up appearance and behaviour of the graph.

Parameters

<i>graph</i>	The graph to be set up.
--------------	-------------------------

8.5.2.24 showActivityInfo

```
void Dinspect::showActivityInfo (
    const QString & message ) [slot]
```

showActivityInfo updates a label in the UI.

Parameters

<i>message</i>	The message to be displayed.
----------------	------------------------------

8.5.2.25 showCurrentMaxInfo

```
void Dinspect::showCurrentMaxInfo (
    double maxwl ) [slot]
```

showCurrentMaxInfo updates a label stating where the maximum was found

Parameters

<i>maxwl</i>	The wavelength at which the maximum appeared
--------------	--

8.5.2.26 showCurrentWLInfo

```
void Dinspect::showCurrentWLInfo (
    double wl ) [slot]
```

showCurrentWLInfo updates a label stating up to which wavelength the scan progressed, so far.

Parameters

<i>wl</i>	The wavelength up to which the measurement progressed so far.
-----------	---

8.5.2.27 showDataInfo

```
void Dinspect::showDataInfo (
    const QString & message ) [slot]
```

showDataInfo updates a label showing the newest aquired frequency/power.

Parameters

<i>message</i>	The message to be displayed.
----------------	------------------------------

8.5.2.28 showPositionInfo

```
void Dinspect::showPositionInfo (
    const QString & message ) [slot]
```

showPositionInfo updates a label stating what the spectrometer is currently doing (Aquiring data? Stopping? Positioning? Waiting for something?...).

Parameters

<i>message</i>	The message to be displayed.
----------------	------------------------------

It is also used by the motor class to display message.

8.5.2.29 showStatusMessage

```
void Dinspect::showStatusMessage (
    const QString & message ) [slot]
```

showStatusMessage updates a label showing the current status of the serial connection (Connected, disconnected, errorcode, ...).

Parameters

<i>message</i>	The message to be displayed.
----------------	------------------------------

8.5.2.30 SNSPD_update_channel

```
void Dinspect::SNSPD_update_channel (
    int getchannel ) [private], [slot]
```

SNSPD_update_channel called when the TTM channel was changel. Updates the snspdcounter and the coincidencecounter with the new channel.

Parameters

<i>getchannel</i>	the new channel
-------------------	-----------------

8.5.2.31 SNSPD_update_coinc_channel

```
void Dinspect::SNSPD_update_coinc_channel (
    int getchannel ) [private], [slot]
```

SNSPD_update_coinc_channel called when the TTM coincidence channel was changel. Updates the snspdcounter and the coincidencecounter with the new channel.

Parameters

<i>getchannel</i>	the new channel
-------------------	-----------------

8.5.2.32 start_detectors()

```
void Dinspect::start_detectors ( ) [private]
```

start_detectors starts the detectors

If [PM100D](#) is selected, this function does nothing.\ If TM8000 is selected, establishes a connection to the TTM.

8.5.2.33 TTM_IP_changed

```
void Dinspect::TTM_IP_changed (
    QString ip ) [private], [slot]
```

TTM_IP_changed changes the IP which is used to listen for ttm-packages.

Parameters

<i>ip</i>	the IP
-----------	--------

8.5.2.34 TTM_Port_changed

```
void Dinspect::TTM_Port_changed (
    int port ) [private], [slot]
```

TTM_Port_changed changes the port which is used to listen for ttm-packages.

Parameters

<i>port</i>	the port
-------------	----------

8.5.2.35 UDPCOMsetdetector

```
void Dinspect::UDPCOMsetdetector (
    QString det ) [private], [slot]
```

UDPCOMsetdetector changes the detector for the measurement. . Called when the [UDPlistener](#) receives the command to change the detector.

Parameters

<i>det</i>	detector identification string (e.g. "PM100D" or "TTM8000")
------------	---

8.5.2.36 UDPCOMsetscanrange

```
void Dinspect::UDPCOMsetscanrange (
    double low,
    double high ) [private], [slot]
```

UDPCOMsetscanrange Sets the scanrange. Called when the [UDPlistener](#) receives the command to change the scanrange.

Parameters

<i>low</i>	starting wavelength
<i>high</i>	stop wavelength

8.5.2.37 UDPCOMsetttmchannel

```
void Dinspect::UDPCOMsetttmchannel (
    int channel ) [private], [slot]
```

UDPCOMsetttmchannel Changes the TTM channel. Called when the [UDPlistener](#) receives the command to change the TTM channel in use.

Parameters

<i>channel</i>	the TTM stop channel to change to.
----------------	------------------------------------

8.5.2.38 UDPCOMsetttmcoincidencechannel

```
void Dinspect::UDPCOMsetttmcoincidencechannel (
    int channel ) [private], [slot]
```

UDPCOMsetttmcoincidencechannel Changes the TTM coincidence channel. Called when the [UDPlistener](#) receives the command to change the TTM coincidence channel in use.

Parameters

<i>channel</i>	the TTM stop channel to change to.
----------------	------------------------------------

8.5.2.39 update_spectrometer_info

```
void Dinspect::update_spectrometer_info (
    uint16_t type,
    uint16_t subversion ) [slot]
```

update_spectrometer_info receives the major and minor version of the connected spectrometer from the motor class and updates the UI to show it.

Parameters

<i>type</i>	The major version of the connected spectrometer.
<i>subversion</i>	The minor version of the connected spectrometer.

8.5.2.40 updateIntegrationtime

```
void Dinspect::updateIntegrationtime (
    int newinttime ) [private], [slot]
```

updateIntegrationtime gets the time over which the frequency is aquired (using TTM-based counters)

Parameters

<i>newinttime</i>	the integrationtime
-------------------	---------------------

The documentation for this class was generated from the following files:

- [dinspect.h](#)
- [dinspect.cpp](#)
- [dinspect_calibration.cpp](#)
- [dinspect_connections.cpp](#)
- [dinspect_io.cpp](#)
- [dinspect_scan.cpp](#)
- [dinspect_udp.cpp](#)
- [dinspect_ui.cpp](#)
- [dinspect_vars.cpp](#)

8.6 readPM100Dlin.dinspectPM100D Class Reference

Public Member Functions

- def **__init__** (self)
- def **read** (self)
- def **getAverageCount** (self)
- def **setAverageCount** (self, cnt)

The documentation for this class was generated from the following file:

- [readPM100Dlin.py](#)

8.7 readPM100Dwin.dinspectPM100D Class Reference

Public Member Functions

- def **__init__** (self)
- def **read** ()
- def **getAverageCount** ()
- def **setAverageCount** (cnt)
- def **setSerialNumber** (srn)

Public Member Functions

- **ManualMotorControlDialog** (QWidget *parent=0)
- void [Connect_to_motor](#) (Motor *getmotor)
Connect_to_motor Receives acces to the PCBMotor serial connection.

Private Slots

- void [invalmicroPbwchanged](#) (int microp)
updates variables according to user input
- void [invalmicroPfwchanged](#) (int microp)
updates variables according to user input
- void [invalstepbwchanged](#) (int s)
updates variables according to user input
- void [invalstepfwchanged](#) (int s)
updates variables according to user input
- void [invalWLchanged](#) (double wl)
updates variables according to user input
- void [invalCCchanged](#) ()
updates variables according to user input
- void [invalfastpositioningchanged](#) (int getfast)
updates variables according to user input

Private Member Functions

- void [sendcustomcommand](#) ()
sendcustomcommand lets the motor carry out a custom command
- void [gomicropulsesforward](#) ()
gomicropulsesforward lets the motor apply a number of micropulses in forward direction
- void [gomicropulsesbackwards](#) ()
gomicropulsesbackwards lets the motor apply a number of micropulses in backward direction
- void [gostepsforward](#) ()
gostepsforward lets the motor go a number of steps in forward direction
- void [gostepsbackwards](#) ()
gostepsbackwards lets the motor go a number of steps in backward direction
- void [gotolambda](#) ()
gotolambda lets the motor position approximately to a specific wavelength
- void [findnextrisingedge](#) ()
findnextrisingedge lets the motor find the next rising edge
- void [findnextfallingedge](#) ()
findnextfallingedge lets the motor find the next falling edge
- void [gotostop](#) ()
gotostop lets the motor go to the mechanical stop
- void [quit](#) ()
quit closes the dialog
- void [sendsignalfallingedge detected](#) ()
sendsignalfallingedge detected lets the motor emit the signal that a falling edge was found (even though that did not happen)
- void [sendsignalrisingedge detected](#) ()
sendsignalrisingedge detected lets the motor emit the signal that a rising edge was found (even though that did not happen)
- void [initconnections](#) ()
initconnections specifies which button calls which function

Private Attributes

- `Ui::ManualMotorControlDialog * ui`
- `Motor * motor`
The serial connection to the PCBMotor currently in use, where all the movements are implemented.
- `double desiredWL`
The wavelength the user wants to position to.
- `uint16_t microPfw`
The number of micropulses the motor shall apply in forward direction.
- `uint16_t microPbw`
The number of micropulses the motor shall apply in backward direction.
- `uint16_t Sfw`
The number of steps the motor shall go in forward direction.
- `uint16_t Sbw`
The number of steps the motor shall go in backward direction.
- `QString customcommand`
The custom command to send to the motor.
- `bool fastpositioning`
True if fast but inaccurate positioning to the desired wavelength is wanted.

8.8.1 Detailed Description

The `ManualMotorControlDialog` class offers the ability to manually operate the PCBMotor.

Bug There are no known bugs.

Sometimes, it is convenient to rotate the PCBMotor the grating is mounted on manually. One might want to find the maximum in the output power before recouping or for polatization optimization. This class offers the ability to do so using an easy-to-use dialog. In addition, it enables a fast(er) way to roughly select a given wavelength.

8.8.2 Member Function Documentation

8.8.2.1 `Connect_to_motor()`

```
void ManualMotorControlDialog::Connect_to_motor (
    Motor * getmotor )
```

`Connect_to_motor` Receives acces to the PCBMotor serial connection.

Parameters

<code>getmotor</code>	The, already established, PCBMotor connection
-----------------------	---

The documentation for this class was generated from the following files:

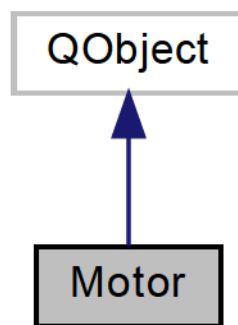
- [manualmotorcontroldialog.h](#)
- [manualmotorcontroldialog.cpp](#)

8.9 Motor Class Reference

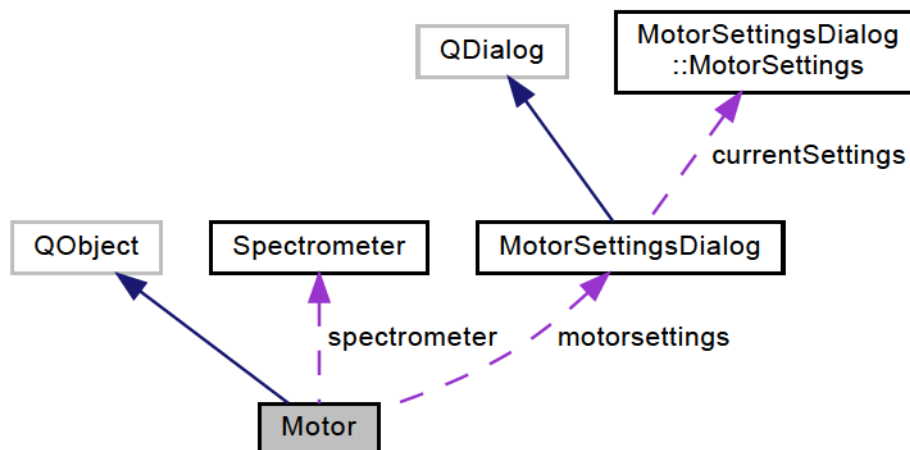
The [Motor](#) class operates the PCB-motor.

```
#include <motor.h>
```

Inheritance diagram for Motor:



Collaboration diagram for Motor:



Public Slots

- `void open ()`
open establishes a connection over a serial port
- `void close ()`
close closes the serialport connection
- `void read ()`
read reads from the serial port
- `void write (const QByteArray &data)`

- write writes to the serialport*
- void [handleError](#) (QSerialPort::SerialPortError error)
 - handleError prints an error message of the serialport connection and closes the connection*
- void [showStatusMessage](#) (const QString &message)
 - showStatusMessage fills the label in the GUI with text*
- void [opensettings](#) ()
 - opensettings opens the serial connection settings dialog*
- bool [isopen](#) ()
 - isopen returns the state of the serial connection*
- void [updateAskmotor](#) (int getaskmotor)
 - updateAskmotor is called to change the specification of which information sent by the motor is currently of interest.*
- void [updateScanIsRunning](#) (bool getscanrunning)
 - updateScanIsRunning*
- void [updateIsFirstStep](#) (bool getfirststep)
 - updateIsFirstStep*
- void [getSpectrometerSettings](#) (Spectrometer *spec)
 - getSpectrometerSettings receives a spectrometer configuration struct*
- void [command_park](#) ()
 - command_park moves the motor to the mechanical stop*
- void [command_to_start_pos](#) ()
 - command_to_start_pos moves the motor to the mechanical stop and then to the (offset) starting position using one command*
- void [go_from_stop_to_startpos_with_singlecmds](#) ()
 - go_from_stop_to_startpos_with_singlesteps moves the motor to the (offset) starting position using one command per step*
- void [command_roughly_to_wl](#) ()
 - command_roughly_to_wl Positions the motor roughly at a certain position*
- void [get_num_of_edges_to_skip](#) ()
 - get_num_of_edges_to_skip finds out how many edges can be skipped for positioning to a certain wavelength*
- void [get_rough_wl](#) (double getwl)
 - get_rough_wl receives the wavelength the user wants to position to*
- void [command_home](#) ()
 - command_home sends commands to position at the mechanical stop and afterwards go to the offset starting position, but in an inaccurate way*
- void [command_info](#) ()
 - command_info sends the command to request the PCBMotor information*
- void [command_help](#) ()
 - command_help sends the command to print the PCBMotor help*
- void [command_frequency_sweep](#) ()
 - command_frequency_sweep sends the PCBMotor command for a frequency sweep*
- void [command_singlestep](#) (QString dirstring)
 - command_singlestep moves the motor a single step in a direction specified by dirstring*
- void [command_step](#) (uint16_t numsteps, QString dirstring)
 - command_step moves the motor numstep steps in a direction specified by dirstring*
- void [command_microstep](#) (uint16_t nummsteps, QString dirstring)
 - command_microstep applies nummsteps micropulses to the motor*
- void [command_scanmove](#) (bool realignment)
 - command_scanmove executes the scanmove command specified in the spectrometer configuration file*
- void [command_scanmove_realignment](#) ()
 - command_scanmove_realignment undescribed*
- void [gotonextedge](#) ()

- gotonextedge finds the next edge (slow)*

 - void `skip_edges` (int num, bool both, bool next, bool onlyrising)
skip_edges initializes parameters to skip initial edges before scan
 - void `skip_edges_rough` (int num, bool both, bool next, bool onlyrising)
skip_edges_rough initializes parameters to skip edges for rough positioning to a certain wavelength
 - void `start_skipping_edges` ()
start_skipping_edges Skips initial edges for scan (slow)
 - void `start_skipping_edges_rough` ()
start_skipping_edges_rough Skips edges for rough positioning to a certain wavelength (fast)
 - void `command_nextedge` (bool rising)
command_nextedge Goes to the next edge (fast)
 - void `stop` (bool stop)
stop Tries to stop movement if possible
 - void `getdsettings` (QSettings *insettings)
getdsettings receives a pointer to the dinspect qsettings instance.
 - void `sendsignalrisingedge detected` ()
sendsignalrisingedge detected emits the signal `risingedge()`
 - void `sendsignalfallingedge detected` ()
sendsignalfallingedge detected emits the signal `fallingedge()`

Signals

- void `motorstatusmessage` (const QString &message)
motorstatusmessage emitted when the status of the serial connection changes, with a string indicating the actual state.
- void `spectrometer_info_updated` (uint16_t type, uint16_t subversion)
spectrometer_info_updated emitted when a connection to a new spectrometer is established. Transmits information about the identity of the spectrometer.
- void `risingedge` ()
risingedge emitted when a rising edge was found (during measurement)
- void `fallingedge` ()
fallingedge emitted when a falling edge was found (during measurement) While the measurement is running, `risingedge()` and `fallingedge()` are emitted whenever an edge is detected In contrast, the signal `edgewasfound()` is used during initial positioning to the starting position.
- void `edgewasfound` (bool rising, int stepstoedge)
edgewasfound emitted when an edge was found (during initial positioning)
- void `skipped_init_edges` ()
skipped_init_edges emitted when the first edges are skipped (to position close to scanrangemin)
- void `skipped_edges_rough` ()
skipped_edges_rough emitted when edges for rough positioning are skipped
- void `UpdatePositionInfo` (QString)
UpdatePositionInfo emitted when motor-class wants to change a label in the GUI. The argument of the signal is the new message.
- void `at_starting_position` ()
at_starting_position is emitted when motor is positioned at offset starting position
- void `ConnectionClosed` ()
emitted when serial connection is closed

Public Member Functions

- [Motor](#) ()
Motor the constructor initializes variables and establishes the serial connection.
- bool [sensordata](#) ()
sensordata returns the current PCBMotor optical encoder wheel sendor state
- bool [isfallingedge](#) ()
isfallingedge returns if a falling edge was just found
- bool [isrisingedge](#) ()
isrisingedge returns if a rising edge was just found
- bool [isedge](#) ()
isedge returns if an edge was just found

Public Attributes

- QString [publicmotorstatusmessage](#)
A string containing the current state of the serial connection.
- QSerialPort * [serial](#)
Qt serial connection interface.

Private Member Functions

- void [CreatePositionLabel](#) ()
CreatePositionLabel updates the GUI-label during initial positioning.

Private Attributes

- [MotorSettingsDialog](#) * [motorsettings](#)
the current PCBMotor settings.
- [Spectrometer](#) * [spectrometer](#)
The current spectrometer.
- QSettings * [dsettings](#)
The dinspect settings.
- bool [serialconnectionok](#)
Set to false if there is a problem with the serial connection.
- int [askmotor](#)
Used to specify which information read from the serial connection (sent by the motor) is currently important.
- uint16_t [SpectrometerType](#)
First part of the identification of the currently connected spectrometer.
- uint16_t [SpectrometerSubversion](#)
Second part of the identification of the currently connected spectrometer.
- QTimer [hometimer](#)
Used to iterate through the steps of 'go to the starting position' - but in an inaccurate way.
- QTimer [startpostimer](#)
Used to iterate through the steps of 'go to the starting position'.
- QTimer [roughwltimer](#)
- QTimer [gotonextedgetimer](#)
Used to iterate through the steps of 'going to the next edge'.
- QTimer [Labeltimer](#)

- Updates the label showing the time left until the starting position is reached.*

 - QTimer [skippedgetimer](#)

For initial positioning before scan starts.

- QTimer [skippedgetimerrough](#)

For positioning roughly to wavelength.

- QTimer [startpossinglecmdtimer](#)

For positioning at (offset) starting position with single commands.

- int [skippededge_current](#)

Current edge which is skipped.

- int [skippededge_num](#)

Edges to skip.

- bool [skippededge_both](#)

Use both edge types?

- bool [skippededge_next](#)

Type of next edge to skip.

- bool [skippededge_onlyrising](#)

If using only one edge-type: which?

- bool [skippededge_in_progeess](#)

True if the process of skipping the initial edges is currently in progress.

- bool [skippededge_before_last_step](#)

True if the progress of skipping the initial edges came to its last edge. (The last one has to be done slow but more accurately again)

- bool [stopnow](#)

Set to true when 'Stop scan' button is clicked.

- int [startpossinglecmd_stepsgone](#)

During initial positioning, counts the steps already gone (to know when to stop)

- int [stepstofirstedge](#)

Holds the number of movements that were done before the first edge was found.

- bool [sensorwashigh](#)

True if the state of the motor encoder wheel sensor was 'high' in the last step.

- bool [sensorishigh](#)

True if the state of the motor encoder wheel sensor is 'high' right now.

- bool [scanisrunning](#)

True if a measurement is currently in progress.

- bool [isfirststep](#)

True only for the first step in a measurement.

- bool [foundedge](#)

True if edge was just found.

- bool [foundrisingedge](#)

True if edge was just found.

- bool [foundfallingedge](#)

True if falling edge was just found.

- bool [gotonextedge_inprogress](#)

True if rising edge was just found.

- uint8_t [homestep](#)

Used to decide which step in the initial positioning process comes next (inaccurate version)

- uint8_t [command_to_start_pos_iteration](#)

Used to decide which step in the initial positioning process comes next.

- uint8_t [command_roughly_to_wl_iteration](#)

Used to decide which step in the process of positioning approximately to a specific wavelength comes next.

- uint8_t [startpossinglecmditeration](#)

Used to decide which step in the initial positioning process comes next (single command version)

- `uint64_t seconds_over`

Counts seconds. Used for one method of determining if the initial starting position is reached. (Method: send one command specified in config file and wait for a certain amount of time, specified in config file too)

- `double roughwl`

Wavelength the shall roughly be positioned to.

- `uint16_t num_edges_to_skip_rough`

Number of edges which can be skipped to position roughly to Wavelength "roughwl".

8.9.1 Detailed Description

The `Motor` class operates the PCB-motor.

Bug There are no known bugs.

The PCBMotor is controllable by sending ASCII commands over a serial connection. This class establishes such a connection and controls the movements of the motor.

8.9.2 Member Function Documentation

8.9.2.1 `command_microstep`

```
void Motor::command_microstep (
    uint16_t nummsteps,
    QString dirstring ) [slot]
```

`command_microstep` applies nummsteps micropulses to the motor

Parameters

<i>nummsteps</i>	number of micropulses to apply
<i>dirstring</i>	string containing the desired direction

`dirstring` may either be "bw" of "fw" for backward/forward movement.

8.9.2.2 `command_nextedge`

```
void Motor::command_nextedge (
    bool rising ) [slot]
```

`command_nextedge` Goes to the next edge (fast)

Parameters

<i>rising</i>	True: look for rising edge. False: look for falling edge.
---------------	---

It uses the motor control driver built-in command to find the next edge in the encoder wheel signal. Fast, but not very accurate (depending on motor operation parameters)

8.9.2.3 `command_roughly_to_wl`

```
void Motor::command_roughly_to_wl ( ) [slot]
```

`command_roughly_to_wl` Positions the motor roughly at a certain position

Goes back to mechanical stop. Then positions at offset starting position, but without waitingtime between single steps to be faster. Finds the next edge (fast) and skips edges (fast too).

8.9.2.4 `command_scanmove`

```
void Motor::command_scanmove (
    bool realignment ) [slot]
```

`command_scanmove` executes the scanmove command specified in the spectrometer configuration file

Parameters

<i>realignment</i>	unused
--------------------	--------

The scanmove command is a motor movement, usually a specific number of micropulses to apply to the motor.

8.9.2.5 `command_singlestep`

```
void Motor::command_singlestep (
    QString dirstring ) [slot]
```

`command_singlestep` moves the motor a single step in a direction specified by `dirstring`

Parameters

<i>dirstring</i>	a string containing the desired movement direction
------------------	--

`Dirstring` may either be "bw" or "fw" for backward/forward movement.

8.9.2.6 `command_step`

```
void Motor::command_step (
    uint16_t numsteps,
    QString dirstring ) [slot]
```

`command_step` moves the motor `numstep` steps in a direction specified by `dirstring`

Parameters

<i>numsteps</i>	number of steps to go
<i>dirstring</i>	direction to go

Dirstring may either be "bw" or "fw" for backward/forward movement.

8.9.2.7 `command_to_start_pos`

```
void Motor::command_to_start_pos ( ) [slot]
```

`command_to_start_pos` moves the motor to the mechanical stop and then to the (offset) starting position using one command

This code creates a single command to bring the motor from the mechanical stop to the starting position. The motor can't be stopped during this time, making early measurement-stops a pain. Therefore, a timer-based solution is implemented below. Both options are still here (choosable via the settings-dialog), because the single-step-way still has to be tested.

See also

[go_from_stop_to_startpos_with_singlecmds\(\)](#)

8.9.2.8 `CreatePositionLabel()`

```
void Motor::CreatePositionLabel ( ) [private]
```

`CreatePositionLabel` updates the GUI-label during initial positioning.

During initial positioning, this function is used to create/update the string which is displayed in the GUI. Out of the waitingtime specified in the configuration file and a counter which counts the seconds since the procedure was started (`seconds_over`), it creates a string stating the time left till the end of the process.

See also

[seconds_over](#)

8.9.2.9 `edgewasfound`

```
void Motor::edgewasfound (
    bool rising,
    int stepstoedge ) [signal]
```

`edgewasfound` emitted when an edge was found (during initial positioning)

Parameters

<i>rising</i>	true if the edge found is rising, false if it is falling
<i>stepstoedge</i>	number of movements that were needed to go to this edge

Prior to starting the measurement, during the process of finding the starting position, this signal is emitted when the first edge is found. In contrast, [risingedge\(\)](#) and [fallingedge\(\)](#) are emitted during the measurement process.

See also

[risingedge\(\)](#)
[fallingedge\(\)](#)

8.9.2.10 fallingedge

```
void Motor::fallingedge ( ) [signal]
```

fallingedge emitted when a falling edge was found (during measurement) While the measurement is running, [risingedge\(\)](#) and [fallingedge\(\)](#) are emitted whenever an edge is detected In contrast, the signal [edgewasfound\(\)](#) is used during initial positioning to the starting position.

See also

[fallingedge\(\)](#)
[edgewasfound\(\)](#)

8.9.2.11 get_rough_wl

```
void Motor::get_rough_wl (
    double getwl ) [slot]
```

get_rough_wl receives the wavelength the user wants to position to

Parameters

<i>getwl</i>	Input: Wavelength to be roughly positioned to
--------------	---

8.9.2.12 getdsettings

```
void Motor::getdsettings (
    QSettings * insettings ) [slot]
```

getdsettings receives a pointer to the dinspect qsettings instance.

Parameters

<i>insettings</i>	pointer to dinspect settings
-------------------	------------------------------

8.9.2.13 `getSpectrometerSettings`

```
void Motor::getSpectrometerSettings (
    Spectrometer * spec ) [slot]
```

`getSpectrometerSettings` receives a spectrometer configuration struct

Parameters

<i>spec</i>	the spectrometer configuration struct
-------------	---------------------------------------

8.9.2.14 `gotonextedge`

```
void Motor::gotonextedge ( ) [slot]
```

`gotonextedge` finds the next edge (slow)

Applies micropulses and checks the state of the encode wheel signal after every step to detect edges. The waiting-time between the micropulses makes it slow, but it is more accurate than the control driver built-in function. Emits signals when an edge was found.

See also

[edgewasfound\(\)](#)

8.9.2.15 `handleError`

```
void Motor::handleError (
    QSerialPort::SerialPortError error ) [slot]
```

`handleError` prints an error message of the serialport connection and closes the connection

Parameters

<i>error</i>	
--------------	--

8.9.2.16 isedge()

```
bool Motor::isedge ( )
```

isedge returns if an edge was just found

Returns

if an edge was just found

8.9.2.17 isfallingedge()

```
bool Motor::isfallingedge ( )
```

isfallingedge returns if a falling edge was just found

Returns

if the just found edge is falling

8.9.2.18 isopen

```
bool Motor::isopen ( ) [slot]
```

isopen returns the state of the serial connection

Returns

true if serial connection was established successfully, false otherwise

8.9.2.19 isrisingedge()

```
bool Motor::isrisingedge ( )
```

isrisingedge returns if a rising edge was just found

Returns

true if the just found edge is rising

8.9.2.20 motorstatusmessage

```
void Motor::motorstatusmessage (
    const QString & message ) [signal]
```

motorstatusmessage emitted when the status of the serial connection changes, with a string indicating the actual state.

Parameters

<i>message</i>	the message
----------------	-------------

8.9.2.21 risingedge

```
void Motor::risingedge ( ) [signal]
```

risingedge emitted when a rising edge was found (during measurement)

While the measurement is running, [risingedge\(\)](#) and [fallingedge\(\)](#) are emitted whenever an edge is detected. In contrast, the signal [edgewasfound\(\)](#) is used during initial positioning to the starting position.

See also

[risingedge\(\)](#)
[edgewasfound\(\)](#)

8.9.2.22 sendsignalfallingedge detected

```
void Motor::sendsignalfallingedge detected ( ) [slot]
```

sendsignalfallingedge detected emits the signal [fallingedge\(\)](#)

See also

[fallingedge\(\)](#)

8.9.2.23 sendsignalrisingedge detected

```
void Motor::sendsignalrisingedge detected ( ) [slot]
```

sendsignalrisingedge detected emits the signal [risingedge\(\)](#)

See also

[risingedge\(\)](#)

8.9.2.24 sensordata()

```
bool Motor::sensordata ( )
```

sensordata returns the current PCBMotor optical encoder wheel sendor state

Returns

the current PCBMotor optical encoder wheel sendor state

8.9.2.25 showStatusMessage

```
void Motor::showStatusMessage (
    const QString & message ) [slot]
```

showStatusMessage fills the label in the GUI with text

Parameters

<i>message</i>	the text to be shown in the label
----------------	-----------------------------------

8.9.2.26 skip_edges

```
void Motor::skip_edges (
    int num,
    bool both,
    bool next,
    bool onlyrising = false ) [slot]
```

skip_edges initializes parameters to skip initial edges before scan

Parameters

<i>num</i>	Number of edges to skip
<i>both</i>	True: use both falling and rising edges
<i>next</i>	The next edge is rising(true) or falling(false)
<i>onlyrising</i>	if both is false, use either only rising(true) or falling(false) edges

8.9.2.27 skip_edges_rough

```
void Motor::skip_edges_rough (
    int num,
    bool both,
    bool next,
    bool onlyrising = false ) [slot]
```

skip_edges_rough initializes parameters to skip edges for rough positioning to a certain wavelength

Parameters

<i>num</i>	Number of edges to skip
<i>both</i>	True: use both falling and rising edges
<i>next</i>	The next edge is rising(true) or falling(false)
<i>onlyrising</i>	if both is false, use either only rising(true) or falling(false) edges

8.9.2.28 spectrometer_info_updated

```
void Motor::spectrometer_info_updated (
    uint16_t type,
    uint16_t subversion ) [signal]
```

spectrometer_info_updated emitted when a connection to a new spectrometer is established. Transmits information about the identity of the spectrometer.

Parameters

<i>type</i>	Output: the spectrometer type (e.g. 1550)
<i>subversion</i>	Output: the subversion of the spectrometer (e.g. 1)

In the memory of the PCBMotor control driver, a unique identifier of the device was stored during the manufacturing process. It consists out of the 'type' (e.g. 1550), and the 'subversion' (e.g. 1). Together they form an identifier like 1550v1.

8.9.2.29 stop

```
void Motor::stop (
    bool stop ) [slot]
```

stop Tries to stop movements if possible

Parameters

<i>stop</i>	Input: True if movements shall be stopped if possible
-------------	---

8.9.2.30 updateAskmotor

```
void Motor::updateAskmotor (
    int getaskmotor ) [slot]
```

updateAskmotor is called to change the specification of which information sent by the motor is currently of interest.

Parameters

<i>getaskmotor</i>	Valid values of the argument are: • 0: nothing • 1: getSpectrometerType • 2: getSpectrometerSubversion
--------------------	--

8.9.2.31 updateIsFirstStep

```
void Motor::updateIsFirstStep (
    bool getfirststep ) [slot]
```

updateIsFirstStep

Parameters

<i>getfirststep</i>	
---------------------	--

8.9.2.32 updateScanIsRunning

```
void Motor::updateScanIsRunning (
    bool getscanrunning ) [slot]
```

updateScanIsRunning

Parameters

<i>getscanrunning</i>	
-----------------------	--

8.9.2.33 write

```
void Motor::write (
    const QByteArray & data ) [slot]
```

write writes to the serialport

Parameters

<i>data</i>	data to be written to the serial port
-------------	---------------------------------------

8.9.3 Member Data Documentation

8.9.3.1 gotonextedgetimer

```
QTimer Motor::gotonextedgetimer [private]
```

Used to iterate through the steps of 'going to the next edge'.

See also

[gotonextedge\(\)](#)

8.9.3.2 hometimer

```
QTimer Motor::hometimer [private]
```

Used to iterate through the steps of 'go to the starting position' - but in an inaccurate way.

See also

[command_home\(\)](#)

8.9.3.3 roughwltimer

```
QTimer Motor::roughwltimer [private]
```

Used to iterate through the steps of 'going approximately to wavelength x'.

See also

[command_roughly_to_wl\(\)](#)

8.9.3.4 startpostimer

```
QTimer Motor::startpostimer [private]
```

Used to iterate through the steps of 'go to the starting position'.

See also

[command_to_start_pos\(\)](#)

The documentation for this class was generated from the following files:

- [motor.h](#)
- [motor.cpp](#)

8.10 MotorSettingsDialog::MotorSettings Struct Reference

The [MotorSettings](#) struct contains contains the information to successfully establish a serial connection to the P↔CBMotor.

```
#include <motor.h>
```

Public Attributes

- QString [name](#)
Name of the device.
- quint32 [baudRate](#)
The speed of the serial connection (std: 19200)
- QString [stringBaudRate](#)
The speed of the serial connection (as a string)
- QSerialPort::DataBits [dataBits](#)
The number of data bits (std: 8)
- QString [stringDataBits](#)
The number of data bits (as a string)
- QSerialPort::Parity [parity](#)
The type of parity used (std: none)
- QString [stringParity](#)
The type of parity used (as a string)
- QSerialPort::StopBits [stopBits](#)
The number of stop bits (std: 1)
- QString [stringStopBits](#)
The number of stop bits (as a string)
- QSerialPort::FlowControl [flowControl](#)
The type of flow control used (std: none)
- QString [stringFlowControl](#)
The type of flow control used (as a string)
- bool [localEchoEnabled](#)
True to enable local echo.

8.10.1 Detailed Description

The [MotorSettings](#) struct contains contains the information to successfully establish a serial connection to the P_{CB}Motor.

The documentation for this struct was generated from the following file:

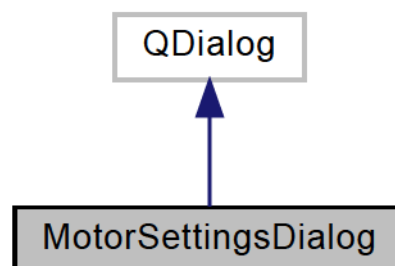
- [motor.h](#)

8.11 MotorSettingsDialog Class Reference

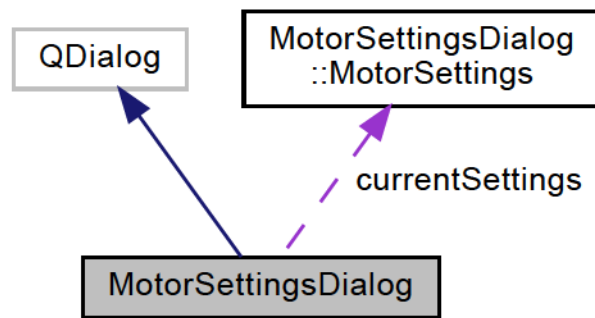
The [MotorSettingsDialog](#) class handles user-input to configure the connection to the PCB-motor.

```
#include <motor.h>
```

Inheritance diagram for MotorSettingsDialog:



Collaboration diagram for MotorSettingsDialog:



Classes

- struct [MotorSettings](#)

The [MotorSettings](#) struct contains contains the information to successfully establish a serial connection to the PC↔BMotor.

Public Member Functions

- **MotorSettingsDialog** (QWidget *parent=0)
- [MotorSettings](#) motorsettings () const

Public Attributes

- int [askmotor](#)

Private Slots

- void [showPortInfo](#) (int idx)
Fills the labels in the dialog with the current information about the serial connection.
- void [apply](#) ()

Private Member Functions

- void [fillPortsInfo](#) ()
Fills the dialog with information about all available devices.
- void [updateSettings](#) ()
Chooses a device.
- void [initsettings](#) ()
Initialized the settings to standard values which should satisfy the PCBMotors needs.

Private Attributes

- Ui::SettingsDialog * ui
- [MotorSettings](#) currentSettings
- QIntValidator * intValidator

8.11.1 Detailed Description

The [MotorSettingsDialog](#) class handles user-input to configure the connection to the PCB-motor.

Bug There are no known bugs.

This dialog should not be needed as the standard values should be fine.

8.11.2 Member Function Documentation

8.11.2.1 apply

```
void MotorSettingsDialog::apply ( ) [private], [slot]
```

Accepts the new serial connection properties and hides the dialog

8.11.3 Member Data Documentation

8.11.3.1 askmotor

```
int MotorSettingsDialog::askmotor
```

See also

[Motor::askmotor](#)

The documentation for this class was generated from the following files:

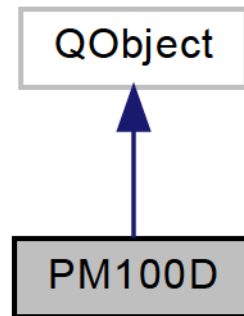
- [motor.h](#)
- [motor.cpp](#)

8.12 PM100D Class Reference

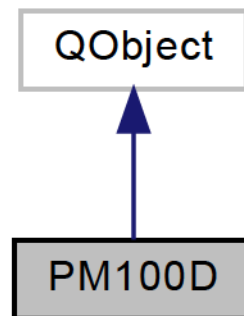
The [PM100D](#) class operates the Thorlabs [PM100D](#) powermeter.

```
#include <pm100d.h>
```

Inheritance diagram for PM100D:



Collaboration diagram for PM100D:



Public Slots

- void [pyprint](#) (const QString &str)
pyprint is a slot called by PythonQt signals which write to stdout and stderr

Public Member Functions

- double [getPower](#) ()
getPower aquires the current power measured with the powermeter in μW .
- int64_t [setAverageCount](#) (int cnt)
setAverageCount sets the number of samples to average over
- int64_t [getAverageCount](#) ()
getAverageCount reads the number of samples the powermeter averages over
- QString [setSerialNumber](#) (QString sn)
setSerialNumber sets the serial number of a [PM100D](#) powermeter

Private Attributes

- QString [path](#)
Path to the application.
- char * [apppath](#)
Path to the application.

8.12.1 Detailed Description

The [PM100D](#) class operates the Thorlabs [PM100D](#) powermeter.

Bug On Windows, `getPower` sometimes fails when using visa. Maybe a timing issue.

The [PM100D](#) class provides wrappers to the python module "ThorlabsPM100" to conveniently operate the Thorlabs [PM100D](#) powermeter from C++ without the strict need to run Windows.

8.12.2 Member Function Documentation

8.12.2.1 `getAverageCount()`

```
int64_t PM100D::getAverageCount ( )
```

`getAverageCount` reads the number of samples the powermeter averages over

Returns

the number of samples the powermeter averages over

The [PM100D](#) is able to internally average over a specific number of 'samples', where one sample is a measurement that takes approximately 3ms. This function reads the current number of samples the [PM100D](#) averages over.

8.12.2.2 `getPower()`

```
double PM100D::getPower ( )
```

`getPower` acquires the current power measured with the powermeter in μW .

Returns

The Power acquired from the powermeter in μW .

Sometimes, the data aquisition from the [PM100D](#) failes. This has so far only been observed on Windows (using VISA). In these cases, a negative value is returned to the caller.

8.12.2.3 `pyprint`

```
void PM100D::pyprint (
    const QString & str ) [slot]
```

`pyprint` is a slot called by PythonQt signals which write to stdout and stderr

Parameters

<i>str</i>	a string to print, coming from python
------------	---------------------------------------

8.12.2.4 setAverageCount()

```
int64_t PM100D::setAverageCount (
    int cnt )
```

setAverageCount sets the number of samples to average over

Parameters

<i>cnt</i>	number of samples to average over
------------	-----------------------------------

Returns

the number of samples the powermeter averages over

The [PM100D](#) is able to internally average over a specific number of 'samples', where one sample is a measurement that takes approximately 3ms. This function sets the current number of samples the [PM100D](#) averages over.

8.12.2.5 setSerialNumber()

```
QString PM100D::setSerialNumber (
    QString sn )
```

setSerialNumber sets the serial number of a [PM100D](#) powermeter

Parameters

<i>sn</i>	the serial number
-----------	-------------------

Returns

The serial number set

On Windows, the Python module ThorlabsPM100 uses VISA, which needs the serial number of the power meter to be able to connect to it. This function sets it.

The documentation for this class was generated from the following files:

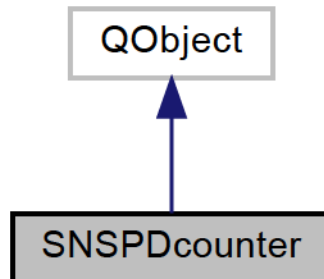
- [pm100d.h](#)
- [pm100d.cpp](#)

8.13 SNSPDcounter Class Reference

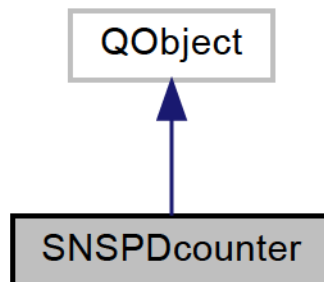
The [SNSPDcounter](#) class acquires single event frequencies from a TM8000.

```
#include <snspdcounter.h>
```

Inheritance diagram for SNSPDcounter:



Collaboration diagram for SNSPDcounter:



Public Member Functions

- **SNSPDcounter** (DataSource *gMainWnd, TTMViewOptions *gOptions, QWidget *parent=0)

8.13.1 Detailed Description

The [SNSPDcounter](#) class acquires single event frequencies from a TM8000.

This class is similar to the 'EventCount' class from the AIT sample program TTMViewer, but a lot of unnecessary functionality was removed. This class is no longer a QWidget and all GUI-parts were removed. Functions were added to interface with dinspect.

Author

Roland Lieger

Bug There are no known bugs.

The documentation for this class was generated from the following files:

- [snspdcounter.h](#)
- [snspdcounter.cpp](#)

8.14 Spectrometer Struct Reference

The [Spectrometer](#) struct holds spectrometer-specific variables.

```
#include <spectrometeroptions.h>
```

Public Attributes

- [QString](#) [config_fname](#)
Path and name of the spectrometer config file.
- [bool](#) [initialized](#)
True if loading of spectrometer config file was successful.
- [uint16_t](#) [id](#)
ID of configuration.
- [uint32_t](#) [Type](#)
Used to check if configuration file fits to connected spectrometer.
- [uint32_t](#) [Subversion](#)
Used to check if configuration file fits to connected spectrometer.
- [QString](#) [VersionString](#)
Stringified combination of Type and Subversion.
- [double](#) [fullstepangle](#)
Angle of a full step [microRad].
- [double](#) [grating_constant](#)
Grating constant in lines/mm.
- [double](#) [grating_width](#)
Ruled width of the grating in mm.
- [double](#) [mfdin](#)
Modefielddiameterer of the input fibre in μm .
- [double](#) [mfdout](#)
Modefielddiameterer of the output fibre in μm .
- [double](#) [nain](#)
Numerical Aperture of the input fibre.
- [double](#) [naout](#)
Numerical Aperture of the output fibre.
- [double](#) [fin](#)
Focal lengths of the input (collimating) mirror or lens in mm.
- [double](#) [fout](#)
Focal lengths of the output (focusing) mirror or lens in mm.
- [double](#) [lcutoffin](#)
Cutoff wavelength of the input (SM)-fibre in nm.
- [double](#) [lcutoffout](#)
Cutoff wavelength of the output (SM)-fibre in nm.
- [double](#) [deviation_angle](#)
Deviation angle in Rad.
- [double](#) [stepsize](#)
Scanmove stepsize in μRad .
- [double](#) [angle_offset](#)
Offset angle in Rad (angle between starting-position and angle where incident light is reflected back to the origin).
- [bool](#) [da_available](#)
true, if deviation_angle is available.

- bool [ss_available](#)
true, if stepsize is available.
- bool [calibrated](#)
true, if deviation_angle and stepsize are available.
- uint32_t [number_of_bps](#)
Number of bandpasses between LowerRangeBound and UpperRangeBound, used to check if calibration-file is corrupted.
- uint32_t [number_of_cepts](#)
Number of coupling efficiency data points, used to check if calibration-file is corrupted.
- uint32_t [number_of_sibp](#)
Number of StepsInBandpass datapoints, used to check if calibration-file is corrupted.
- uint32_t [number_of_was](#)
Number of WavelengthAtStep datapoints, used to check if calibration-file is corrupted.
- uint32_t [LowerRangeBound](#)
Lowest wavelength accessible to the spectrometer.
- uint32_t [UpperRangeBound](#)
Highest wavelength accessible to the spectrometer.
- uint32_t [stepmin](#)
Step at which LowerRangeBound appears.
- uint32_t [stepmax](#)
Step at which UpperRangeBound appears.
- uint16_t [offset_steps](#)
PCB-Motor: steps to go between mechanical STOP-position and 0° angle of incidence.
- uint16_t [offset_microsteps](#)
PCB-Motor: microsteps to go between mechanical STOP-position and 0° angle of incidence.
- uint32_t [steps_needed](#)
PCB-Motor: (μ)steps between scanrangeMin and scanrangeMax = number of datapoints in spectrum.
- uint16_t [steptime](#)
PCB-Motor: time to wait between 2 steps in scan-loop.
- uint16_t [gotostart_timebetweensteps](#)
PCB-Motor: time between single steps in command_go_to_start. Example: s-1,d250,x519: 250milliseconds.
- uint32_t [steps_before_realignment](#)
PCB-Motor: microstep-movements to be done before 'realignment'-move is necessary.
- QString [scanmove_command](#)
PCB-Motor: command for one scanmove. Example: "u-2\r".
- QString [startpos_command](#)
PCB-Motor: command to 1:go to mechanical stop and 2:go to start position.
- uint32_t [gotostart_waittime](#)
PCB-Motor: time it takes to 1:go to mechanical stop and 2:go to start position.
- uint32_t [gotostart_rough_waittime](#)
PCB-Motor: time it takes in fast (inaccurate) mode to go from mechanical stop to start position.
- QString [nextrisingedge_command](#)
PCB-Motor: command to rotate until next rising edge is found.
- QString [nextfallingedge_command](#)
PCB-Motor: command to rotate until next falling edge is found.
- uint16_t [nextedge_command_time](#)
PCB-Motor: time it takes to go to next edge with next(rising/falling)edge_command.
- uint16_t [parkcommand_waittime](#)
PCB-Motor: time it takes to do 'go to mechanical stop'.
- bool [userisingedge](#)
TRUE: Use only rising edge. FALSE: Use only falling edge.

- bool [usebothedges](#)
TRUE: Use both rising and falling edge. FALSE: Use only one edge.
- bool [calibratedwithgedetection](#)
TRUE if edge-detection was used during calibration.

8.14.1 Detailed Description

The [Spectrometer](#) struct holds spectrometer-specific variables.

Bug There are no known bugs.

The documentation for this struct was generated from the following file:

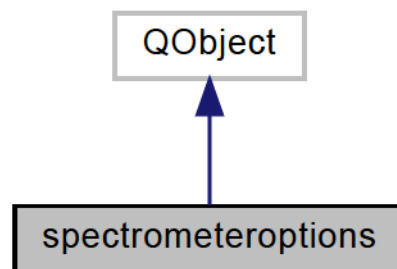
- [spectrometeroptions.h](#)

8.15 spectrometeroptions Class Reference

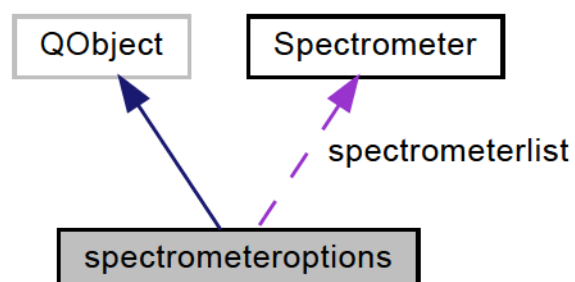
The spectrometeroptions class handles spectrometer-configurationfiles.

```
#include <spectrometeroptions.h>
```

Inheritance diagram for spectrometeroptions:



Collaboration diagram for spectrometeroptions:



Public Slots

- void [write_config](#) ([Spectrometer](#) *spectrometer, QVector< double > *WavelengthAtBP, QVector< double > *EfficiencyAtBP, QVector< int > *StepsInBP, QVector< double > *WavelengthAtStep)
write_config write a configuration to a file
- bool [read_bp_data](#) ([Spectrometer](#) *spectrometer, QVector< double > *WavelengthAtBP)
read_bp_data undescribed
- bool [read_ce_data](#) ([Spectrometer](#) *spectrometer, QVector< double > *EfficiencyAtBP)
read_ce_data undescribed
- bool [read_stepsinbp_data](#) ([Spectrometer](#) *spectrometer, QVector< int > *StepsInBP)
read_stepsinbp_data undescribed
- bool [read_wavelengthAtStep](#) ([Spectrometer](#) *spectrometer, QVector< double > *WavelengthAtStep)
read_wavelengthAtStep undescribed

Public Member Functions

- **spectrometeroptions** (QObject *parent=0)
- [Spectrometer](#) [return_single_spectrometer](#) (uint16_t id)
return_single_spectrometer Returns the spectrometer configuration with id number 'id'
- uint32_t [returnType](#) (uint16_t id)
returnType Returns the spectrometer type of the spectrometer configuration with id 'id'

Public Attributes

- [Spectrometer](#) * **spectrometerlist**
- uint16_t [number_of_files](#)
the number of spectrometer configuration files found

Private Slots

- void [read_config](#) (uint16_t i, [Spectrometer](#) *spectrometer)
read_config Reads a spectrometer configuration file and sets the values in the appropriate spectrometer configuration struct

Private Member Functions

- void [initVars](#) (int i, [Spectrometer](#) *spectrometer)
initVars initializes all variables to sane values

Private Attributes

- QDir [confdir](#)
the directory in which spectrometer configuration files are searched for
- QStringList [filters](#)
File filters to make shure that only spectrometer configuration files are found.

8.15.1 Detailed Description

The spectrometeroptions class handles spectrometer-configurationfiles.

Bug There are no known bugs.

8.15.2 Member Function Documentation

8.15.2.1 initVars()

```
void spectrometeroptions::initVars (
    int i,
    Spectrometer * spectrometer ) [private]
```

initVars initializes all variables to sane values

Parameters

<i>i</i>	
<i>spectrometer</i>	pointer to a spectrometer configuration struct

Not all possible variables may be set in a configuration file. To prohibit insane behaviour, all variables are initialized to sane values.

8.15.2.2 read_bp_data

```
bool spectrometeroptions::read_bp_data (
    Spectrometer * spectrometer,
    QVector< double > * WavelengthAtBP ) [slot]
```

read_bp_data undescribed

Parameters

<i>spectrometer</i>	
<i>WavelengthAtBP</i>	

Returns

8.15.2.3 read_ce_data

```
bool spectrometeroptions::read_ce_data (
    Spectrometer * spectrometer,
    QVector< double > * EfficiencyAtBP ) [slot]
```

read_ce_data undescribed

Parameters

<i>spectrometer</i>	
<i>EfficiencyAtBP</i>	

Returns

8.15.2.4 read_config

```
void spectrometeroptions::read_config (
    uint16_t i,
    Spectrometer * spectrometer ) [private], [slot]
```

read_config Reads a spectrometer configuration file and sets the values in the appropriate spectrometer configuration struct

Parameters

<i>i</i>	the id of the spectrometer configuration file
<i>spectrometer</i>	the spectrometer configuration struct to write to

8.15.2.5 read_stepsinbp_data

```
bool spectrometeroptions::read_stepsinbp_data (
    Spectrometer * spectrometer,
    QVector< int > * StepsInBP ) [slot]
```

read_stepsinbp_data undescribed

Parameters

<i>spectrometer</i>	
<i>StepsInBP</i>	

Returns

8.15.2.6 read_wavelengthAtStep

```
bool spectrometeroptions::read_wavelengthAtStep (
    Spectrometer * spectrometer,
    QVector< double > * WavelengthAtStep ) [slot]
```

read_wavelengthAtStep undescribed

Parameters

<i>spectrometer</i>	
<i>WavelengthAtStep</i>	

Returns

8.15.2.7 return_single_spectrometer()

```
Spectrometer spectrometeroptions::return_single_spectrometer (
    uint16_t id )
```

return_single_spectrometer Returns the spectrometer configuration with id number 'id'

Parameters

<i>id</i>	the id number
-----------	---------------

Returns

A spectrometer configuration struct

8.15.2.8 returnType()

```
uint32_t spectrometeroptions::returnType (
    uint16_t id )
```

returnType Returns the spectrometertype of the spectrometerconfiguration with id 'id'

Parameters

<i>id</i>	the id number
-----------	---------------

Returns

the spectrometer type

8.15.2.9 write_config

```
void spectrometeroptions::write_config (
    Spectrometer * spectrometer,
    QVector< double > * WavelengthAtBP,
    QVector< double > * EfficiencyAtBP,
    QVector< int > * StepsInBP,
    QVector< double > * WavelengthAtStep ) [slot]
```

write_config write a configuration to a file

Parameters

<i>spectrometer</i>	the spectrometer configuration struct
<i>WavelengthAtBP</i>	
<i>EfficiencyAtBP</i>	
<i>StepsInBP</i>	
<i>WavelengthAtStep</i>	

The documentation for this class was generated from the following files:

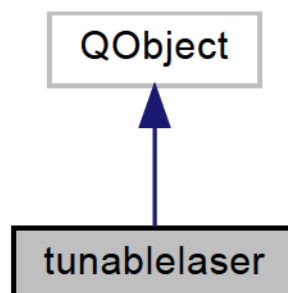
- [spectrometeroptions.h](#)
- [spectrometeroptions.cpp](#)

8.16 tunablelaser Class Reference

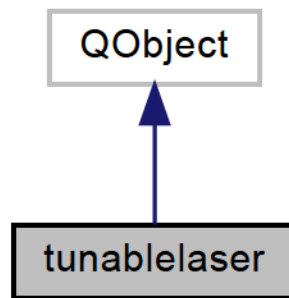
The tunablelaser class is able to communicate with a tunable laser using a file.

```
#include <tunablelaser.h>
```

Inheritance diagram for tunablelaser:



Collaboration diagram for tunablelaser:



Public Member Functions

- **tunablelaser** (QWidget *parentwidget=0, QString configfile="", QSettings *getsettings=0)
- void **setup** ()
setup Checks the provided ctrlfilename for validity and sets up the filewatcher.
- void **setwl** (double wl)
setwl Sets the wavelength of the laser by writing the appropriate command to the ctrlfile.
- void **close** ()
close Turns off the laser by writing the appropriate command to the ctrlfile.

Private Slots

- void **configfile_changed** (QString path)
configfile_changed is called when the filewatcher observes that the ctrlfile was changed.
- void **configdir_changed** (QString path)
configdir_changed is called when the filewatcher observes that the folder containing the ctrlfile was changed.

Private Member Functions

- void **read** ()
read Reads the ctrlfile and passes the first line to `processlasermg()`.
- void **write** (QString command)
write Writes a string to the ctrlfile.
- void **processlasermg** (QString line)
processlasermg searches for known messages/commands in the string provided.
- void **setupfilewatcher** ()
setupfilewatcher Initializes the filewatcher to observe the ctrlfile for changes.

Private Attributes

- QWidget * **parent**
- QString **filename**
- QFile **file**
- QFileInfo **fileinfo**
- QSettings * **settings**
The dinspect settings.
- QString **LaserONmsg**
Message indicating that the laser is turned on.
- QString **LaserDONEmsg**
Message indicating that the laser has set the wavelength.
- QString **LaserOFFmsg**
Message indicating that the laser is turned off.
- QString **LaserOFFcmd**
Command to turn off the laser.
- QString **LaserSETcmd**
Command to set the wavelength of the laser.
- QString **LaserCNTRLfname**
The name of the controlfile.
- bool **foundconfig**
- bool **fileready**
- QFileSystemWatcher * **configwatcher**
The filewatcher observing the controlfile for changes.

8.16.1 Detailed Description

The tunablelaser class is able to communicate with a tunable laser using a file.

Bug There are no known bugs.

8.16.2 Member Function Documentation

8.16.2.1 configdir_changed

```
void tunablelaser::configdir_changed (
    QString path ) [private], [slot]
```

configdir_changed is called when the filewatcher observes that the folder containing the controlfile was changed.

Parameters

<i>path</i>	the observed path
-------------	-------------------

Some texteditors don't modify the file but delete it and save a new one. In such cases, QFileSystemWatcher::File↵

Changed does no longer recognize the file to watch, but it still recognizes the directory-change.

8.16.2.2 configfile_changed

```
void tunablelaser::configfile_changed (
    QString path ) [private], [slot]
```

configfile_changed is called when the filewatcher observes that the controlfile was changed.

Parameters

<i>path</i>	the observed path
-------------	-------------------

8.16.2.3 processlasermsg()

```
void tunablelaser::processlasermsg (
    QString line ) [private]
```

processlasermsg searches for known messages/commands in the string provided.

Parameters

<i>line</i>	input: the string
-------------	-------------------

See also

[read\(\)](#)

8.16.2.4 read()

```
void tunablelaser::read ( ) [private]
```

read Reads the controlfile and passes the first line to [processlasermsg\(\)](#).

See also

[processlasermsg\(\)](#)

8.16.2.5 `setup()`

```
void tunablelaser::setup ( )
```

`setup` Checks the provided `ctrlfilename` for validity and sets up the filewatcher.

See also

[setupfilewatcher\(\)](#)

8.16.2.6 `setwl()`

```
void tunablelaser::setwl (
    double wl )
```

`setwl` Sets the wavelength of the laser by writing the appropriate command to the ctrlfile.

Parameters

<i>wl</i>	The wavelength to set the laser to.
-----------	-------------------------------------

8.16.2.7 `write()`

```
void tunablelaser::write (
    QString command ) [private]
```

`write` Writes a string to the ctrlfile.

Parameters

<i>command</i>	The string which shall be written to the ctrlfile.
----------------	--

The documentation for this class was generated from the following files:

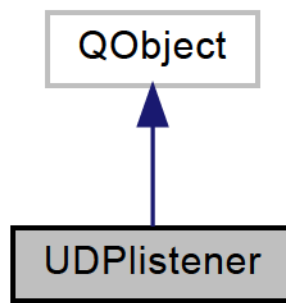
- [tunablelaser.h](#)
- [tunablelaser.cpp](#)

8.17 UDPlistener Class Reference

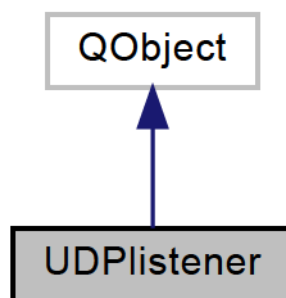
The [UDPlistener](#) class is used to control dinspect with UDP packages.

```
#include <udplistener.h>
```

Inheritance diagram for UDPlistener:



Collaboration diagram for UDPlistener:



Public Slots

- void [bind](#) ()
bind Binds to a new UDP port

Signals

- void [startMeasurement](#) ()
startMeasurement is emitted when the command to start a measurement was received
- void [setDetector](#) (QString det)
setDetector is emitted when the command to change/set the detector was received
- void [setScanrange](#) (double low, double high)
setScanrange is emitted when the command to change the wavelength mesaurement range was received
- void [stopMeasurement](#) ()
stopMeasurement is emitted when the command to stop a running measurement was received
- void [setTTMChannel](#) (int channel)
setTTMChannel is emitted when the command to change the TTM channel was received
- void [setTTMCoincidenceChannel](#) (int channel)
setTTMCoincidenceChannel is emitted when the command to change the TTM coincidence channel was received

Public Member Functions

- [UDPlistener](#) (QSettings *settings, QObject *parent=0)
UDPlistener listen for commands on a UDP port and execute them.

Private Slots

- void [processPendingDatagrams](#) ()
processPendingDatagrams reads data from the UDP socket
- void [processCommands](#) (QString msg)
processCommands extracts commands out of received data and executes them

Private Attributes

- QSettings * [settings](#)
Dinspect configuration.
- QUdpSocket [socket](#)
the UDP socket
- uint [port](#)
port the listener listens to
- bool [alreadybound](#)
true if the listener is already listening to a port

8.17.1 Detailed Description

The [UDPlistener](#) class is used to control dinspect with UDP packages.

Bug There are no known bugs

8.17.2 Constructor & Destructor Documentation

8.17.2.1 UDPlistener()

```
UDPlistener::UDPlistener (
    QSettings * settings,
    QObject * parent = 0 ) [explicit]
```

[UDPlistener](#) listen for commands on a UDP port and execute them.

Parameters

settings	a pointer to the dinspect settings, used to get the port to bind to and known commands
--------------------------	--

[UDPlistener](#) opens a UDP socket and binds to a port specified in the dinspect settings. Incoming packages are analysed to check if they contain known commands. If they do, these commands are executed.

The commands can be changes in the dinspect settings dialog, but the standard ones are:

- StartMeas

- StopMeas
- SetMeasRange(double, double)
- SetDetectorTTM
- SetDetectorPM100D
- SetTTMStop(int)
- SetTTMCoinStop(int)

Inside the parentheses, the expected datatypes are given.

8.17.3 Member Function Documentation

8.17.3.1 bind

```
void UDPlistener::bind ( ) [slot]
```

bind Binds to a new UDP port

This function checks whether the UDP socket is already bound to a specific port. If so, it closes this connection and binds to the new port. If not, it binds to the port right away.

8.17.3.2 processCommands

```
void UDPlistener::processCommands (
    QString msg ) [private], [slot]
```

processCommands extracts commands out of received data and executes them

Parameters

<i>msg</i>	input: the received message
------------	-----------------------------

8.17.3.3 setDetector

```
void UDPlistener::setDetector (
    QString det ) [signal]
```

setDetector is emitted when the command to change/set the detector was received

Parameters

<i>det</i>	output: String indicating which detector to use
------------	---

8.17.3.4 setScanrange

```
void UDPlistener::setScanrange (
    double low,
    double high ) [signal]
```

setScanrange is emitted when the command to change the wavelength measurement range was received

Parameters

<i>low</i>	output: the lower wavelength measurement range bound
<i>high</i>	output: the upper wavelength measurement range bound

8.17.3.5 setTTMChannel

```
void UDPlistener::setTTMChannel (
    int channel ) [signal]
```

setTTMChannel is emitted when the command to change the TTM channel was received

Parameters

<i>channel</i>	output: Chosen TTM channel
----------------	----------------------------

8.17.3.6 setTTMCoincidenceChannel

```
void UDPlistener::setTTMCoincidenceChannel (
    int channel ) [signal]
```

setTTMCoincidenceChannel is emitted when the command to change the TTM coincidence channel was received

Parameters

<i>channel</i>	output: Chosen TTM coincidence channel
----------------	--

The documentation for this class was generated from the following files:

- [udplistener.h](#)
- [udplistener.cpp](#)

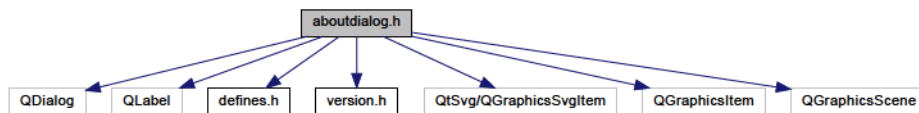
Chapter 9

File Documentation

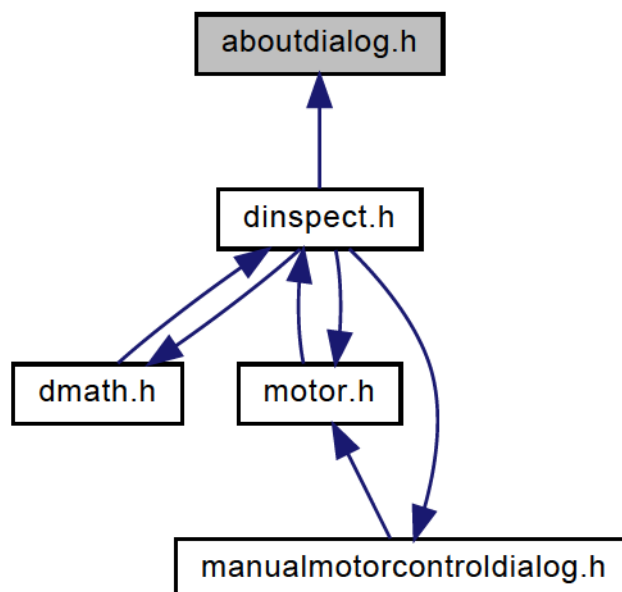
9.1 aboutdialog.h File Reference

```
#include <QDialog>
#include <QLabel>
#include "defines.h"
#include "version.h"
#include <QtSvg/QGraphicsSvgItem>
#include <QGraphicsItem>
#include <QGraphicsScene>
```

Include dependency graph for aboutdialog.h:



This graph shows which files directly or indirectly include this file:



Classes

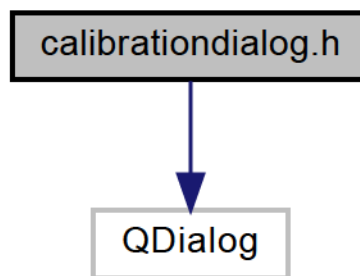
- class [AboutDialog](#)

The [AboutDialog](#) class is just used to show the about-dialog.

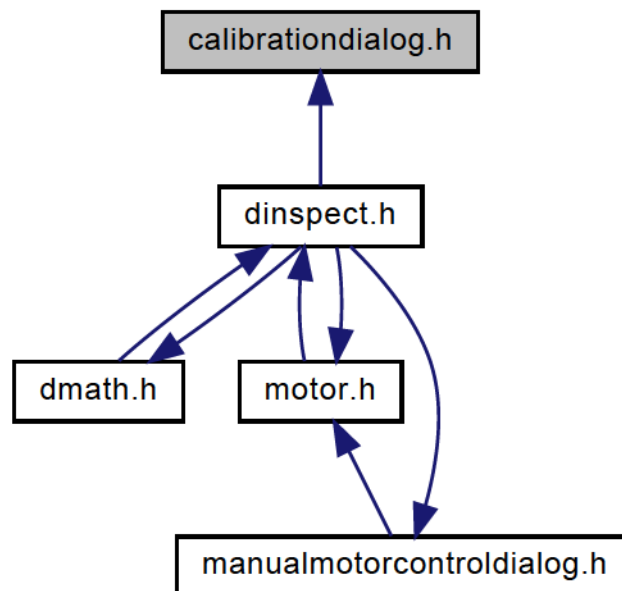
9.2 calibrationdialog.h File Reference

```
#include <QDialog>
```

Include dependency graph for calibrationdialog.h:



This graph shows which files directly or indirectly include this file:



Classes

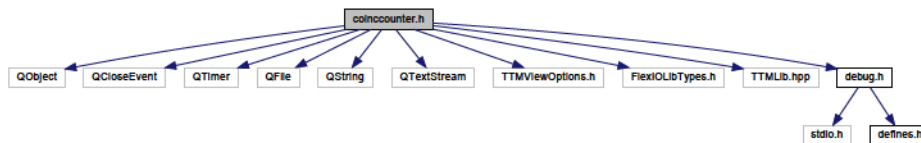
- class [CalibrationDialog](#)

The [CalibrationDialog](#) class handles userinput to configure the calibration-procedure.

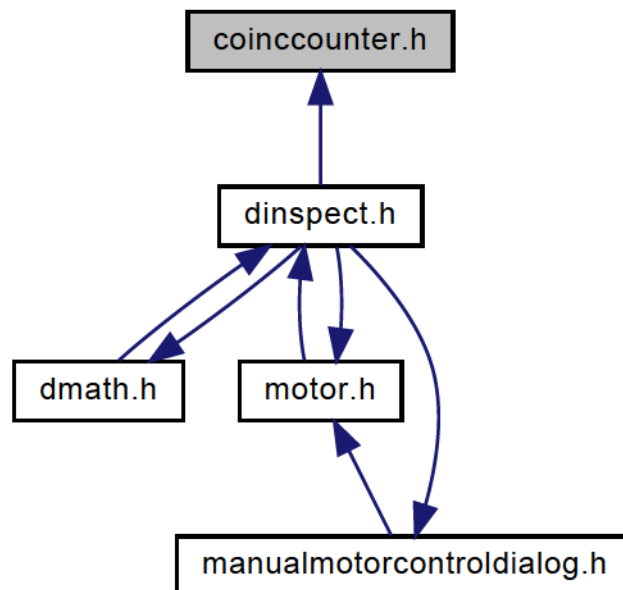
9.3 coinccounter.h File Reference

```
#include <QObject>
#include <QCloseEvent>
#include <QTimer>
#include <QFile>
#include <QString>
#include <QTextStream>
#include "TTMViewOptions.h"
#include "FlexIOLibTypes.h"
#include "TTMLib.hpp"
#include "debug.h"
```

Include dependency graph for coinccounter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [CoincCounter](#)

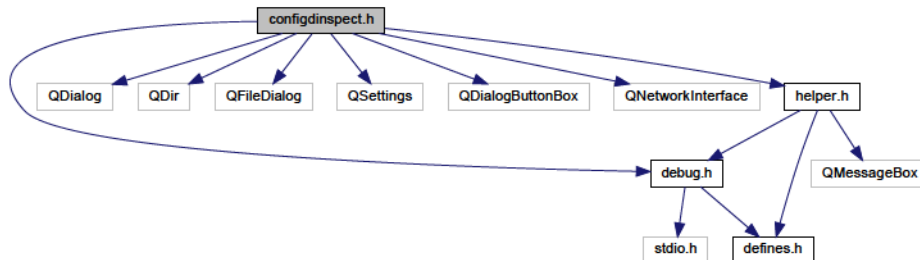
The [CoincCounter](#) class acquires coincidence frequencies from a TM8000.

9.4 configdinspect.h File Reference

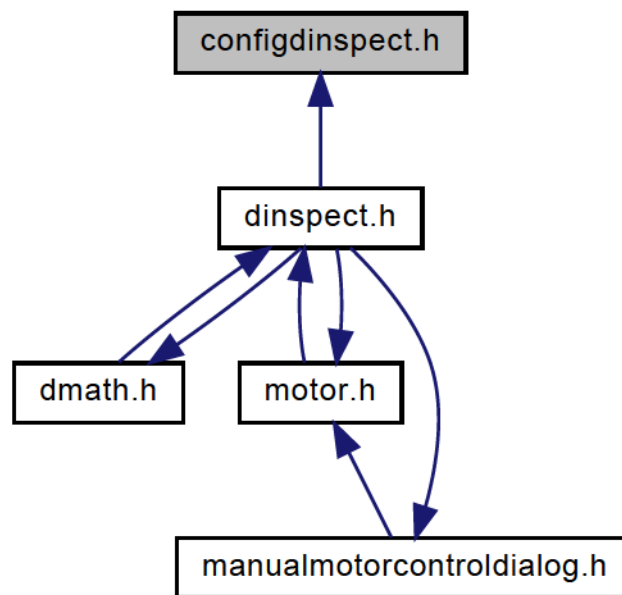
```
#include "debug.h"
#include <QDialog>
```

```
#include <QDir>
#include <QFileDialog>
#include <QSettings>
#include <QDialogButtonBox>
#include <QNetworkInterface>
#include "helper.h"
```

Include dependency graph for configdinspect.h:



This graph shows which files directly or indirectly include this file:



Classes

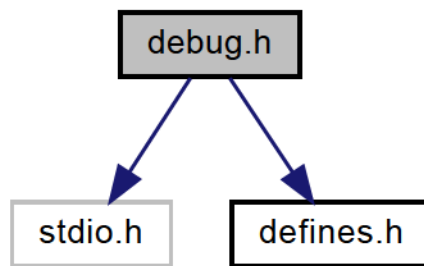
- class [ConfigDinspect](#)

The [ConfigDinspect](#) class handles user input to configure the behaviour of `dinspect`.

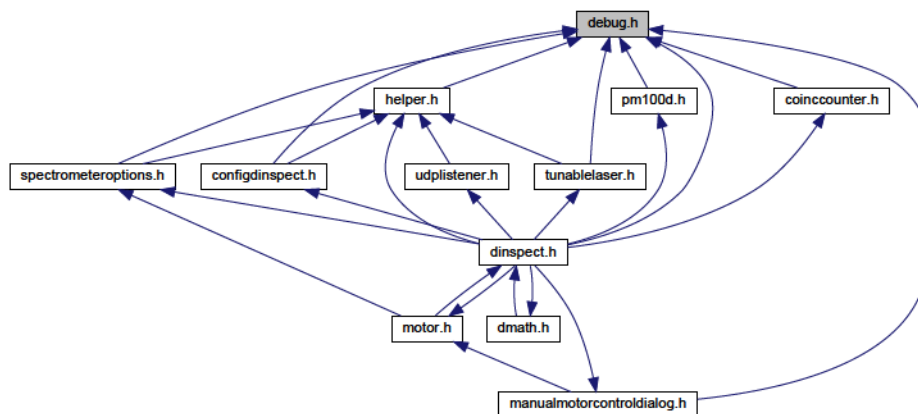
9.5 debug.h File Reference

contains debug macros

```
#include "stdio.h"
#include "defines.h"
Include dependency graph for debug.h:
```



This graph shows which files directly or indirectly include this file:



9.5.1 Detailed Description

contains debug macros

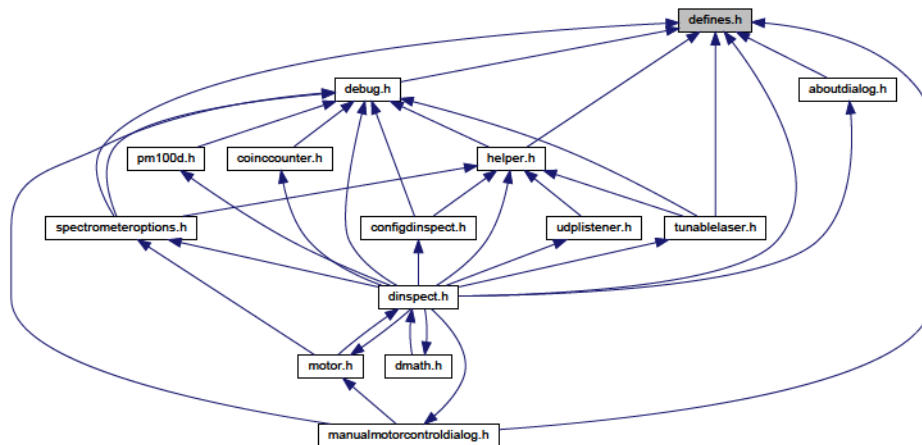
Bug Printing to console does not work on Windows. Workaround: Redirect `stderr` to `stdout` and redirect `stdout` to a file.

This file defines macros to style and simplify output to console.

9.6 defines.h File Reference

Various compile-time definitions.

This graph shows which files directly or indirectly include this file:



Macros

- `#define DEBUGSPECTROMETERCONFIG FALSE`
- `#define CURRENT_SAVE_STRUCTURE_VERSION 21`
- `#define CALIBRATION_RUNS_NEEDED 5`
- `#define CALIBRATION_RUNS_NEEDED_ED 3`
- `#define CALIBRATE_APPROXIMATELY_WITH_ARTIFICIAL_DATA FALSE`
- `#define CALIBRATE_WITH_EDGE_DETECTION TRUE`
- `#define DEBUG true`
- `#define DEBUGERROR true`
- `#define DEBUGWARNING true`
- `#define DEBUGINFO true`
- `#define EPS 0.0000001`
- `#define PI 3.14159265358979323846`
- `#define DEGTORAD PI/180`
- `#define RADTODEG 180/PI`
- `#define ALLOW_UNCALIBRATED_MEASUREMENT FALSE`
- `#define CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_TYPE "rd243"`
- `#define CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_SUBVERSION "rd245"`

9.6.1 Detailed Description

Various compile-time definitions.

Bug There are no known bugs.

Contains definitions of various kind - mathematical, version constants, debug-variables, ...

9.6.2 Macro Definition Documentation

9.6.2.1 ALLOW_UNCALIBRATED_MEASUREMENT

```
#define ALLOW_UNCALIBRATED_MEASUREMENT FALSE
```

Allow measurement-runs to be carried out although the spectrometer is uncalibrated.
default: FALSE

9.6.2.2 CALIBRATE_APPROXIMATELY_WITH_ARTIFICIAL_DATA

```
#define CALIBRATE_APPROXIMATELY_WITH_ARTIFICIAL_DATA FALSE
```

This switch is useful during development/debugging of the calibration routines. The maxima and their positions can be entered in the code instead of being calculated out of saved datasets.
default: FALSE

9.6.2.3 CALIBRATE_WITH_EDGE_DETECTION

```
#define CALIBRATE_WITH_EDGE_DETECTION TRUE
```

Specifies if the sensordata of the PCB-motor is used to calculate the scanmove-stepsizes at runtime.
default: TRUE

9.6.2.4 CALIBRATION_RUNS_NEEDED

```
#define CALIBRATION_RUNS_NEEDED 5
```

Defines the number of datasets needed to calibrate the spectrometer. Used for old routines which tried to calculate the stepsize in prior.
default: 5

9.6.2.5 CALIBRATION_RUNS_NEEDED_ED

```
#define CALIBRATION_RUNS_NEEDED_ED 3
```

Defines the number of datasets needed to calibrate the spectrometer when Edge-detection is used.
default: 2

9.6.2.6 CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_SUBVERSION

```
#define CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_SUBVERSION "rd245"
```

Command to access memory position on PCB-Motor controller in which minor version of the [Spectrometer](#) is stored.

9.6.2.7 CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_TYPE

```
#define CHAR_READ_MOTOR_MEMORY_POSITION_SPECTROMETER_TYPE "rd243"
```

Command to access memory position on PCB-Motor controller in which major version of the [Spectrometer](#) is stored.

9.6.2.8 CURRENT_SAVE_STRUCTURE_VERSION

```
#define CURRENT_SAVE_STRUCTURE_VERSION 21
```

defines the current version of the savefile-structure. Used when reading/writing savefiles to assure the right data is read/written from/into the file. The following versions are implemented:

```
1 :Step && FrequencyAtStep
11 :Step && FrequencyAtStep && Sensordata
2 :BP && WavelengthAtBP && FrequencyAtBP
21 :BP && WavelengthAtBP && FrequencyAtBP_EC
210:StepsInBP
211:WavelengthAtStep_EC
212:WavelengthAtBP && FrequencyAtBP_EC && StepsInBP_EC
```

9.6.2.9 DEBUG

```
#define DEBUG true
```

Enables the execution of various debug-paths used during development.
default: FALSE.

9.6.2.10 DEBUGERROR

```
#define DEBUGERROR true
```

If set to TRUE, enables the execution of the `DEBUG_ERROR()` makro which is used to write error messages (critical) to stdout.
default: true

9.6.2.11 DEBUGINFO

```
#define DEBUGINFO true
```

If set to TRUE, enables the execution of the `DEBUG_INFO()` makro which is used to write usefull information to stdout.
default: true

9.6.2.12 DEBUGWARNING

```
#define DEBUGWARNING true
```

If set to TRUE, enables the execution of the `DEBUG_WARNING()` makro which is used to write warnings about unexpected behaviour to stdout.
default: true

9.6.2.13 DEGTORAD

```
#define DEGTORAD PI/180
```

Conversion factor from degree to radians. $\text{PI}/180$

9.6.2.14 EPS

```
#define EPS 0.0000001
```

'epsilon' used to check floatingpoint variables in if-conditions.
default: 0.0000001

9.6.2.15 PI

```
#define PI 3.14159265358979323846
```

Pi.
default: 3.14159265358979323846

9.6.2.16 RADTODEG

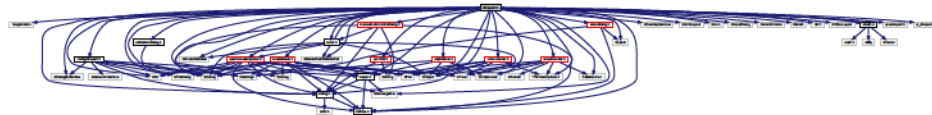
```
#define RADTODEG 180/PI
```

Conversion factor from radians to degree. 180/PI

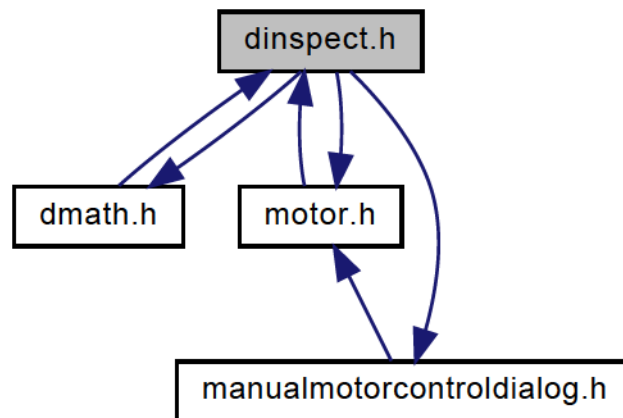
9.7 dinspect.h File Reference

```
#include <QApplication>
#include <QDebug>
#include <QDesktopServices>
#include <QDialog>
#include <QDialogButtonBox>
#include <QDir>
#include <QFileDialog>
#include <QGridLayout>
#include <QIcon>
#include <QInputDialog>
#include <QLabel>
#include <QMainWindow>
#include <QMessageBox>
#include <QObject>
#include <QSettings>
#include <QString>
#include <QtCore/QtGlobal>
#include <QtMath>
#include <QtSerialPort/QtSerialPort>
#include <QTimer>
#include <QUrl>
#include <QVBoxLayout>
#include "calibrationdialog.h"
#include "configdinspect.h"
#include "DataSource.h"
#include "debug.h"
#include "defines.h"
#include "dmath.h"
#include "helper.h"
#include "motor.h"
```

```
#include "qcustomplot.h"
#include "snspdcounter.h"
#include "coinccounter.h"
#include "spectrometeroptions.h"
#include "TTMViewOptions.h"
#include "udplistener.h"
#include "ui_dinspect.h"
#include "tunablelaser.h"
#include "pm100d.h"
#include "aboutdialog.h"
#include "manualmotorcontroldialog.h"
Include dependency graph for dinspect.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Dinspect](#)

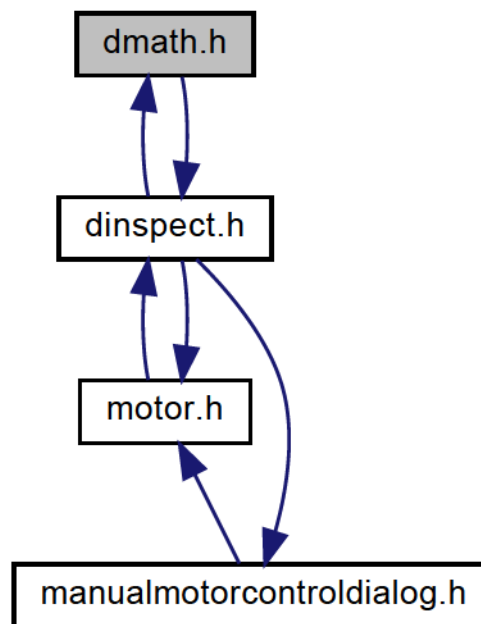
The [Dinspect](#) class is the main class of this application.

9.8 dmath.h File Reference

```
#include "dinspect.h"
#include <math.h>
#include <utility>
#include <QVector>
Include dependency graph for dmath.h:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [dmath](#)

Functions

- `int dmath::find\_max (QVector< double > vec)`
find_max determines the biggest entry in a vector and returns its position
- `std::pair< double, double > dmath::find\_max (QVector< double > frequ, QVector< double > wl)`
find_max determines the biggest entry in a vector and returns the corresponding entry of both this and another vector
- `double dmath::normal\_distribution (double A, double sigma, double mu, double x)`
normal_distribution evaluates the normal distribution with given parameters at a given point
- `double dmath::fwhm\_gauss (double s)`
fwhm_gauss calculates the FWHM of a gaussian out of its standard deviation
- `double dmath::calc\_stepsize (QVector< double > wl, QVector< int > mpos)`
calc_stepsize calculates the scanmove-stepsize of the spectrometer when edge-detection is not used
- `double dmath::calc\_stepsize (double l1, double l2, double l3, double l4, double ds12, double ds34)`
calc_stepsize calculates the scanmove-stepsize of the spectrometer when edge-detection is not used
- `double dmath::calc\_unneeded\_stepsize (Spectrometer *spectrometer, QVector< bool > SensorDataAtStep)`
calc_unneeded_stepsize calculates the mean scanmove-stepsize of the spectrometer
- `double dmath::calc\_deviationangle (double G, double l1, double l2, double da)`
calc_deviationangle calculates the deviation angle of the spectrometer
- `double dmath::calc\_deviationangle\_without\_stepsize (double gratingconstant, QVector< double > wl, QVector< double > mangle)`
calc_deviationangle_without_stepsize calculates the deviation angle of the spectrometer (used in calibration). Does not rely on a fixed stepsize.
- `double dmath::calc\_bandpass (double G, double fin, double win, double alpha)`
calc_bandpass the size of a bandpass

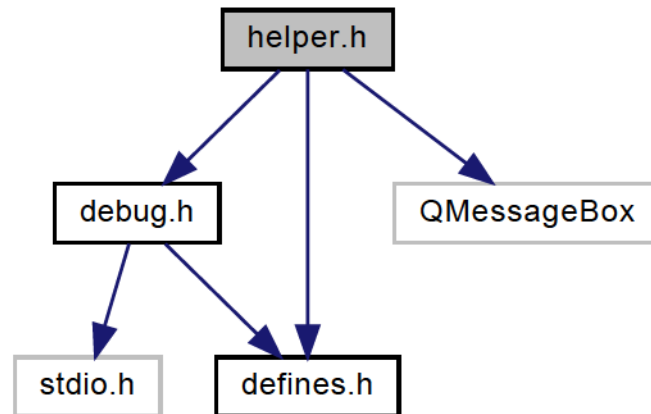
- `std::pair< uint32_t, uint32_t > dmath::calc_range (Spectrometer *spectrometer, double sec_angle, double sec_percentage)`
calc_range calculates the wavelength range accessible by the spectrometer
- `double dmath::calc_angle_offset (Spectrometer *spectrometer, double l, int mpos)`
calc_angle_offset calculates the offset-angle (used in calibration).
- `double dmath::calc_angle_offset_without_stepsize (Spectrometer *spectrometer, QVector< double > wl, QVector< double > mangle)`
calc_angle_offset_without_stepsize calculates the offset-angle (used in calibration). Does not rely on a fixed stepsize.
- `double dmath::relate_maxpos_to_angle_difference (Spectrometer *spectrometer, QVector< bool > SensorDataAtStep, int mpos, bool calibrated_with_edge_detection)`
relate_maxpos_to_angle_difference finds the angle of rotation at which the maximum was measured
- `double dmath::calc_alpha (double G, double l, double dv)`
calc_alpha calculates the angle of light incident on the grating at a given wavelength
- `double dmath::calc_beta (double G, double l, double dv)`
calc_beta calculates the angle of light reflected from the grating at a given wavelength
- `double dmath::calc_lambda_from_alpha (double G, double a, double dv)`
calc_lambda_from_alpha calculates the wavelength selected by a spectrometer at a rotation angle a
- `void dmath::calc_coupling_efficiency (Spectrometer *spectrometer, QVector< double > *EfficiencyAtBP, QVector< double > *WavelengthAtBP)`
calc_coupling_efficiency calculates the maximal coupling efficiency at each bandpass
- `void dmath::calc_steps_in_BP (QVector< double > *WavelengthAtStep, QVector< double > *WavelengthAtBP, QVector< int > *StepsInBP)`
calc_steps_in_BP calculates the number of steps in each bandpass
- `std::pair< uint32_t, uint32_t > dmath::calc_bandpasses_in_scanrange (std::pair< double, double > scanrange, QVector< double > *WavelengthAtBP)`
calc_bandpasses_in_scanrange calculates the number of bandpasses covered by the scanrange of the current measurement
- `uint32_t dmath::calc_number_of_steps_in_scanrange (std::pair< uint32_t, uint32_t > bpos, QVector< int > *StepsInBP)`
calc_number_of_steps_in_scanrange calculates the estimated number of steps between two bandpasses
- `double dmath::recalc_wavelength_at_step (Spectrometer *spectrometer, double newstepsize, uint64_t minstep, uint64_t maxstep, QVector< double > *WavelengthAtStep_EC, double oldangle)`
recalc_wavelength_at_step recalculates the wavelength at each measurement step already carried out, knowing the number of steps between two edges (which fix the angle of rotation)
- `uint16_t dmath::calc_steps_between_edges (QVector< uint64_t > *EdgeAtStep)`
calc_steps_between_edges calculates how many steps lie between
- `std::pair< int, uint64_t > dmath::recalc_steps_in_BP (QVector< double > *WavelengthAtStep, QVector< double > *WavelengthAtBP, QVector< int > *StepsInBP, uint64_t minpos, uint64_t maxpos, std::pair< int, uint64_t > oldstate)`
recalc_steps_in_BP calculates the number of datapoints found in each bandpass
- `uint16_t dmath::calc_how_many_edges_to_skip (Spectrometer *spectrometer, double wl)`
calc_how_many_edges_to_skip calculates how many edges have to be skipped to position as close as possible to a certain wavelength

9.9 helper.h File Reference

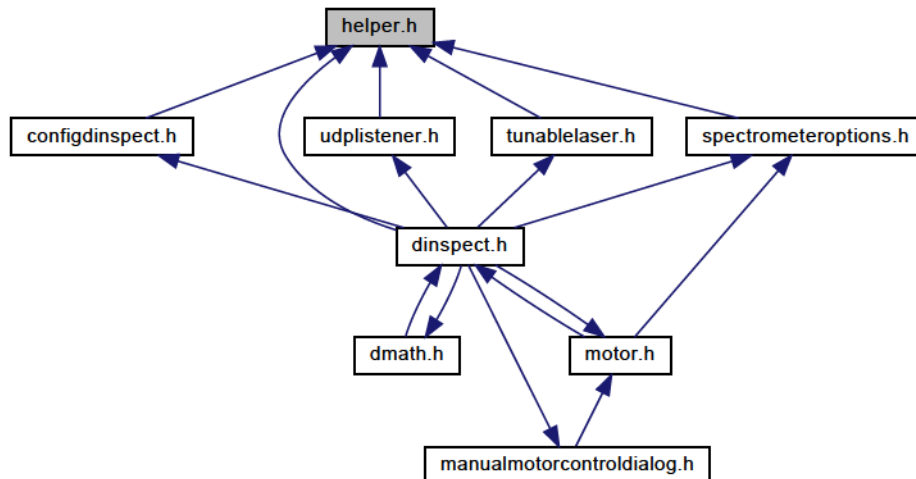
```
#include "debug.h"
#include "defines.h"
```

```
#include <QMessageBox>
```

Include dependency graph for helper.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- [helper](#)
contains small functions to display messages

Functions

- void [helper::message](#) (QString msg)
message displays a message box
- void [helper::error](#) (QString msg)
error displays an error-messagebox and writes a debug_error message to stdout
- void [helper::warning](#) (QString msg)
warning displays warning-messagebox and writes a debug_warning message to stdout
- void [helper::info](#) (QString msg)
info displays an info-messagebox and writes a debug_info message to stdout

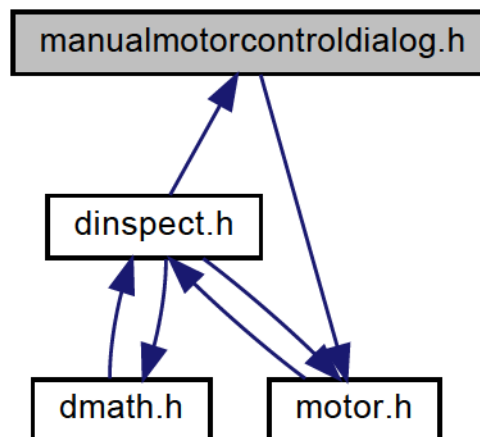
9.10 manualmotorcontroldialog.h File Reference

```
#include <QDialog>
#include <QPushButton>
#include <QSpinBox>
#include <QDoubleSpinBox>
#include <QLineEdit>
#include <QString>
#include <QCheckBox>
#include "debug.h"
#include "defines.h"
#include "motor.h"
```

Include dependency graph for manualmotorcontroldialog.h:



This graph shows which files directly or indirectly include this file:



Classes

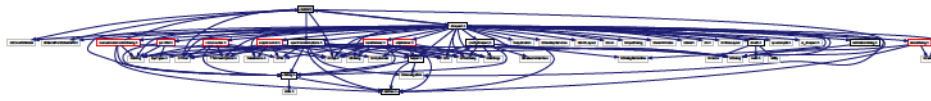
- class [ManualMotorControlDialog](#)

The [ManualMotorControlDialog](#) class offers the ability to manually operate the PCBMotor.

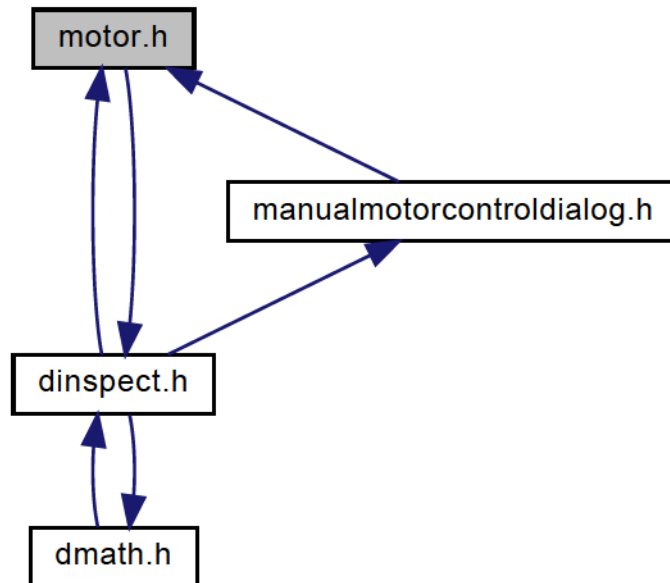
9.11 motor.h File Reference

```
#include <QDialog>
#include <QtCore/QtGlobal>
#include <QtSerialPort/QSerialPort>
#include <QObject>
#include <QTimer>
#include "dinspect.h"
```

```
#include "spectrometeroptions.h"
Include dependency graph for motor.h:
```



This graph shows which files directly or indirectly include this file:



Classes

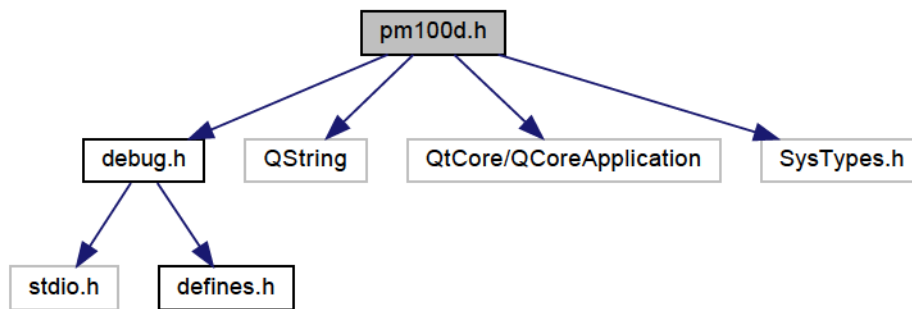
- class [MotorSettingsDialog](#)
The [MotorSettingsDialog](#) class handles user-input to configure the connection to the PCB-motor.
- struct [MotorSettingsDialog::MotorSettings](#)
The [MotorSettings](#) struct contains contains the information to successfully establish a serial connection to the PC↔BMotor.
- class [Motor](#)
The [Motor](#) class operates the PCB-motor.

9.12 pm100d.h File Reference

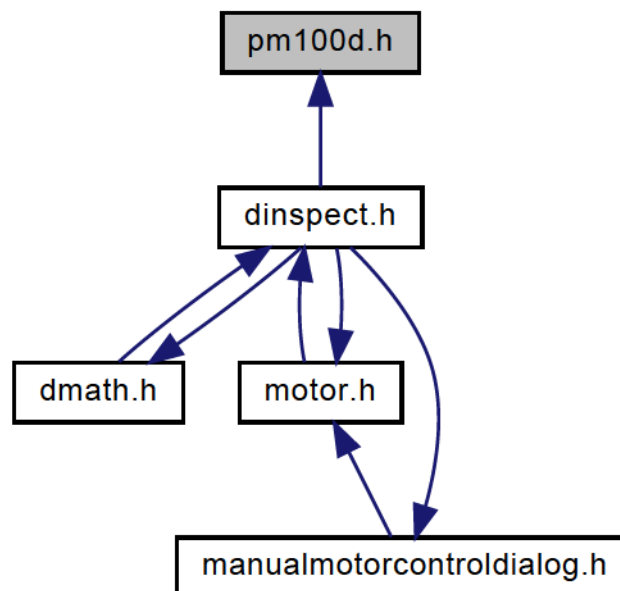
```
#include "debug.h"
#include <QString>
#include <QtCore/QCoreApplication>
```

```
#include "SysTypes.h"
```

Include dependency graph for pm100d.h:



This graph shows which files directly or indirectly include this file:



Classes

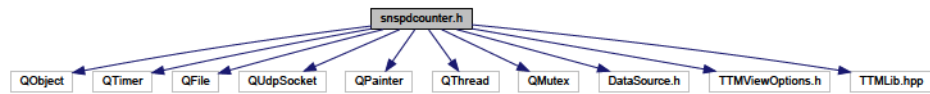
- class [PM100D](#)

The [PM100D](#) class operates the Thorlabs [PM100D](#) powermeter.

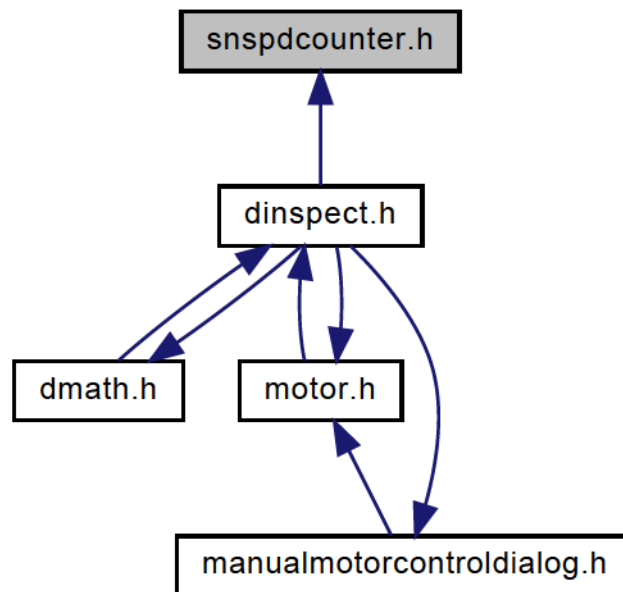
9.13 snspdcounter.h File Reference

```
#include <QObject>
#include <QTimer>
#include <QFile>
#include <QUdpSocket>
#include <QPainter>
#include <QThread>
#include <QMutex>
#include "DataSource.h"
```

```
#include "TTMViewOptions.h"
#include "TTMLib.hpp"
Include dependency graph for snspdcounter.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [SNSPDcounter](#)

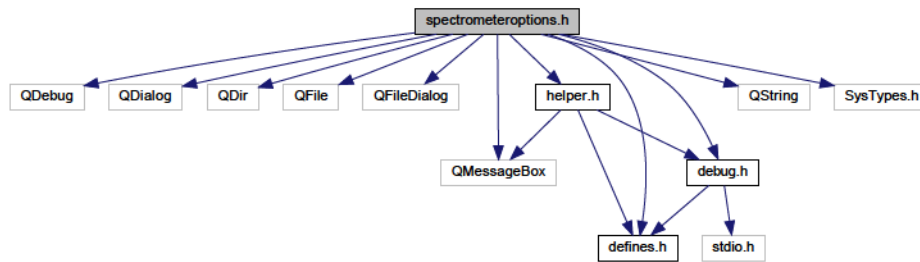
The [SNSPDcounter](#) class acquires single event frequencies from a TM8000.

9.14 spectrometeroptions.h File Reference

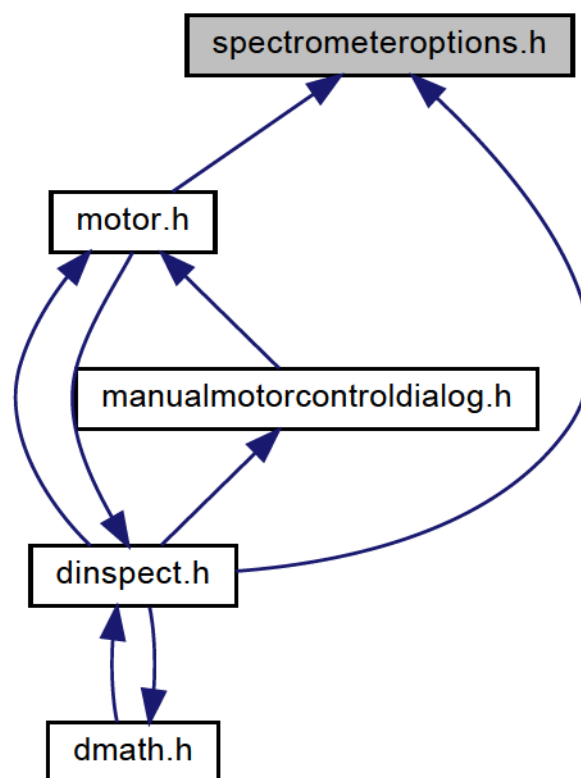
```
#include <QDebug>
#include <QDialog>
#include <QDir>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QString>
#include "debug.h"
#include "defines.h"
#include "helper.h"
```

```
#include "SysTypes.h"
```

Include dependency graph for spectrometeroptions.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [Spectrometer](#)
The [Spectrometer](#) struct holds spectrometer-specific variables.
- class [spectrometeroptions](#)
The [spectrometeroptions](#) class handles spectrometer-configurationfiles.

9.15 tunablelaser.h File Reference

```
#include "debug.h"
#include "defines.h"
```

```

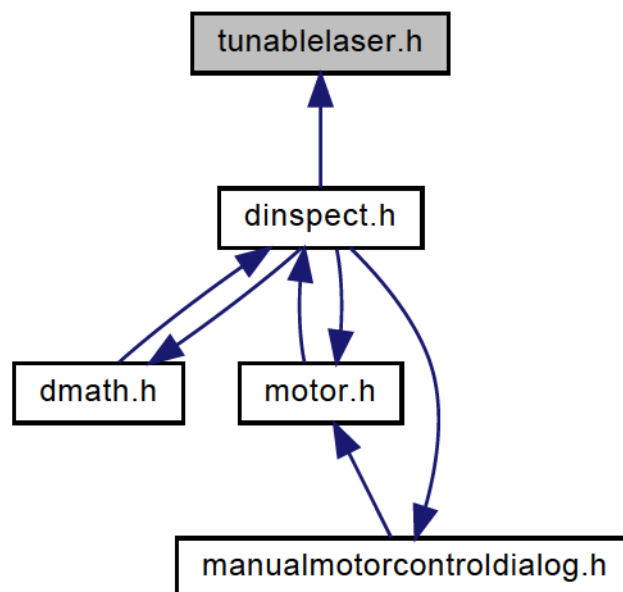
#include "helper.h"
#include <QFile>
#include <QDialog>
#include <QDir>
#include <QFileDialog>
#include <QString>
#include <QTimer>
#include <QTextStream>
#include <QWidget>
#include <QFileSystemWatcher>
#include <QFileInfo>
#include <QObject>
#include <QDebug>
#include <QSettings>

```

Include dependency graph for tunablelaser.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [tunablelaser](#)

The tunablelaser class is able to communicate with a tunable laser using a file.

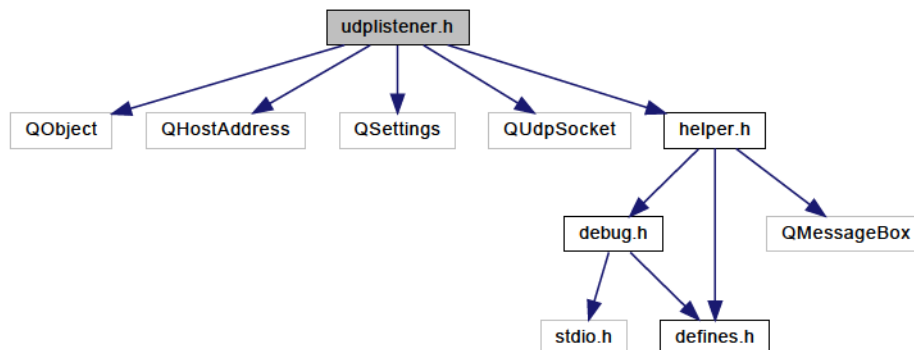
9.16 udplistener.h File Reference

```

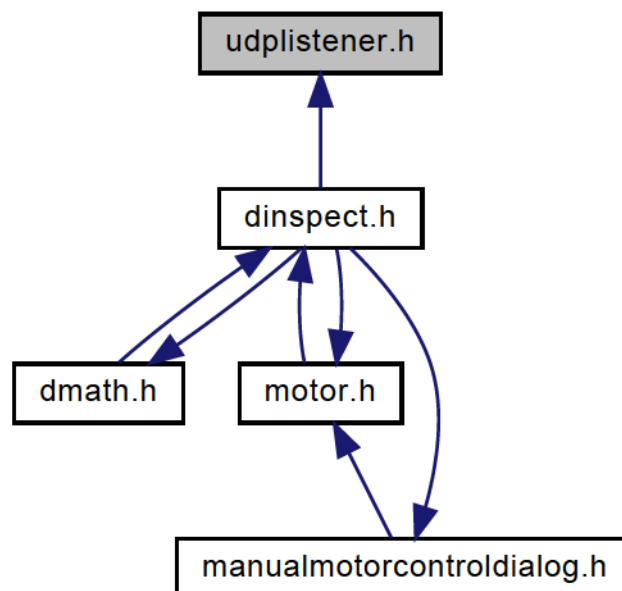
#include <QObject>
#include <QHostAddress>

```

```
#include <QSettings>
#include <QUdpSocket>
#include "helper.h"
Include dependency graph for udplistener.h:
```



This graph shows which files directly or indirectly include this file:



Classes

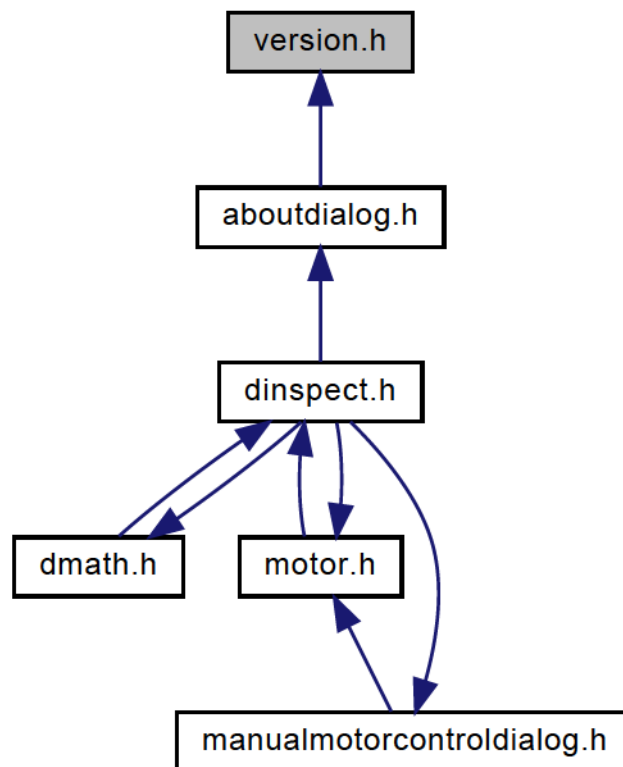
- class `UDPListener`

The `UDPListener` class is used to control `dinspect` with UDP packages.

9.17 version.h File Reference

This file contains information about the code version.

This graph shows which files directly or indirectly include this file:



Macros

- `#define VERSION_GIT "v0.17-4-g58377d0"`
The git commit description.
- `#define VERSION_GIT_DATE 201701031610`
The date of the git commit.
- `#define VERSION_BUILD_DATE 201701031653`
The builddate.

9.17.1 Detailed Description

This file contains information about the code version.

Bug There are no known bugs.

The definitions in this file are used to fill the about-dialog with information about the code: The git commit description, the commit date and the builddate.

Index

ALLOW_UNCALIBRATED_MEASUREMENT
 defines.h, [122](#)
AboutDialog, [41](#)
aboutdialog.h, [117](#)
ActionAfterEdgeDetection
 Dinspect, [60](#)
apply
 MotorSettingsDialog, [95](#)
askmotor
 MotorSettingsDialog, [95](#)

bind
 UDPlistener, [114](#)

CALIBRATE_APPROXIMATELY_WITH_ARTIFICIAL↔
 _DATA
 defines.h, [123](#)
CALIBRATE_WITH_EDGE_DETECTION
 defines.h, [123](#)
CALIBRATION_RUNS_NEEDED_ED
 defines.h, [123](#)
CALIBRATION_RUNS_NEEDED
 defines.h, [123](#)
CHAR_READ_MOTOR_MEMORY_POSITION_SPE↔
 CTROMETER_SUBVERSION
 defines.h, [123](#)
CHAR_READ_MOTOR_MEMORY_POSITION_SPE↔
 CTROMETER_TYPE
 defines.h, [123](#)
CURRENT_SAVE_STRUCTURE_VERSION
 defines.h, [123](#)
calc_alpha
 dmath, [26](#)
calc_angle_offset
 dmath, [27](#)
calc_angle_offset_without_stepsize
 dmath, [27](#)
calc_bandpass
 dmath, [28](#)
calc_bandpasses_in_scanrange
 dmath, [28](#)
calc_beta
 dmath, [28](#)
calc_coupling_efficiency
 dmath, [29](#)
calc_deviationangle
 dmath, [29](#)
calc_deviationangle_without_stepsize
 dmath, [30](#)
calc_how_many_edges_to_skip
 dmath, [31](#)
calc_lambda_from_alpha
 dmath, [31](#)
calc_number_of_steps_in_scanrange
 dmath, [31](#)
calc_range
 dmath, [32](#)
calc_steps_between_edges
 dmath, [32](#)
calc_steps_in_BP
 dmath, [33](#)
calc_stepsize
 dmath, [33](#)
calc_unneeded_stepsize
 dmath, [34](#)
CalibrationDialog, [42](#)
 CalibrationDialogDone, [44](#)
 dialogfinished, [44](#)
 get_onlysinglerun, [45](#)
 get_security_angle, [45](#)
 get_security_percentage, [45](#)
 get_steps_to_go, [45](#)
 get_steps_wo_measurement, [46](#)
 ret_singlerun, [46](#)
 ret_understood, [46](#)
 ret_wantnewscan, [46](#)
CalibrationDialogDone
 CalibrationDialog, [44](#)
CalibrationDialogFinished
 Dinspect, [60](#)
calibrationdialog.h, [118](#)
CoincCounter, [47](#)
coinccounter.h, [119](#)
command_microstep
 Motor, [80](#)
command_nextedge
 Motor, [80](#)
command_roughly_to_wl
 Motor, [81](#)
command_scanmove
 Motor, [81](#)
command_singlestep
 Motor, [81](#)
command_step
 Motor, [81](#)
command_to_start_pos
 Motor, [83](#)
ConfigDinspect, [48](#)
 LoadConfig, [49](#)

- SaveConfig, 49
- configinspect.h, 119
- configdir_changed
 - tunablelaser, 109
- configfile_changed
 - tunablelaser, 110
- Connect_to_motor
 - ManualMotorControlDialog, 74
- CreatePositionLabel
 - Motor, 83
- DEBUGERROR
 - defines.h, 124
- DEBUGINFO
 - defines.h, 124
- DEBUGWARNING
 - defines.h, 124
- DEBUG
 - defines.h, 124
- DEGTORAD
 - defines.h, 124
- debug.h, 120
- defines.h, 121
 - ALLOW_UNCALIBRATED_MEASUREMENT, 122
 - CALIBRATE_APPROXIMATELY_WITH_ARTIFICIAL_DATA, 123
 - CALIBRATE_WITH_EDGE_DETECTION, 123
 - CALIBRATION_RUNS_NEEDED_ED, 123
 - CALIBRATION_RUNS_NEEDED, 123
 - CHAR_READ_MOTOR_MEMORY_POSITION ↔ SPECTROMETER_SUBVERSION, 123
 - CHAR_READ_MOTOR_MEMORY_POSITION ↔ SPECTROMETER_TYPE, 123
 - CURRENT_SAVE_STRUCTURE_VERSION, 123
 - DEBUGERROR, 124
 - DEBUGINFO, 124
 - DEBUGWARNING, 124
 - DEBUG, 124
 - DEGTORAD, 124
 - EPS, 124
 - PI, 125
 - RADTODEG, 125
- dialogfinished
 - CalibrationDialog, 44
- Dinspect, 50
 - ActionAfterEdgeDetection, 60
 - CalibrationDialogFinished, 60
 - end_scan, 61
 - get_detector_data, 61
 - GetFrequencyFromSNSPDcounter, 61
 - GetPowerFromPM100D, 61
 - getSpectrometerSubversion, 62
 - getSpectrometerType, 62
 - GraphSelectionChanged, 62
 - initActionsConnections, 62
 - motor_is_at_starting_position, 62
 - motorfoundfallingedge, 63
 - motorfoundinitedge, 63
 - motorfoundrisingedge, 63
 - on_DetectorComboBox_activated, 64
 - plot_savefile, 64
 - recalc_with_dynamic_stepsize, 64
 - SNSPD_update_channel, 68
 - SNSPD_update_coinc_channel, 68
 - save, 64
 - saveAutomatically, 65
 - SaveGraph, 65
 - ScanRangeMinPositioning, 65
 - setPM100D_AverageCount, 65
 - setupGraph, 66
 - showActivityInfo, 66
 - showCurrentMaxInfo, 66
 - showCurrentWLInfo, 67
 - showDataInfo, 67
 - showPositionInfo, 67
 - showStatusMessage, 67
 - start_detectors, 68
 - TTM_IP_changed, 68
 - TTM_Port_changed, 69
 - UDPCOMsetdetector, 69
 - UDPCOMsetscanrange, 69
 - UDPCOMsetttmchannel, 70
 - UDPCOMsetttmcoincidencechannel, 70
 - update_spectrometer_info, 70
 - updateIntegrationtime, 71
- dinspect.h, 125
- dmath, 25
 - calc_alpha, 26
 - calc_angle_offset, 27
 - calc_angle_offset_without_stepsize, 27
 - calc_bandpass, 28
 - calc_bandpasses_in_scanrange, 28
 - calc_beta, 28
 - calc_coupling_efficiency, 29
 - calc_deviationangle, 29
 - calc_deviationangle_without_stepsize, 30
 - calc_how_many_edges_to_skip, 31
 - calc_lambda_from_alpha, 31
 - calc_number_of_steps_in_scanrange, 31
 - calc_range, 32
 - calc_steps_between_edges, 32
 - calc_steps_in_BP, 33
 - calc_stepsize, 33
 - calc_unneeded_stepsize, 34
 - find_max, 34, 35
 - fwhm_gauss, 35
 - normal_distribution, 36
 - recalc_steps_in_BP, 36
 - recalc_wavelength_at_step, 37
 - relate_maxpos_to_angle_difference, 37
- dmath.h, 126
- EPS
 - defines.h, 124
- edgewasfound
 - Motor, 83
- end_scan
 - Dinspect, 61

- error
 - helper, 38
- fallingedge
 - Motor, 84
- find_max
 - dmath, 34, 35
- fwhm_gauss
 - dmath, 35
- get_detector_data
 - Dinspect, 61
- get_onlysinglerun
 - CalibrationDialog, 45
- get_rough_wl
 - Motor, 84
- get_security_angle
 - CalibrationDialog, 45
- get_security_percentage
 - CalibrationDialog, 45
- get_steps_to_go
 - CalibrationDialog, 45
- get_steps_wo_measurement
 - CalibrationDialog, 46
- getAverageCount
 - PM100D, 97
- GetFrequencyFromSNSPDcounter
 - Dinspect, 61
- getPower
 - PM100D, 97
- GetPowerFromPM100D
 - Dinspect, 61
- getSpectrometerSettings
 - Motor, 85
- getSpectrometerSubversion
 - Dinspect, 62
- getSpectrometerType
 - Dinspect, 62
- getdsettings
 - Motor, 84
- gotonextedge
 - Motor, 85
- gotonextedgetimer
 - Motor, 91
- GraphSelectionChanged
 - Dinspect, 62
- handleError
 - Motor, 85
- helper, 38
 - error, 38
 - info, 38
 - message, 39
 - warning, 39
- helper.h, 128
- hometimer
 - Motor, 91
- info
 - helper, 38
- initActionsConnections
 - Dinspect, 62
- initVars
 - spectrometeroptions, 104
- isedge
 - Motor, 85
- isfallingedge
 - Motor, 86
- isopen
 - Motor, 86
- isrisingedge
 - Motor, 86
- LoadConfig
 - ConfigDinspect, 49
- ManualMotorControlDialog, 72
 - Connect_to_motor, 74
- manualmotorcontroldialog.h, 130
- message
 - helper, 39
- Motor, 75
 - command_microstep, 80
 - command_nextedge, 80
 - command_roughly_to_wl, 81
 - command_scanmove, 81
 - command_singlestep, 81
 - command_step, 81
 - command_to_start_pos, 83
 - CreatePositionLabel, 83
 - edgewasfound, 83
 - fallingedge, 84
 - get_rough_wl, 84
 - getSpectrometerSettings, 85
 - getdsettings, 84
 - gotonextedge, 85
 - gotonextedgetimer, 91
 - handleError, 85
 - hometimer, 91
 - isedge, 85
 - isfallingedge, 86
 - isopen, 86
 - isrisingedge, 86
 - motorstatusmessage, 86
 - risingedge, 87
 - roughwltimer, 92
 - sendsignalfallingedgedetected, 87
 - sendsignalarisingedgedetected, 87
 - sensordata, 87
 - showStatusMessage, 88
 - skip_edges, 89
 - skip_edges_rough, 89
 - spectrometer_info_updated, 89
 - startpostimer, 92
 - stop, 90
 - updateAskmotor, 90
 - updatesFirstStep, 90
 - updateScanIsRunning, 91

- write, 91
- motor.h, 130
- motor_is_at_starting_position
 - Dinspect, 62
- MotorSettingsDialog, 93
 - apply, 95
 - askmotor, 95
- MotorSettingsDialog::MotorSettings, 92
- motorfoundfallingedge
 - Dinspect, 63
- motorfoundinitedge
 - Dinspect, 63
- motorfoundrisingedge
 - Dinspect, 63
- motorstatusmessage
 - Motor, 86
- normal_distribution
 - dmath, 36
- on_DetectorComboBox_activated
 - Dinspect, 64
- PM100D, 96
 - getAverageCount, 97
 - getPower, 97
 - pyprint, 97
 - setAverageCount, 98
 - setSerialNumber, 98
- PI
 - defines.h, 125
- plot_savefile
 - Dinspect, 64
- pm100d.h, 131
- processCommands
 - UDPlistener, 114
- processlasermg
 - tunablelaser, 110
- pyprint
 - PM100D, 97
- RADTODEG
 - defines.h, 125
- read
 - tunablelaser, 110
- read_bp_data
 - spectrometeroptions, 104
- read_ce_data
 - spectrometeroptions, 104
- read_config
 - spectrometeroptions, 105
- read_stepsinbp_data
 - spectrometeroptions, 105
- read_wavelengthAtStep
 - spectrometeroptions, 106
- readPM100Dlin.dinspectPM100D, 71
- readPM100Dwin.dinspectPM100D, 71
- recalc_steps_in_BP
 - dmath, 36
- recalc_wavelength_at_step
 - dmath, 37
- recalc_with_dynamic_stepsize
 - Dinspect, 64
- relate_maxpos_to_angle_difference
 - dmath, 37
- ret_singlerun
 - CalibrationDialog, 46
- ret_understood
 - CalibrationDialog, 46
- ret_wantnewscan
 - CalibrationDialog, 46
- return_single_spectrometer
 - spectrometeroptions, 106
- returnType
 - spectrometeroptions, 106
- risingedge
 - Motor, 87
- roughwltimer
 - Motor, 92
- SNSPD_update_channel
 - Dinspect, 68
- SNSPD_update_coinc_channel
 - Dinspect, 68
- SNSPDcounter, 99
- save
 - Dinspect, 64
- saveAutomatically
 - Dinspect, 65
- SaveConfig
 - ConfigDinspect, 49
- SaveGraph
 - Dinspect, 65
- ScanRangeMinPositioning
 - Dinspect, 65
- sendsignalfallingedge detected
 - Motor, 87
- sendsignalrisingedge detected
 - Motor, 87
- sensordata
 - Motor, 87
- setAverageCount
 - PM100D, 98
- setDetector
 - UDPlistener, 114
- setPM100D_AverageCount
 - Dinspect, 65
- setScanrange
 - UDPlistener, 115
- setSerialNumber
 - PM100D, 98
- setTTMChannel
 - UDPlistener, 115
- setTTMCoincidenceChannel
 - UDPlistener, 115
- setup
 - tunablelaser, 110
- setupGraph

- Dinspect, 66
- setwl
 - tunablelaser, 111
- showActivityInfo
 - Dinspect, 66
- showCurrentMaxInfo
 - Dinspect, 66
- showCurrentWLInfo
 - Dinspect, 67
- showDataInfo
 - Dinspect, 67
- showPositionInfo
 - Dinspect, 67
- showStatusMessage
 - Dinspect, 67
 - Motor, 88
- skip_edges
 - Motor, 89
- skip_edges_rough
 - Motor, 89
- snsdcounter.h, 132
- Spectrometer, 100
- spectrometer_info_updated
 - Motor, 89
- spectrometeroptions, 102
 - initVars, 104
 - read_bp_data, 104
 - read_ce_data, 104
 - read_config, 105
 - read_stepsinbp_data, 105
 - read_wavelengthAtStep, 106
 - return_single_spectrometer, 106
 - returnType, 106
 - write_config, 107
- spectrometeroptions.h, 133
- start_detectors
 - Dinspect, 68
- startpostimer
 - Motor, 92
- stop
 - Motor, 90
- TTM_IP_changed
 - Dinspect, 68
- TTM_Port_changed
 - Dinspect, 69
- tunablelaser, 107
 - configdir_changed, 109
 - configfile_changed, 110
 - processlasermg, 110
 - read, 110
 - setup, 110
 - setwl, 111
 - write, 111
- tunablelaser.h, 134
- UDPCOMsetdetector
 - Dinspect, 69
- UDPCOMsetscanrange
 - Dinspect, 69
- UDPCOMsetttmchannel
 - Dinspect, 70
- UDPCOMsetttmcoincidencechannel
 - Dinspect, 70
- UDPlistener, 111
 - bind, 114
 - processCommands, 114
 - setDetector, 114
 - setScanrange, 115
 - setTTMChannel, 115
 - setTTMCoincidenceChannel, 115
 - UDPlistener, 113
- udplistener.h, 135
- update_spectrometer_info
 - Dinspect, 70
- updateAskmotor
 - Motor, 90
- updateIntegrationtime
 - Dinspect, 71
- updateIsFirstStep
 - Motor, 90
- updateScansRunning
 - Motor, 91
- version.h, 136
- warning
 - helper, 39
- write
 - Motor, 91
 - tunablelaser, 111
- write_config
 - spectrometeroptions, 107