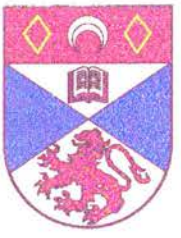
A Fourier-transform spectrometer based on a Wollaston prism: theoretical analysis and application to the detection of hydrogen sulphide

Johannes Courtial

A dissertation submitted in candidature for the degree of Master of Philosophy in the University of St. Andrews



November 1995

I, Johannes Courtial, hereby certify that this thesis, which is approximately 17000 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

date 27/11/95 signature of candidate .

I was admitted as a research student in October 1994; the higher study for which this is a record was carried out in the University of St. Andrews between 1994 and 1995.

date 27/11/95 signature of candidate

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of M.Phil. in the University of St. Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date 27 Nov 95 signature of supervisor

In submitting this thesis to the University of St. Andrews I wish access to it to be subject to the following conditions:

for a period of 2 years from the date of submission, the thesis shall be made available for use only with the consent of the Head of the department in which the work was carried out.

I understand, however, that the title and abstract of the thesis will be published during this period of restricted access; and that after the expiry of this period the thesis will be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright in the work not being affected thereby, and a copy of the work may be made and supplied to any bona fide library or research worker.

date 27/11/95 signature of candidate .

Abstract

This work concerns a Wollaston prism based Fourier-transform spectrometer and its applicability to the detection of selected pollutant gases. Within this project a theoretical study of the spectrometer design has been completed. Specific attention has been paid to the angular dependent path difference and the resulting visibility of the interferogram and field of view of the instrument. A spectrometer has been designed and developed for operation in the ultra-violet spectral region. Testing under laboratory conditions has shown sensitivity for the detection of hydrogen sulphide corresponding to 1ppm over a 6m path length.

Table of contents

Introduction

Recently, researchers at St. Andrews University have developed a novel Fourier-transform (FT) spectrometer based on Wollaston prisms [1; 2; 3; 4]. This work was awarded the 1994 NPL Metrology Prize. During the summer of 1994 the feasibility of applying this technology to the detection of nitrogen dioxide (NO_2) was successfully demonstrated [5]. On the basis of these results it was decided to develop an ultra-violet (UV) version of the instrument, with particular emphasis on the detection of hydrogen sulphide (H_2S). This thesis describes the work carried out within this project.

Chapter 1 gives an outline of the operating principle of Wollaston prism based Fourier-transform spectrometers.

Chapter 2 deals with theoretical aspects of the instrument design. As a Fourier-transform spectrometer it possesses the *Jacquinot advantage* over dispersive instruments, i.e. its light gathering capability (*étendue*) is typically 190 times greater [7, p.72]. The theoretical work in this thesis enables modelling of the optical properties of Wollaston prism spectrometers. To this end a computer program was written in the language *Mathematica* [13] that enables ray tracing in birefringent media. The propagation of a ray is calculated using equations derived in [10]. Particular attention is paid to the modelling of the field of view of Wollaston prism spectrometers. Using the model developed in this chapter, methods for increasing the field of view of such instruments have been identified [8]. The theoretical work also enables modelling of the spectrometer output, which can be an important part of theoretical feasibility studies.

Chapter 3 details the design of an ultra-violet Fourier-transform spectrometer and presents results obtained during the successful demonstration of a prototype spectrometer in June 1995 at Shell Thornton Research Centre [6]. An Apple Macintosh application program was written in the C programming language that reads data from the spectrometer and performs the processing according to chapter 2.

The appendices contain software developed within this project. Appendix A lists a program that allows general ray tracing in birefringent media. The program described in appendix B uses the ray tracing program to supply routines for the calculation of the angular-dependent path difference relation, spatial distribution of stationary points and field of view of Wollaston prism based spectrometers.

1. Wollaston prism FT spectrometers

A Wollaston prism is an optical component that consists of two geometrically similar wedge-shaped prisms of birefringent material. The optic axes in the wedges are perpendicular to the optical axis of the prism as well as to each other (fig. 1).

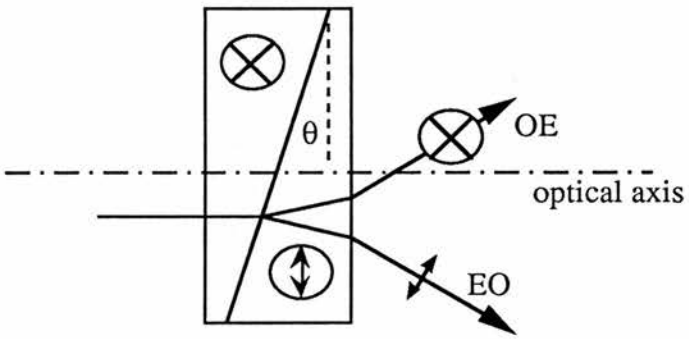


Figure 1: Side view of a Wollaston prism and a ray that is being split by it. The ray marked OE is ordinary in the left and extraordinary in the right wedge, the EO ray is extraordinary in the left and ordinary in the right wedge. The symbols ↕ and ⊗ indicate the orientation of the optic axis in birefringent materials and the polarisation direction of rays, respectively. The figure is drawn for a Wollaston prism fabricated from negative birefringent material.

Usually Wollaston prisms are used as birefringent beam splitters. An incident ray is split because the polarisation component of the ray that is ordinary in the first wedge becomes extraordinary in the second wedge and vice versa. Since usually the refractive indices as experienced by ordinary and extraordinary rays are different, at the interface between the wedges one of the rays changes from lower refractive index to higher refractive index and is therefore refracted towards the normal to the interface, whereas the other ray experiences a change from higher to lower refractive index and is refracted away from the normal.

If the Wollaston prism is to be used as a component within a beam splitting interferometer, a method needs to be adopted to make the two rays intersect again. There are several possible ways of doing this (fig. 2).

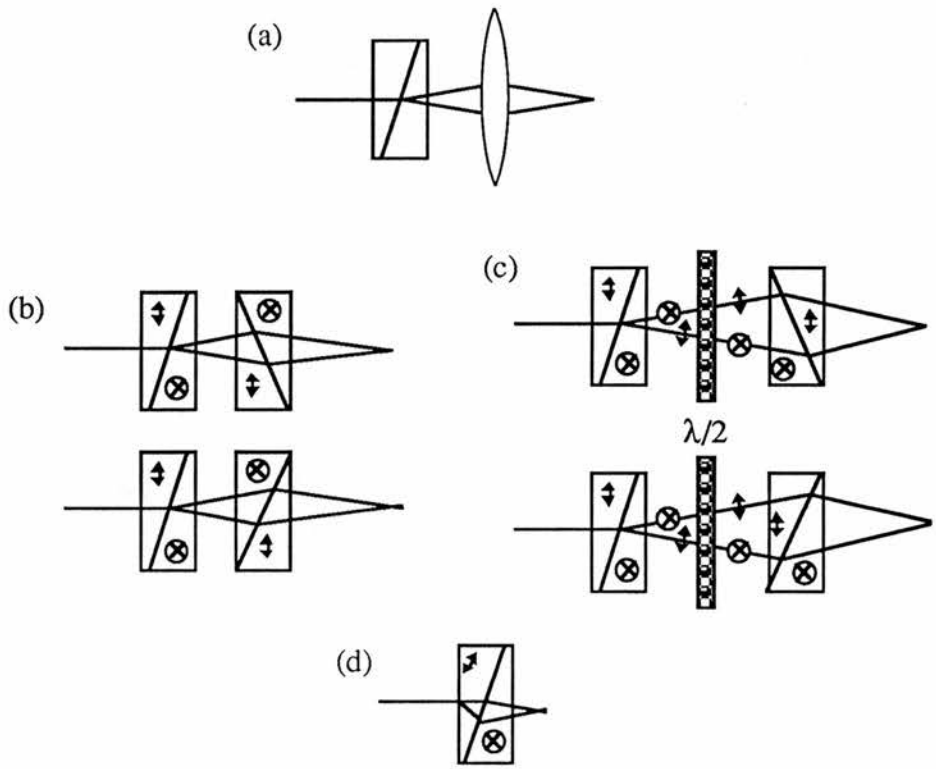


Figure 2: Possible designs for an interferometer based on or derived from Wollaston prisms.

(a) is the originally proposed design and uses a lens to focus the two orthogonally polarised rays to a position behind the lens [1]. In (b) the lens is replaced by a second Wollaston prism that refracts the rays so that they intersect behind the second prism [2]. (c) is similar to (b) other than the polarisation states are rotated between the prisms with a half-wave plate. The advantage of (c) is that it usually has a large field of view [9]. (d) is a modified Wollaston prism. The intersection between the rays is moved out of the prism by inclining the optic axis in one of the wedges [10; 11, p. 33].

In order to observe interference between the two rays, polarisers have to be placed before and after the birefringent elements.

In a Michelson interferometer, the path difference between the two rays is varied by moving one of the mirrors. If its output is recorded as a function of the path difference between the two arms, an interferogram is obtained that is the Fourier transform of the spectral distribution of the input light.

In a Wollaston prism interferometer, the path difference between the two polarisations at the intersection point is varied with the displacement of the input light ray. For design (a) in figure 2, for example, the relation between this path difference pd and the displacement l of the input light ray is

$$\mathrm{pd}(l) \approx 2 \cdot l \cdot (n_o - n_e) \tan\theta$$

[1], where n_o and n_e are the ordinary and extraordinary refractive indices in the Wollaston prism and θ is the inclination angle of the interface between the wedges. Therefore the Fourier transform of the spectral distribution of the input light can be obtained by recording the output of a Wollaston prism interferometer as a function of the displacement of the input ray. This can be done simultaneously for all path differences by illuminating the entire aperture of the Wollaston prism and using a detector array positioned in the plane of the ray intersections to record the interferogram.

1.1. The double Wollaston prism spectrometer

The methods presented in this work can be applied to all of the above types of interferometer. However, some of the results, for example the shape of the path difference relation, were found specifically for the double Wollaston prism design without the waveplate and do not necessarily apply to the other designs. The double Wollaston design is described in more detail in figure 3.

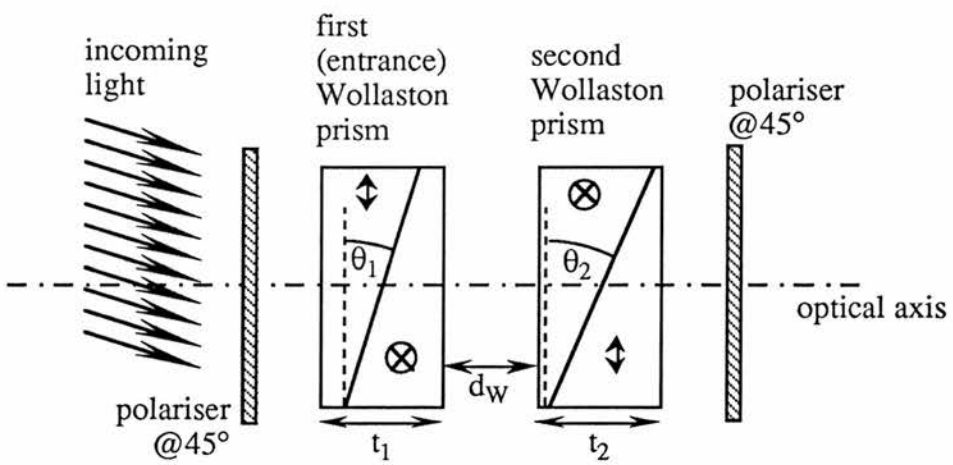


Figure 3: Schematic layout of the optics of a double Wollaston prism spectrometer and definition of basic design parameters. The θ_i are called *internal angles* of the Wollaston prisms. The camera would be positioned on the right.

2. The interferogram in a Wollaston prism based spectrometer

The aim of this chapter is to show how the interferogram recorded by the camera in a Wollaston prism based spectrometer can be calculated from the design parameters of the spectrometer. Additionally this chapter demonstrates how the spectrum of light falling into the spectrometer can be inferred from the recorded data.

The first section of this chapter considers how the path difference, at a particular detector position, between the two polarisation states varies with the input angle. The second section considers how this angular dependent path difference influences the visibility of fringes as produced by an extended source. The variation in fringe visibility with the position of the detector can be considered as a localisation of the fringes. The position of fringe localisation corresponds to the optimum position for the array detector if the spectrometer is to be used with an extended source.

2.1. The angular dependent path difference relation $pd_D(\alpha,\beta)$

2.1.1. The path difference relation

A Wollaston prism based Fourier-transform spectrometer has the property that for any two points L and D on either side of the spectrometer optics there are exactly two distinct light paths that intersect both L and D (see fig. 4).

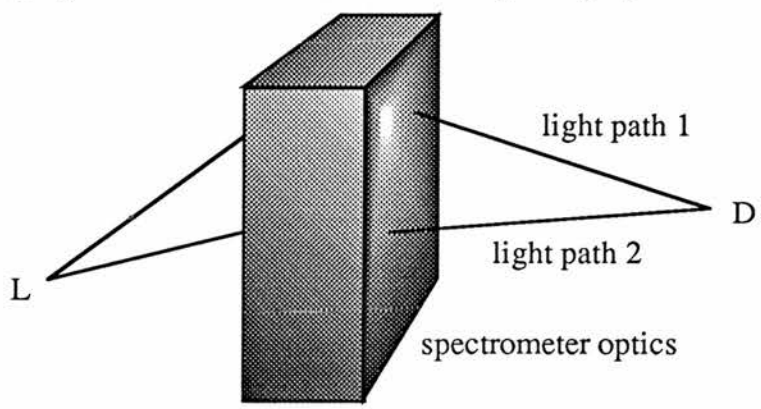


Figure 4: For every pair of points L, D on different sides of the spectrometer optics there exist exactly two physical light paths which intersect both L and D.

In Michelson interferometers a beam splitting mirror reflects one half of the impinging light into one path while the transmitted component takes the other one. In Wollaston prism spectrometers the first prism splits the light into two components, according to their polarisation state.

The optical path difference pd(L,D) can be defined as

pd(L,D) = (optical path length between L and D via light path 1)
– (optical path length between L and D via light path 2).

pd(L,D) plays an important role in determining the signal recorded by a point light detector element at D behind a set of spectrometer optics which is lit by a point light source at L.

2.1.2. Definition of basic directions and input angles α, β

It is convenient to define the orientation of the spectrometer in a cartesian coordinate system. Here the directions are chosen as indicated in fig. 5.

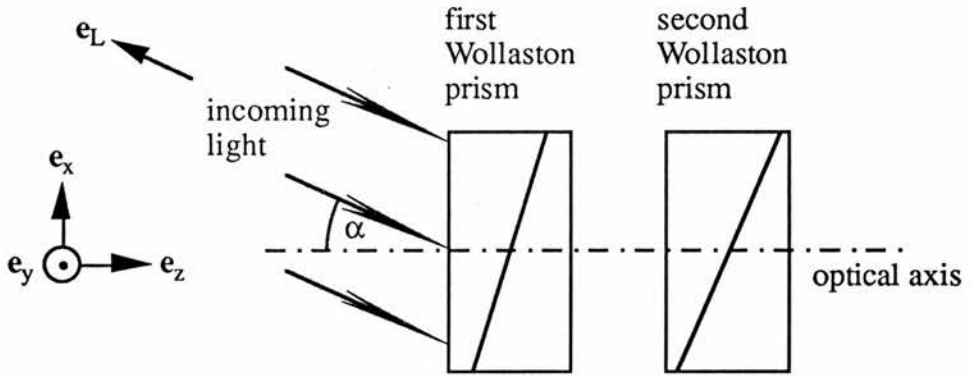


Figure 5: The orientation of the spectrometer optics in a x, y, z coordinate system and the input angle α. The other input angle, β, lies in the yz plane. The directions of the optic axes will be indicated in terms of these x and y directions.

Additionally, input angles α and β of light incident from a direction parallel to $\mathbf{e}_L$ are defined according to

$$\frac{\mathbf{e}_L \cdot \mathbf{e}_x}{\mathbf{e}_L \cdot \mathbf{e}_z} = -\tan\alpha$$

and

$$\frac{\mathbf{e}_L \cdot \mathbf{e}_y}{\mathbf{e}_L \cdot \mathbf{e}_z} = -\tan\beta$$

(see also fig. 5).

The size of a solid angle element $d\Omega(\alpha,\beta)$ for these input angles is

$$d\Omega(\alpha,\beta) = \frac{16\sqrt{2}\cos\alpha\cos\beta}{\left(6+2\cos 2\alpha+2\cos 2\beta-\cos 2(\alpha-\beta)-\cos 2(\alpha+\beta)\right)^{3/2}} d\alpha \cdot d\beta. \qquad (1)$$

2.1.3. The angular path difference relation $pd_D(\alpha,\beta)$

For practical reasons, i.e. computability and our present interest in the use of the spectrometer situated 'far' from the light source, the following considerations have been made for an infinitely distant light source.

It is convenient to assign a path difference $pd_D(\alpha,\beta)$ to the input angles (α,β), at which an infinitely distant point $L(\alpha,\beta)$ is seen by the spectrometer, rather than to the position of the point itself, through the definition

$$pd_D(\alpha,\beta) \equiv pd(L(\alpha,\beta),D).$$

2.1.4. Computation of $pd_D(\alpha,\beta)$

The main problem arising in this computation is finding the two light paths that intersect both L and D. For finite distances between L and the Wollaston prism, the light paths have to be found by iterative ray tracing or lengthy analytical methods. For an infinitely distant point source L, the two incoming rays are parallel. In an optical system that has no optically active materials and no curved surfaces, the output directions depend only on the incident angle and not on the lateral position. Therefore the paths of the two component rays that intersect at D can be found by tracing backwards from D the two components of any light ray impinging from the desired input angles after transmission through the optical system (see fig. 6).

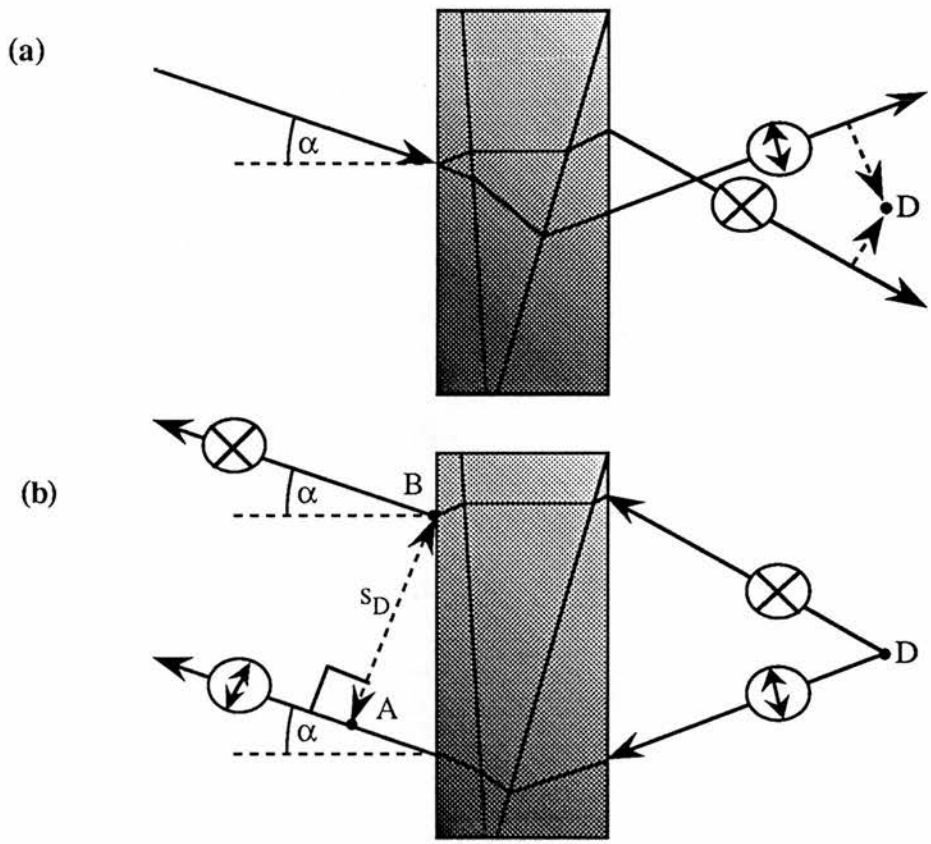


Figure 6: Practical computation of the path difference $pd_D(\alpha, \beta)$, drawn for $\beta=0$. The first step consists of tracing a ray, that impinges from the specified input angles (α, β), through the spectrometer optics, thereby splitting into two rays with mutually perpendicular polarisation (a). The second step is tracing two rays, originating at D, each with the same polarisation state and opposite ray direction as one of the rays resulting from (a), backwards through the system (b). The path difference $pd_D(\alpha,\beta)$ can then be computed as the difference between the optical path lengths between A and D and B and D, respectively. The separation between the two rays, s, which is also indicated in (b), is important for the practical calculation of the plane of fringe localisation.

2.2. Stationary points and the form of the angular dependent path difference relation

2.2.1. Stationary positions

If a point D behind the spectrometer satisfies the conditions

$$\left.\frac{\partial pd_D(\alpha',\beta)}{\partial \alpha'}\right|_{\alpha'=\alpha} = 0 = \left.\frac{\partial pd_D(\alpha,\beta')}{\partial \beta'}\right|_{\beta'=\beta} \qquad (2)$$

it is called *stationary* for the input angles α and β and labelled $D_{stationary}(\alpha,\beta)$. It shall be shown later that points $D_{stationary}$ form the plane to which fringes are localised. It is useful to define the distance Δz of a point D from a stationary point $D_{stationary}(0°,0°)$ which is displaced in the z direction (fig. 7).

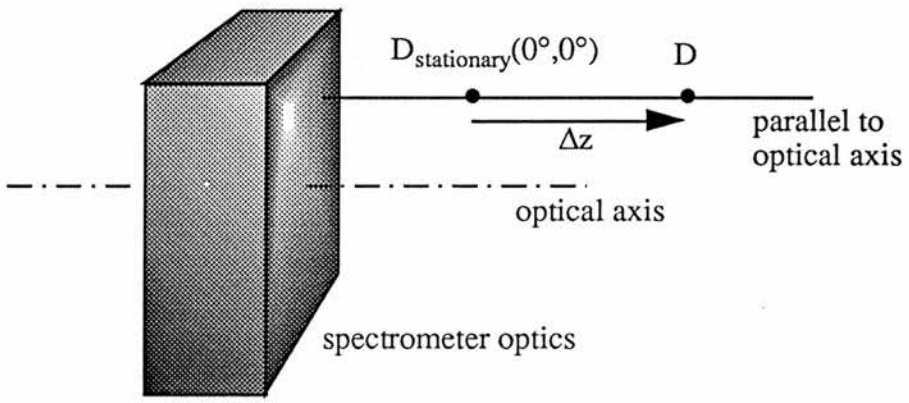


Figure 7: Definition of the displacement Δz of a point D along a parallel to the optical axis from a stationary point $D_{stationary}(0°,0°)$.

2.2.2. The form of the angular dependent path difference relation in a Wollaston prism spectrometer

With the help of the ray tracing program written within this project it transpired that in almost all cases[1] sections along α=const. and β=const. through path difference relations $pd_D(\alpha,\beta)$ corresponding to Wollaston spectrometers are, to a very good approximation, parabolic (see fig. 8).

[1] i.e. for all designs for which the shape of the path difference relation is clearly either saddle shaped or bowl shaped

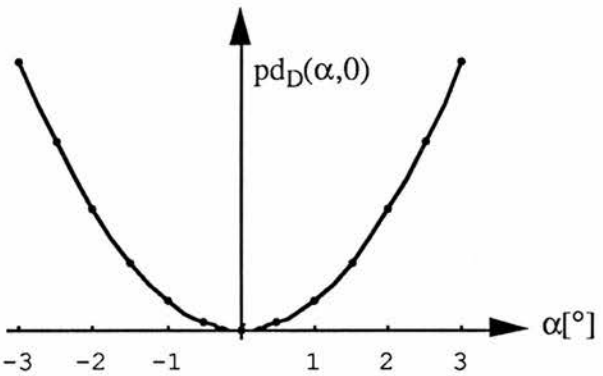


Figure 8: Path differences[1] calculated for various α for β=0 (dots) and a fitted parabola (solid line). The average deviation of the points in this graph from the parabola relative to the total vertical extent of the graph is less than 0.03%. Other spot checks confirm this observation.

Depending upon the sign of the product of the 2nd derivatives $\partial^2 pd_D(\alpha,\beta)/\partial\alpha^2$ and $\partial^2 pd_D(\alpha,\beta)/\partial\beta^2$, path difference relations can be saddle shaped, bowl shaped or take on intermediate shapes. Figure 9 shows an example of the angular path difference relations for various spectrometer designs, which were calculated for the on-axis stationary positions $D_{stationary}(0°, 0°)$. The transformation between the various forms of path difference relations was achieved by varying only the internal angle in the second Wollaston prism, θ_2, while leaving all the other design parameters unchanged[2].

---

[1]These were calculated for light with a wavelength of 633nm shining on a spectrometer fabricated from calcite with internal angles θ_1=1.3° and θ_2=2.1° and a separation of d_W=3mm between the Wollaston prisms of thicknesses $t_1=t_2$=4mm (see fig.3). The orientations of the optic axes in the wedges were specified as x,y,y,x.

[2]The design specifications are: both Wollaston prisms are fabricated from quartz; $t_1=t_2$=4mm; d_W=10mm; θ_1= 8° in (a), 12° in (b) and 18° in (c), respectively;θ_2=25° in all cases. The orientation of the optic axes in the wedges is x,y,y,x.

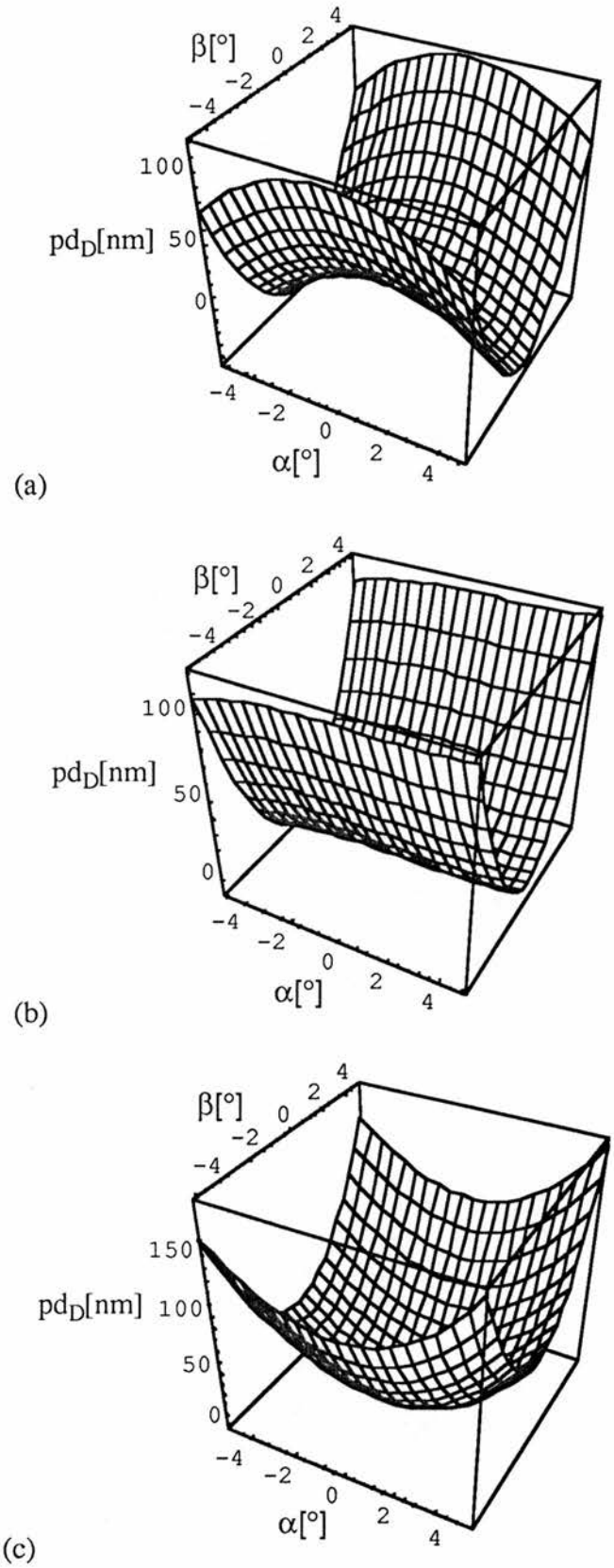


Figure 9: Three basic shapes of path difference relations for double Wollaston spectrometers. In some cases the saddle (a) can, upon variation of one or more design parameters, be turned via a valley (b) into a bowl (c).

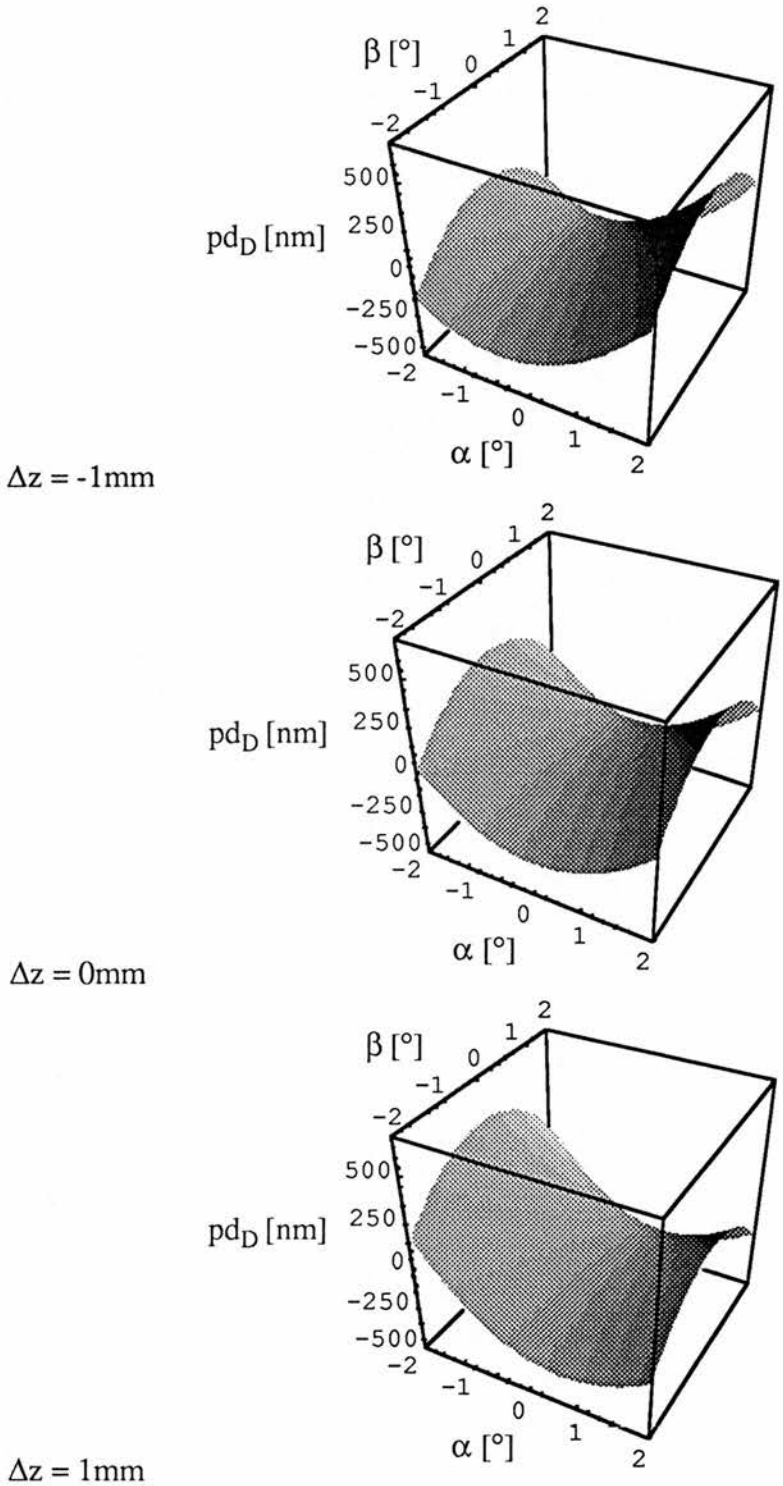


Figure 10: The displacement of the path difference relation corresponding to on-axis positions D which are displaced from the on-axis stationary position $D_{stationary}(0°, 0°)$ by Δz, whereby a positive Δz corresponds to a displacement away from the spectrometer optics. What looks like a rocking movement of the saddle is in fact a displacement (compare next figures).

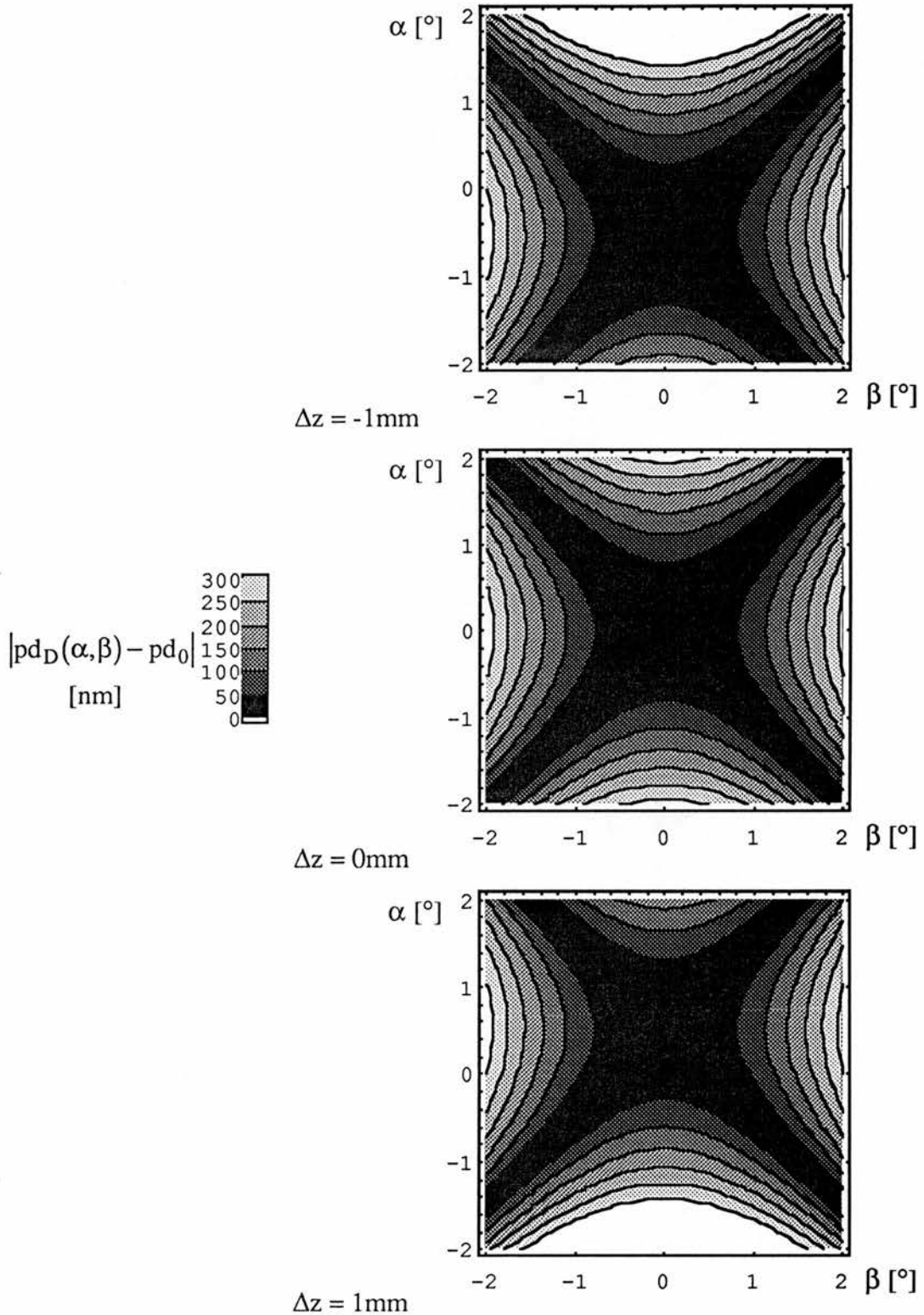


Figure 11: The displacement of the path difference relation in the previous plot can be seen more easily in these contour plots. The lines correspond to different absolute values of the difference between $pd_D(\alpha,\beta)$ and pd_0, the value of $pd_D(\alpha,\beta)$ in the centre of the saddle where pd_D is stationary.

It transpires that as Δz is varied, the path difference relations are almost identical in shape but displaced relative to each other in α direction (fig. 10-12[1]). The displacement of the path difference relation is approximately proportional to Δz.

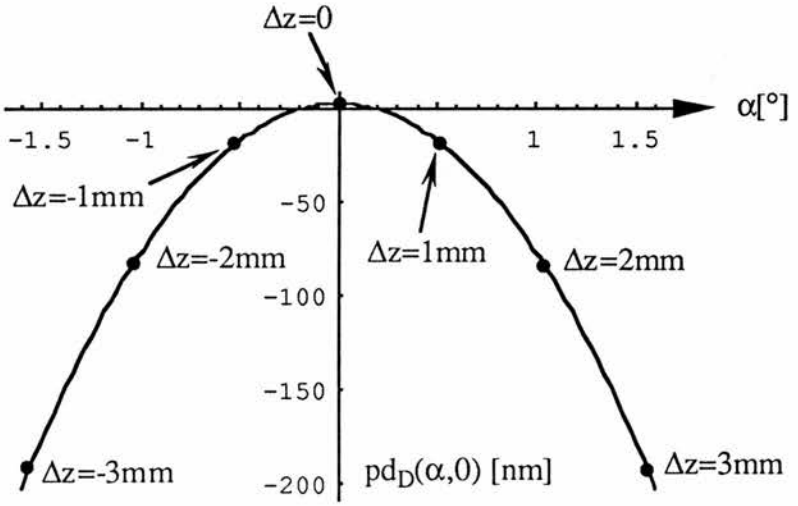


Figure 12: The curve followed by the centre of the saddle from the previous figures as Δz is varied. This curve is approximately parabolic.

2.2.2.1. Experimental observation of the angular dependent path difference

The shift of the path difference relation can be observed experimentally.

Light emanating from D and interfering at L has of course the same path difference as light emanating from L and interfering at D. This is the principle of reversibility ([12], p.70). Therefore, if light from a point light source positioned at D behind the spectrometer optics is made to interfere at a point $L(\alpha,\beta)$, its phase difference $\Delta\phi(\alpha,\beta)=\left(\left(2\pi\, pd_D(\alpha,\beta)/\lambda\right) \bmod 2\pi\right)$ is a measure of $pd_D(\alpha,\beta)$. By positioning L in the back focal plane of a lens, L is effectively moved into the far field.

A possible way of measuring $\Delta\phi(\alpha,\beta)$ is to record the interference pattern which is generated in the camera plane by the setup in fig. 13.

---

[1]These figures were calculated for light with a wavelength of 633nm shining on a spectrometer fabricated from calcite with internal angles $\theta_1=1.3°$ and $\theta_2=2.1°$ and a separation of $d_W=3mm$ between the Wollaston prisms of thicknesses $t_1=t_2=4mm$. The orientation of the optic axes in the wedges was specified as x,y,y,x.

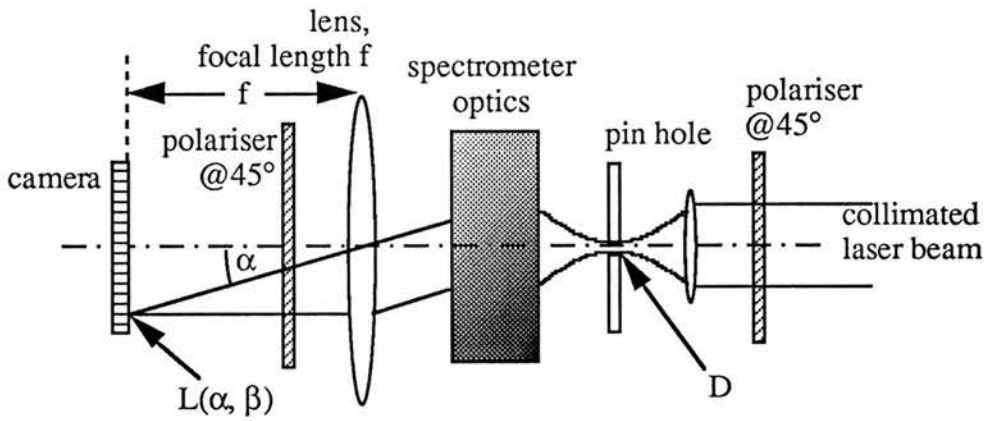


Figure 13: Experimental arrangement for measuring the phase difference $\Delta\phi(\alpha, \beta)$ and therefore indirectly $pd_D(\alpha, \beta)$.

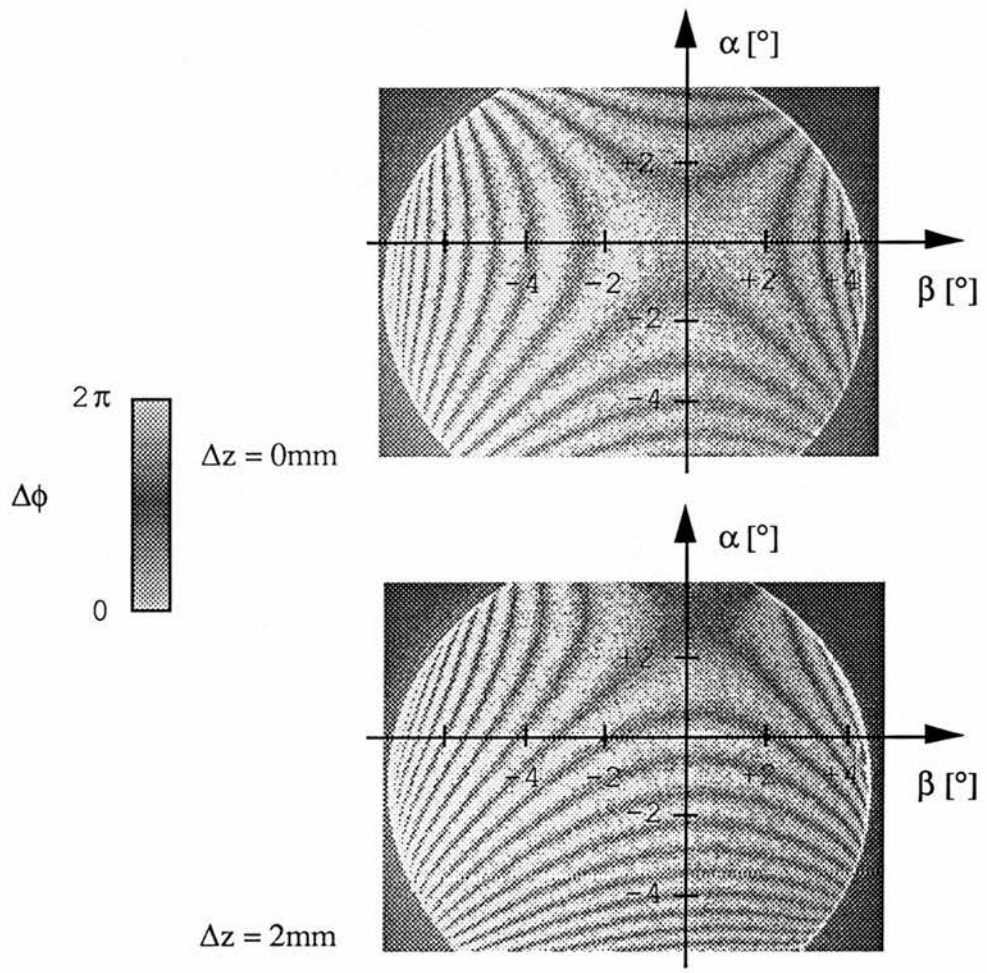


Figure 14: Interferograms obtained with a setup described above. Note that the form and relative displacement of these interference patterns matches those in the next figure. The angles α and β are mapped into the camera plane such that the distance from the optical axis in the α and β direction is proportional to sin(α) and sin(β), respectively.

The experimental accuracy required to make direct comparison to the phases is difficult to achieve. However, the form of the obtained interferograms in addition to

their relative displacement with Δz can be compared with the predicted interferograms.

Measured and calculated interferograms[1] are shown in figs 14 and 15, respectively.

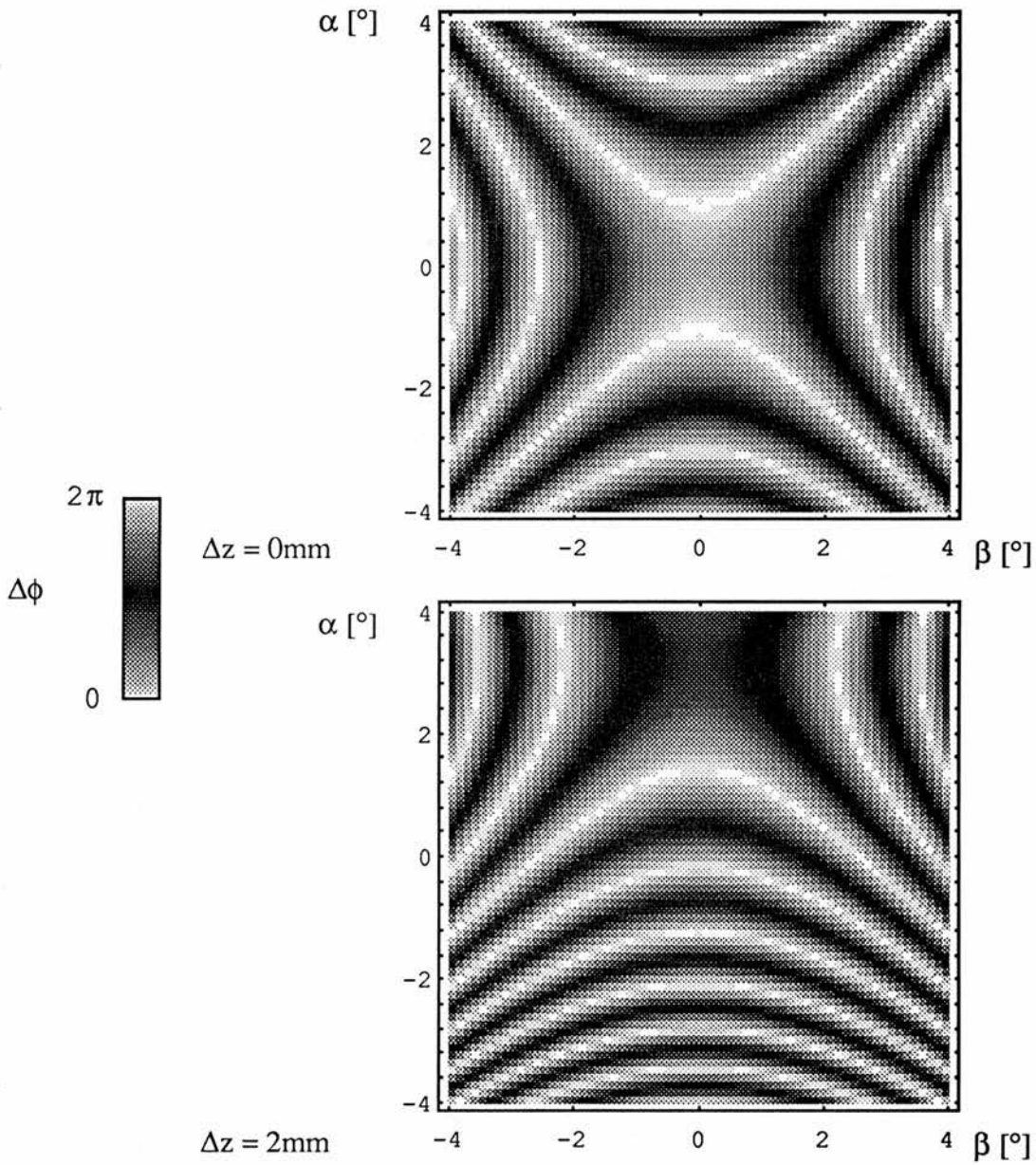


Figure 15: The predicted versions of the measured interferograms from the last figure.

---

[1]The specifications of the spectrometer optics design are: Wollaston prisms fabricated from calcite, orientation of optic axes x,y,y,x, θ_1=3.6°, θ_2=5.9°, t_1=t_2=4mm, d_W=5.8mm; an imaging lens with f=30mm, a "Pulnix TM520" camera with CCD chip dimensions of 6.47mm by 4.83mm, and a pin hole with a diameter of 5μm were used.

2.2.2.2. Properties of Wollaston prism based spectrometers designed to have a valley shaped path difference relationship

A design which produces a valley shaped path difference relation has some interesting properties, such as high fringe visibility (see later chapters). This is a useful attribute if it can be combined with other design requirements.

In parameter space, valley shaped path difference relations mark the transition region between saddle shaped and bowl shaped pd relations, i.e. they occur when one of the second derivatives of $pd_D(\alpha,\beta)$, $\partial^2 pd_D(\alpha,\beta)/\partial\alpha^2$ or $\partial^2 pd_D(\alpha,\beta)/\partial\beta^2$, changes sign as the design is varied. This behaviour corresponds to specific spectrometer designs, since for one of the second derivatives of $pd_D(\alpha,\beta)$ to change sign this second derivative must take positive as well as negative values within this design (see table 1).

orientation of optic axes in wedges	x,y,y,x	y,x,x,y
$\operatorname{sign}\frac{\partial^2 pd_D(\alpha,\beta)}{\partial\alpha^2}$	+ and –	+
$\operatorname{sign}\frac{\partial^2 pd_D(\alpha,\beta)}{\partial\beta^2}$	+	–

Table 1: The sign of the second order derivatives $\partial^2/\partial\alpha^2$ and $\partial^2/\partial\beta^2$ of the path difference relation for any point positioned behind two Wollaston prisms fabricated from positively birefringent crystals (like quartz), whose optic axes in the four wedges are oriented as indicated. For a negative crystal the signs are reversed.

2.3. Visibility of the fringes produced by the interferometer

Consider a fringe pattern I(x). Let I_{top} and I_{bottom} be the functions defining the envelope and let the oscillating part of the fringe pattern be given by $f_{osc}(x)$, so

$$I(x) = I_{bottom}(x) + \left(I_{top}(x) - I_{bottom}(x)\right)\cdot f_{osc}(x)$$

(see fig. 14(a)). The *visibility* at x is defined as

$$V(x) = \frac{I_{top}(x) - I_{bottom}(x)}{I_{top}(x) + I_{bottom}(x)}.$$

It can range between 0 (if $I_{bottom}(x)=I_{top}(x)$) and 1 (if $I_{bottom}(x)=0$).

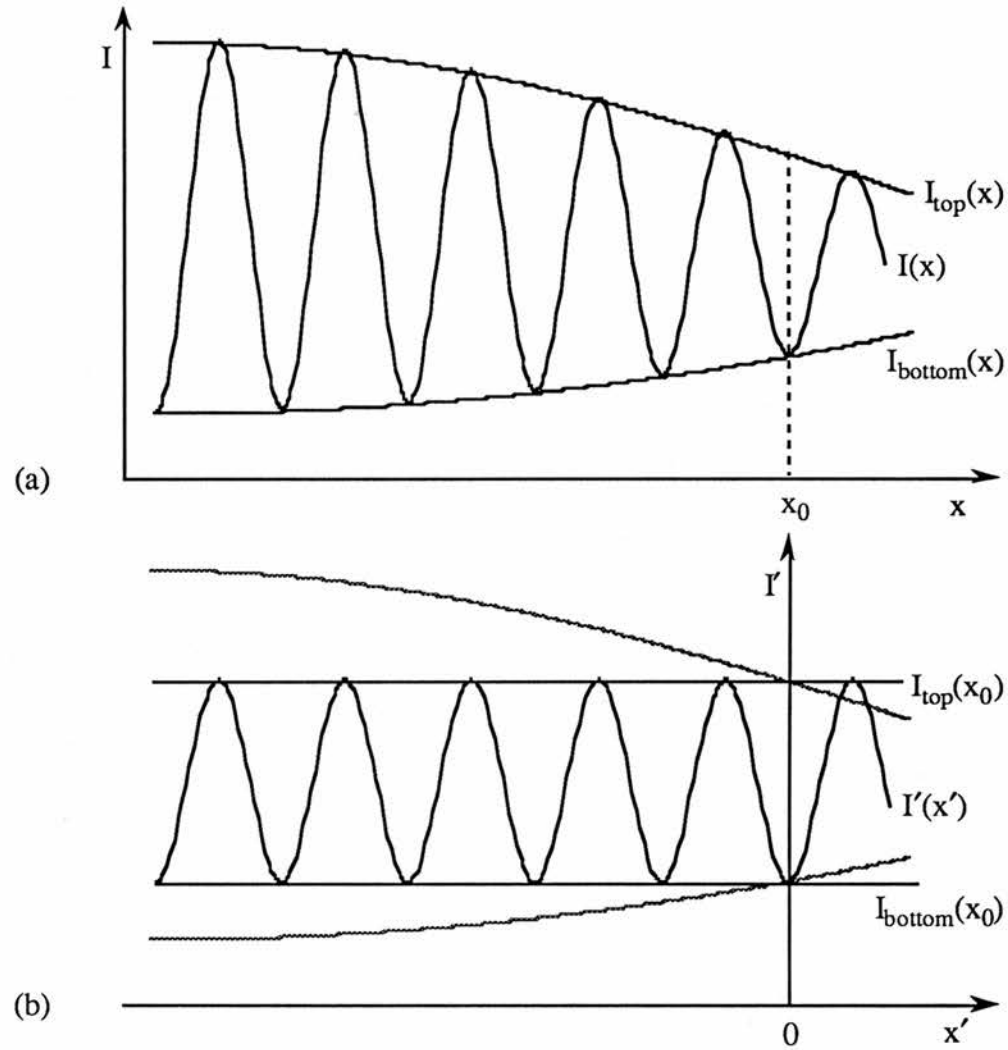


Figure 16: Definition of (a) I(x) and (b) $I'_{x_0}(x')$.

It is sometimes convenient to calculate the value of the upper and lower envelope at a value x_0 by calculating the extrema of the *phased intensity at x_0*,

$$I'_{x_0}(x') = I_{bottom}(x_0) + \left(I_{top}(x_0) - I_{bottom}(x_0)\right) \cdot f_{osc}(x_0 + x'),$$

where x' determines the phase of the purely oscillatory factor (see fig. 16(b)).

In this notation the visibility of fringes at D due to light in the wavelength range $[\lambda, \lambda+d\lambda]$, which is falling into the spectrometer with an angular distribution $i_\lambda^{lamp}(\alpha,\beta)$, can be written as

$$V_\lambda(D) = \frac{\max_{x'}\left(i'_{\lambda,D}(x')\right) - \min_{x'}\left(i'_{\lambda,D}(x')\right)}{\max_{x'}\left(i'_{\lambda,D}(x')\right) + \min_{x'}\left(i'_{\lambda,D}(x')\right)},$$

where $i'_{\lambda,D}(x')$ is the phased intensity at D due to light in the spectral range $[\lambda, \lambda+d\lambda]$. The corresponding (unphased) fringe pattern $i_\lambda(D)$ is the linear superposition over the solid angle range of the light source of the fringe patterns $i_{\lambda,(\alpha,\beta)}(D)$, which are due to light in the spectral range $[\lambda, \lambda+d\lambda]$ from an

infinitesimal solid angle element $d\Omega(\alpha,\beta)$. These individual fringe patterns have the form

$$i_{\lambda,(\alpha,\beta)}(D) \propto i_{\lambda}^{lamp}(\alpha,\beta)\left(1+\cos\frac{2\pi \cdot pd_D(\alpha,\beta)}{\lambda}\right) \qquad (3)$$

[1, equation (13)], and therefore unit visibility. Therefore $i_\lambda(D)$ is a superposition of fringe patterns each with unit visibility. However, addition of fringe patterns, each with unit visibility, does of course not necessarily imply unit total visibility (see fig. 17).

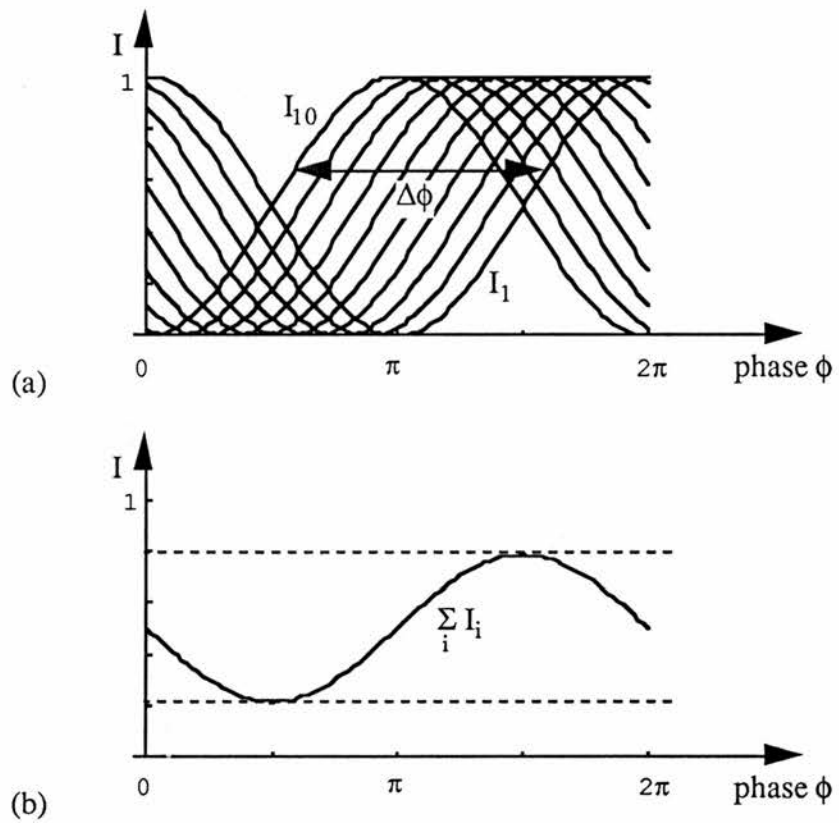
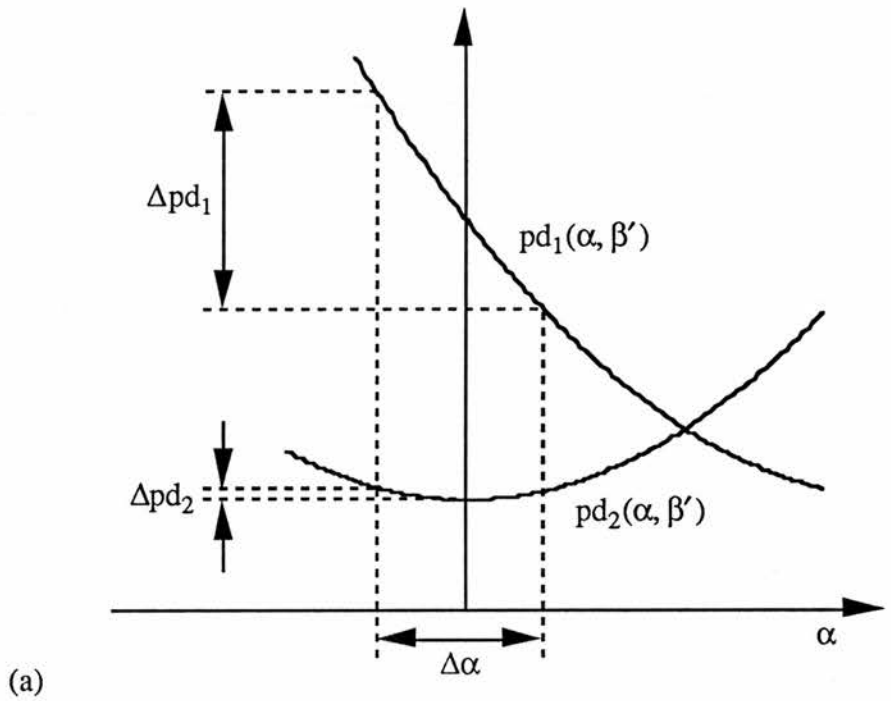
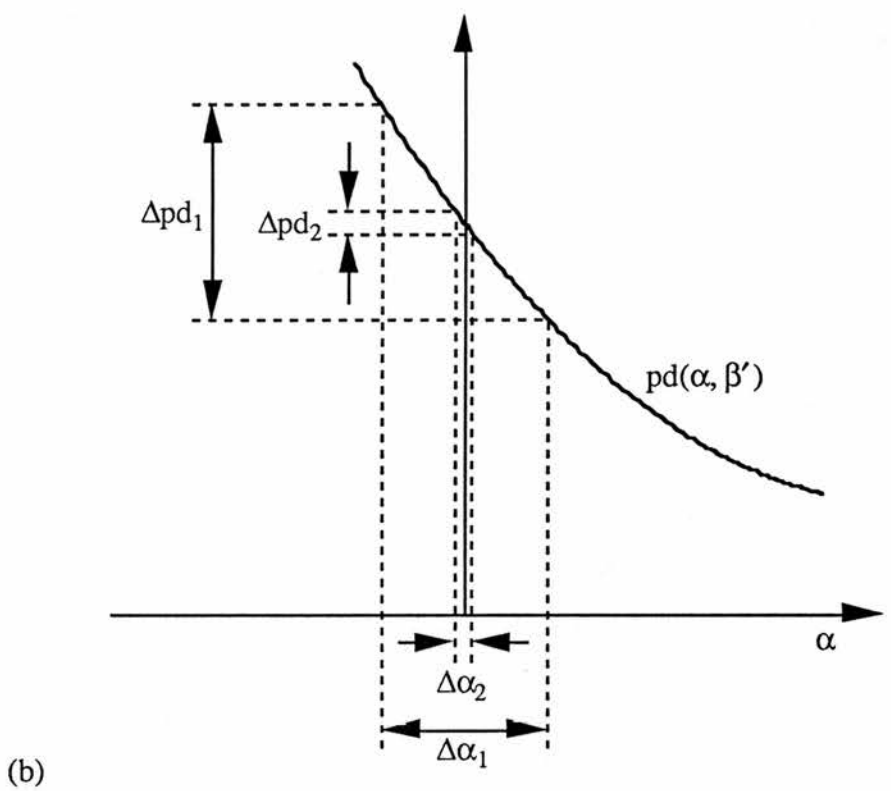


Figure 17: (a) 10 fringe patterns I_1, ..., I_{10}, each with unit visibility, which are displaced with respect to each other, their phase differences covering a range of 0..$\Delta\phi$. Because of this range of phases the visibility of their normalised superposition is diminished (b).

The visibility of the sum of individual fringe patterns decreases as the range of their phase differences increases. In the case of our spectrometer, where the phase of the individual fringe patterns is given by $2\pi pd_D(\alpha,\beta)/\lambda$, the range of phases is proportional to the range of $pd_D(\alpha,\beta)$ over the angular extent of the light source (see fig. 18).



(a)



(b)

Figure 18: (a) Two possible path difference relations $pd_1(\alpha,\beta)$ and $pd_2(\alpha,\beta)$, plotted for $\beta=\beta'$. A light source of angular extent $\Delta\alpha$ would produce fringes of much higher visibility at a point with an associated path difference relation $pd_2(\alpha,\beta)$ than at a point with path difference relation $pd_1(\alpha,\beta)$ because of the much smaller extent of phases. (b) By the same token, a light source with a smaller angular extent $\Delta\alpha_2$ produces fringes with a higher visibility than a larger source with an angular extent $\Delta\alpha_1$.

With $i'_{\lambda,(\alpha,\beta),D}(x')$, the phased version of $i_{\lambda,(\alpha,\beta)}(D)$, the phased intensity at D due to light in the wavelength range $[\lambda, \lambda+d\lambda]$, $i'_{\lambda,D}(x')$, becomes

$$i'_{\lambda,D}(x') = \int_{2\pi} i_{\lambda,(\alpha,\beta),D}(x')d\Omega(\alpha,\beta).$$

Substitution of the expression for $i_{\lambda,(\alpha,\beta)}(D)$ and transformation as follows yields a useful result:

$$\begin{aligned} i'_{\lambda,D}(x') &= \int_{2\pi} i'_{\lambda,(\alpha,\beta),D}(x')d\Omega(\alpha,\beta) \\ &\propto \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)\left(1+\cos\left(2\pi\frac{pd_D(\alpha,\beta)}{\lambda}+x'\right)\right)d\alpha\cdot d\beta \\ &= \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)\cdot d\alpha\cdot d\beta + \mathrm{Re}\left(\int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)e^{i\left(2\pi\frac{pd_D(\alpha,\beta)}{\lambda}+x'\right)}d\alpha\cdot d\beta\right) \\ &= \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)\cdot d\alpha\cdot d\beta + \mathrm{Re}\left(e^{ix'}\int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)e^{i2\pi\frac{pd_D(\alpha,\beta)}{\lambda}}d\alpha\cdot d\beta\right). \end{aligned}$$

With the definitions

$$i_{\lambda,total}^{lamp} \equiv \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)d\alpha d\beta \qquad (4)$$

and

$$t_{interference}(D) \equiv \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{lamp}(\alpha,\beta)e^{i2\pi\frac{pd_D(\alpha,\beta)}{\lambda}}d\alpha d\beta \qquad (5)$$

the expression for $i'_{\lambda,D}(x')$ becomes

$$i'_{\lambda,D}(x') \propto i_{\lambda,total}^{lamp} + \mathrm{Re}\left(e^{ix'}t_{interference}(D)\right).$$

The first term, $i_{\lambda,total}^{lamp}$, gives the total intensity in the wavelength range $[\lambda, \lambda+d\lambda]$ that is falling into the spectrometer. $t_{interference}$ in the second term is just a complex number, and therefore the second term yields in the expression for the visibility

$$\min_{x'}\left(\mathrm{Re}\left(t_{interference}(D)e^{ix'}\right)\right) = -\left|t_{interference}(D)\right|$$

and

$$\max_{x'}\left(\mathrm{Re}\left(t_{interference}(D)e^{ix'}\right)\right) = \left|t_{interference}(D)\right|.$$

Hence as result for the visibility

$$V_\lambda(D) = \frac{\left|t_{interference}(D)\right|}{i_{\lambda,total}^{lamp}}$$

is obtained.

This is a key result and will be used below to calculate the visibility for various forms of path difference relations.

2.3.1. Fringe visibility for a linear angular path difference relation

Here we study one of the cases where an elegant expression for the visibility can be easily written down. Another such case is a quadratic path difference relation.

The general linear path difference relation has the form

$$pd_D(\alpha,\beta) = a\alpha + b\beta + c,$$

where a, b and c are parameters. The interference term becomes

$$t_{\text{int erference}}(D) = \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{\text{lamp}}(\alpha,\beta)e^{i2\pi pd_D(\alpha,\beta)/\lambda}d\alpha\cdot d\beta$$

$$= e^{i\frac{2\pi}{\lambda}c}FT_{\alpha,\beta}\left\{j_\lambda^{\text{lamp}}(\alpha,\beta)\right\}(2\pi a/\lambda, 2\pi b/\lambda),$$

where

$$j_\lambda^{\text{lamp}}(\alpha,\beta) = \begin{cases} 2\pi\cdot i_\lambda^{\text{lamp}}(\alpha,\beta) & \text{if } |\alpha|,|\beta| \le \pi/2 \\ 0 & \text{everywhere else} \end{cases}.$$

$FT_{\vartheta,\varphi}\{f(\vartheta,\varphi)\}(x,y)$ denotes the 2-dimensional Fourier transform of the function $f(\vartheta,\varphi)$, evaluated at values x and y.

Moreover, a similar calculation yields

$$i_{\lambda,\text{total}}^{\text{lamp}} = \int_{-\pi/2}^{\pi/2}\int_{-\pi/2}^{\pi/2} i_\lambda^{\text{lamp}}(\alpha,\beta)\cdot d\alpha\cdot d\beta = FT_{\alpha,\beta}\left\{j_\lambda^{\text{lamp}}(\alpha,\beta)\right\}(0,0),$$

and the expression for the visibility becomes

$$V_\lambda(D) = \frac{\left|FT_{\alpha,\beta}\left\{j_\lambda^{\text{lamp}}(\alpha,\beta)\right\}(2\pi a/\lambda, 2\pi b/\lambda)\right|}{FT_{\alpha,\beta}\left\{j_\lambda^{\text{lamp}}(\alpha,\beta)\right\}(0,0)}.$$

2.3.2. Visibility for a quadratic angular path difference relationship

Similar to the linear case, a quadratic path difference relation

$$pd_D(\alpha,\beta) = a(\alpha - A)^2 + b(\beta - B)^2 + c,$$

where a, b, c, A and B are parameters, yields

$$t_{\text{int erference}}(D) = e^{i\frac{2\pi}{\lambda}c}FT_{\alpha,\beta}\left\{k_\lambda^{\text{lamp}}(\alpha,\beta)\right\}(2\pi a/\lambda, 2\pi b/\lambda) \qquad (6)$$

and

$$i_{\lambda,\text{total}}^{\text{lamp}} = FT_{\alpha,\beta}\left\{k_\lambda^{\text{lamp}}(\alpha,\beta)\right\}(0,0).$$

Here

$$k_\lambda^{\text{lamp}}(\vartheta,\varphi) = \begin{cases} k_\lambda^{+,\text{lamp}}(\vartheta,\varphi) & \text{if } \vartheta,\varphi \ge 0 \\ 0 & \text{everywhere else} \end{cases},$$

$$k_\lambda^{+,\text{lamp}}(\vartheta,\varphi) = \frac{\pi}{2\sqrt{\vartheta\varphi}}\left(j_\lambda^{\text{lamp}}\left(-\sqrt{\vartheta},-\sqrt{\varphi}\right) + j_\lambda^{\text{lamp}}\left(-\sqrt{\vartheta},+\sqrt{\varphi}\right) + \right.$$

$$\left. + j_\lambda^{\text{lamp}}\left(+\sqrt{\vartheta},-\sqrt{\varphi}\right) + j_\lambda^{\text{lamp}}\left(+\sqrt{\vartheta},+\sqrt{\varphi}\right)\right)$$

and

$$j_\lambda^{lamp}(\alpha',\beta') = \begin{cases} i_\lambda^{lamp}(\alpha'+A,\beta'+B) & \text{if } |\alpha'+A|,|\beta'+B| \le \pi/2 \\ 0 & \text{everywhere else} \end{cases}.$$

ϑ and φ are dummy parameters and play the roles of $(\alpha-A)^2$ and $(\beta-B)^2$, respectively. So the expression for the visibility becomes

$$V_\lambda(D) = \frac{\left|FT_{\vartheta,\varphi}\left\{k_\lambda^{lamp}(\vartheta,\varphi)\right\}(2\pi a/\lambda, 2\pi b/\lambda)\right|}{FT_{\vartheta,\varphi}\left\{k_\lambda^{lamp}(\vartheta,\varphi)\right\}(0,0)}.$$

Since quadratic path difference relations are very good approximations to many pd relations occurring in Wollaston prism designs (as previously), this result is elaborated upon in the following section.

2.3.2.1. Calculated results for quadratic angular path difference relations

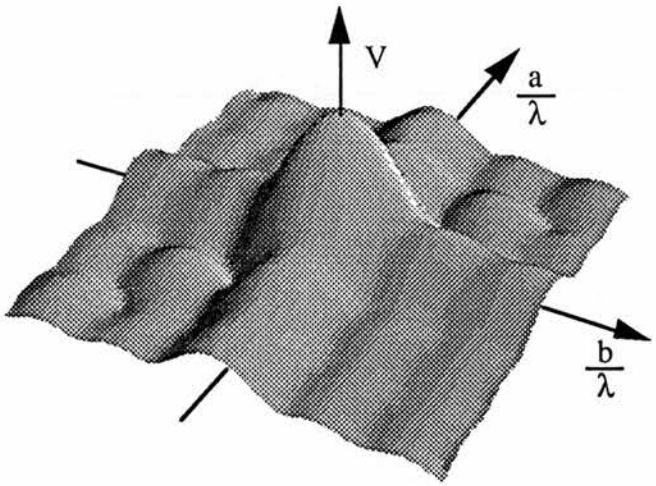
These results were calculated for a uniform circular light source of angular radius r=1°, which is centred around the optical axis. To a good approximation, they scale with r^1.

Figures 19 and 20 show the influence of the wavelength of the light and the parameters of the parabolic pd relation, whereas figures 21 and 22 concentrate on the variation of fringe visibility with the displacement of the parabola, which is proportional to the displacement, Δz, of the camera (compare figs 7 and 10-12).

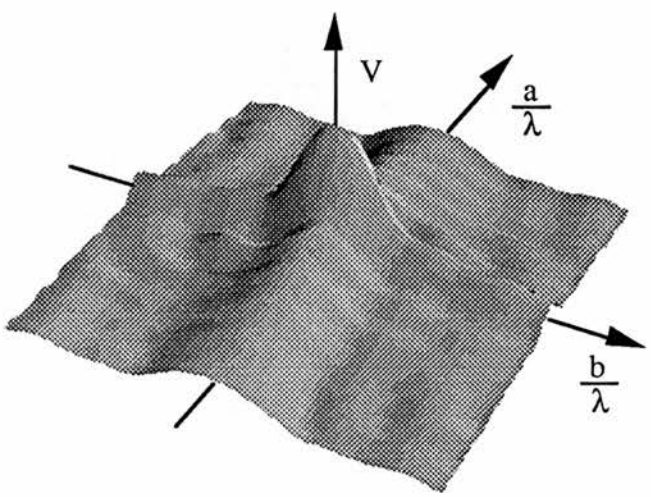
---

[1]To a good approximation, the results hold for any uniform circular light source, that is centred around the optical axis and has a small angular radius r. Here an angular radius r is small as long as

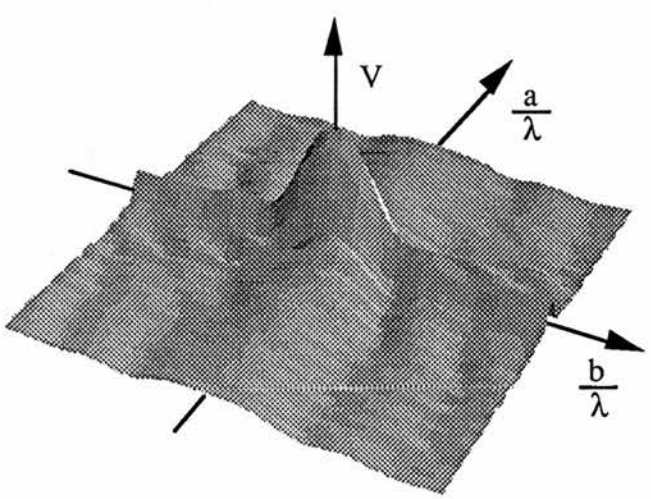
$$d\Omega(\alpha,\beta) \approx d\Omega(0°,0°) \quad \forall \alpha,\beta: \alpha^2+\beta^2 \le r^2.$$



A=0.0°, B=0°



A=0.4°, B=0°



A=0.8°, B=0°

Figure 19: The visibility of fringes due to a circular homogeneous light source with an angular radius of 1°, centred around (α, β)=(0°, 0°), as a function of the second derivatives a and b of a quadratic path difference relation. V ranges from 0 to 1, a/λ and b/λ range from -3.2/°^{2} to +3.2/°^{2}. The values above the a/λ and b/λ axes correspond to valley shaped pd relations. The next figure shows corresponding density plots.

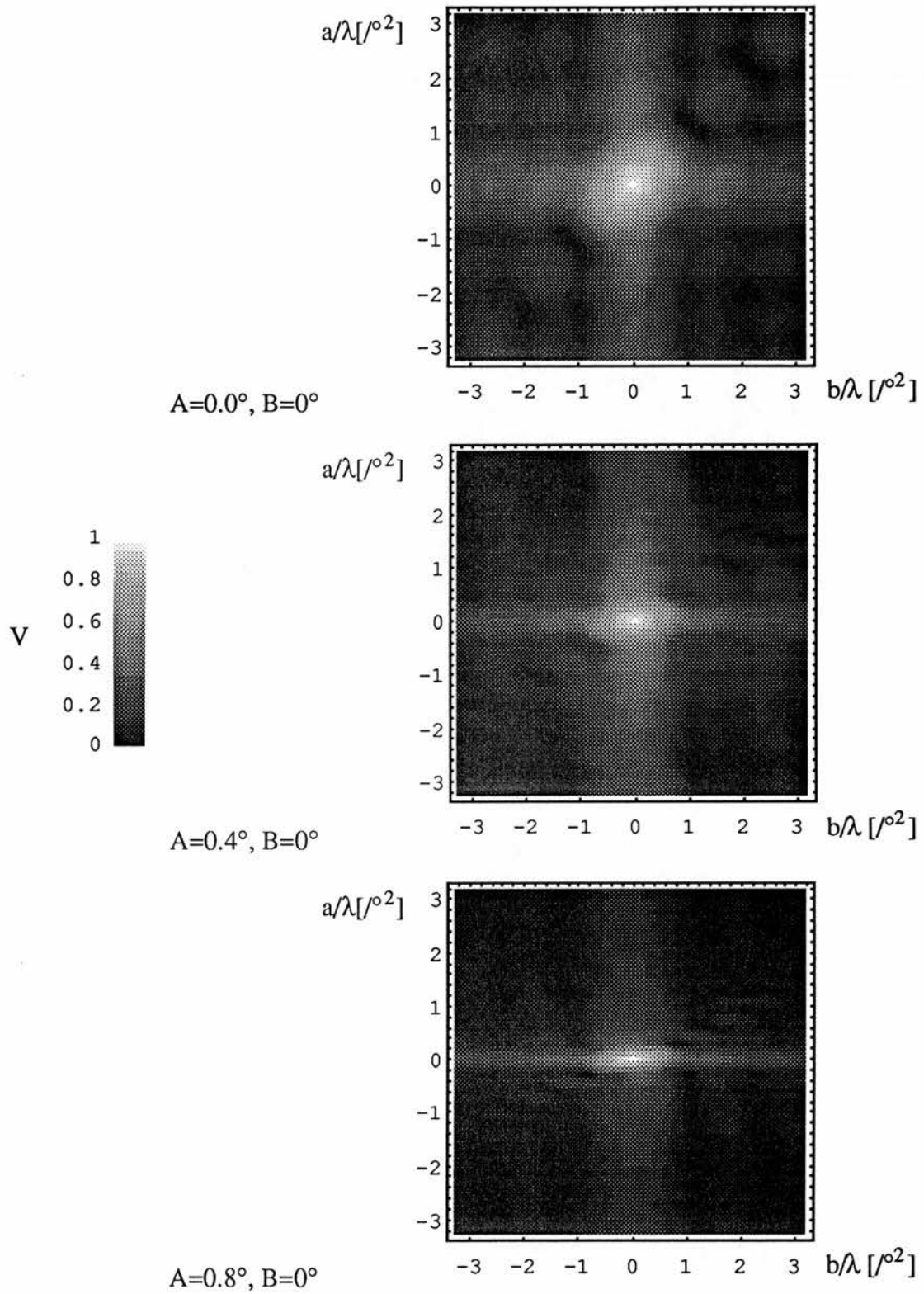


Figure 20: Density plots corresponding to the previous figure. In the plot for A=0° a prominent diagonal along the line a=b becomes visible, which corresponds to pd relations shaped like rotationally invariant paraboloids.

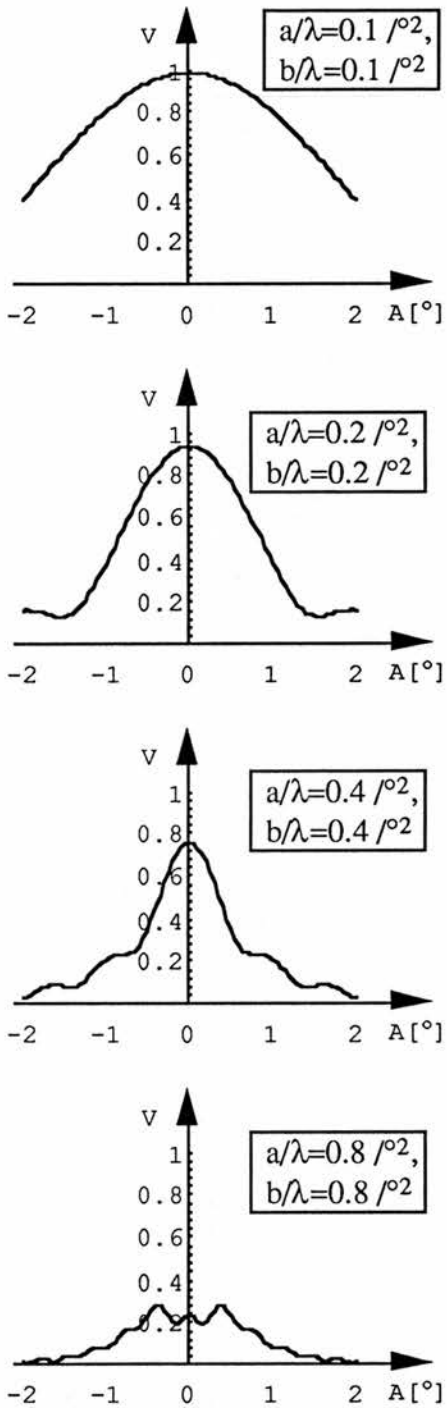
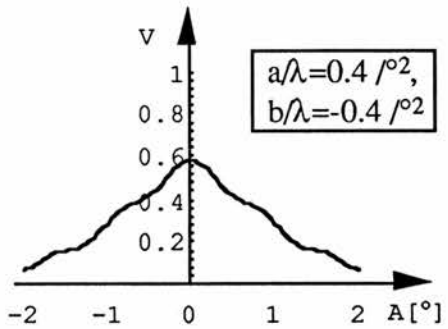
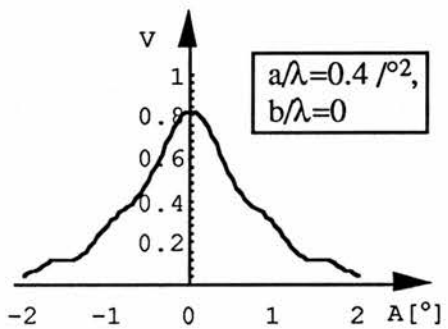


Figure 21: The visibility of fringes due to the same light source that was modelled in the previous figures, as a function of the angular displacement A, for B=0. As the second derivatives of the quadratic pd relation, a and b, get larger the visibility decreases more quickly with A. However, the behaviour of the visibility is not always that straightforward, especially the global maximum of V(A) is not necessarily located at A=0°.

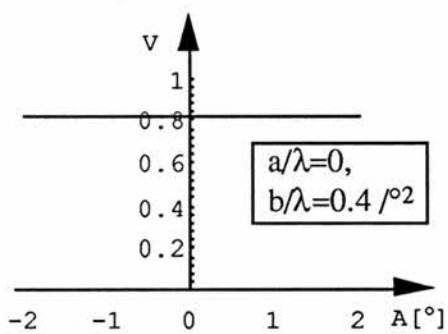
saddle



valley ⊥ displacement



valley ∥ displacement



paraboloid

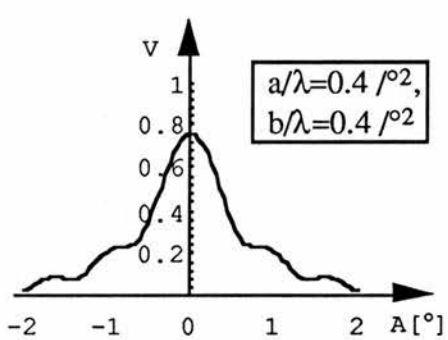


Figure 22: Comparison of the visibility for various shapes of the path difference relation (see fig. 7). As already obvious from fig. 18, it can be seen that a valley shaped path difference offers the highest visibility. The visibility corresponding to a rotational paraboloid sees the strongest variation with displacement A of the pd relation as well as with wavelength (see fig. 19 and 20; this case corresponds to the prominent diagonal in the figure for A=0°).

2.4. Fringe localisation within the Wollaston prism based spectrometer

As can be seen from the previous figures, for an extended light source the visibility of interference fringes depends on the position of the detector. If the extended light source is centred around input angles (α,β), the zones of highest fringe visibility are centred around the stationary positions $D_{stationary}(\alpha,\beta)$, which are defined in (2). The fringes are said to be localised there.

2.4.1. Ray separation at the input face and its relation to the angular dependent path difference

In Young's double slit experiment the (complex) degree of coherence of light at two points is closely related to the separation between the two points and indeed takes on its maximum value when the separation is zero. It is by no means obvious that fringes formed from light rays with no separation should always exhibit the highest visibility in any type of interferometer. However, in a double Wollaston prism spectrometer this seems to be precisely the case.

For the configurations studied in this work there is a relationship between the ray separation and the gradient of the angular dependent path difference relation (see fig. 23[1]), given by

$$s_D(\alpha,\beta) \propto \left|\mathrm{grad}\left(pd_D(\alpha,\beta)\right)\right| = \sqrt{\left(\left.\frac{\partial pd_D(\alpha',\beta)}{\partial\alpha'}\right|_{\alpha'=\alpha}\right)^2 + \left(\left.\frac{\partial pd_D(\alpha,\beta')}{\partial\beta'}\right|_{\beta'=\beta}\right)^2} \quad (7).$$

[1]The specifications of the spectrometer that was modelled are: λ=633nm, θ_1=15°, θ_2=31.95°, $t_1=t_2=d_W$=30mm, wedge material: calcite, orientation of optic axes: xyyx.

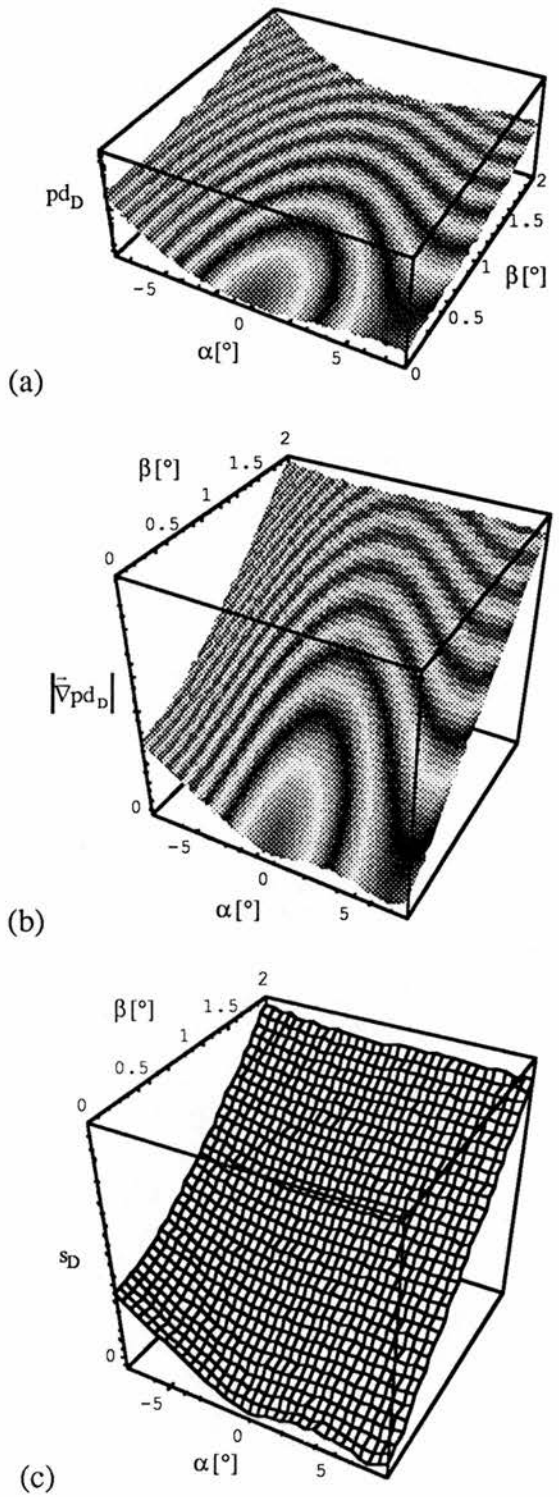


Figure 23: The path difference relation (a, with a corresponding phase pattern projected on it), the absolute value of its gradient (b, with the same projected pattern), and the ray separation (c), all as functions of the input angles α and β, for what is considered a particularly challenging Wollaston setup, i.e. a design with a path difference relation in the shape of a distorted valley. Note especially that s_D=0 at the same combination of input angles as grad(pd_D) = 0.

An important special case of (7) is that for stationary points $D_{stationary}(\alpha,\beta)$. Since the first derivatives of the corresponding path difference relations for these points are zero at input angles (α,β), (7) becomes for this case

$$s_{D_{stationary}(\alpha,\beta)}(\alpha,\beta) \equiv 0. \qquad (8)$$

2.4.2. The spatial distribution of stationary positions

Firstly, consider the following

lemma:

Consider a pair of light rays, that intersect in a point A′, propagating in a medium 1 with arbitrary directions. After being refracted at a plane interface $\mathfrak{I}$ into medium 2, they intersect in a point B′ (see fig. 24). Neither intersection need to be real.

Consider now another pair of light rays, which are in the same polarisation states and have the same directions as the first pair but intersect in different points A″ and B″.

Two lines, $\mathfrak{A}$ and $\mathfrak{B}$, can now be drawn through A′, A″ and B′, B″, respectively.

Then any pair of light rays in the same polarisation states and with the same directions as the first pair, that intersect in any point $A \in \mathfrak{A}$, will intersect again in a point $B \in \mathfrak{B}$. Moreover, the intersection S between $\mathfrak{A}$ and $\mathfrak{B}$ lies in $\mathfrak{I}$.

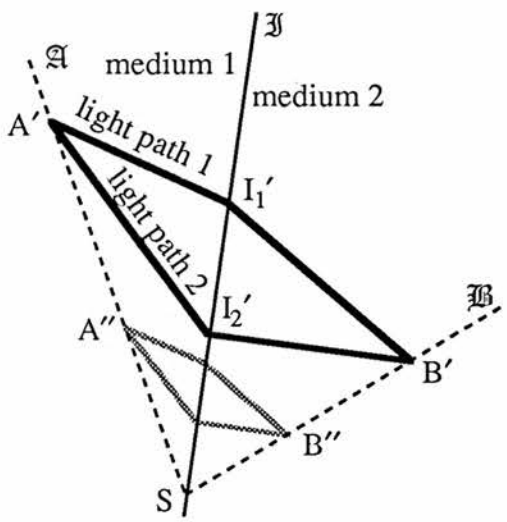


Figure 24: Geometry of the lemma.

The explanation lies in the similarity of the polygons $A'I_1'B'I_2'A'$, $A''I_1''B''I_2''A''$ and AI_1BI_2A.

This lemma also holds in three dimensions when 3 pairs rather than 2 pairs of rays, and when planes through 3 intersections rather than lines through 2 intersections are considered.

It can now be applied to derive the distribution of stationary points.

It was stated before that fringes produced by a far field light source, which is centred around input angles (α,β), are localised at the corresponding stationary points $D_{stationary}(\alpha,\beta)$. According to (8), fringes at the stationary points are formed by interference of light rays from input angles (α,β) which have zero separation when they enter the spectrometer. The two orthogonally polarised rays split at a planar surface defined by the Wollaston prism. The lemma implies that the ray intersections in the next medium will themselves form a plane that may be real or virtual.

The same argument can be applied to all interfaces of the system, which results in the fringes being localised to a plane.

Françon and Mallick [11, p.31], without proof, state the same result.

2.5. Field of view of the spectrometer (FOV)

The field of view determines the shape and size of the largest light source that does not significantly compromise the visibility of fringes.

Our definition of the field of view is the range of input angles over which the angular dependent path differences does not differ by more than $\lambda/2$ from the path difference for the on-axis ray.

Adopting this convention, the field of view for a detector position D can be defined as that range of input angles (α, β) for which

$$2\pi\frac{|pd_D(\alpha,\beta) - pd_D(0,0)|}{\lambda} \leq \pi.$$

2.5.1. Shape of the field of view

The localisation of fringes can be understood in terms of the field of view.

As stated earlier, a shift in Δz corresponds to a shift of the angular dependent path difference relation in α direction. The visibility of the resulting fringes depends on the degree of overlap between the field of view and the angular extent of the light source.

Figures 25 and 26 show possible shapes of the FOV for paraboloid and saddle shaped pd relations .

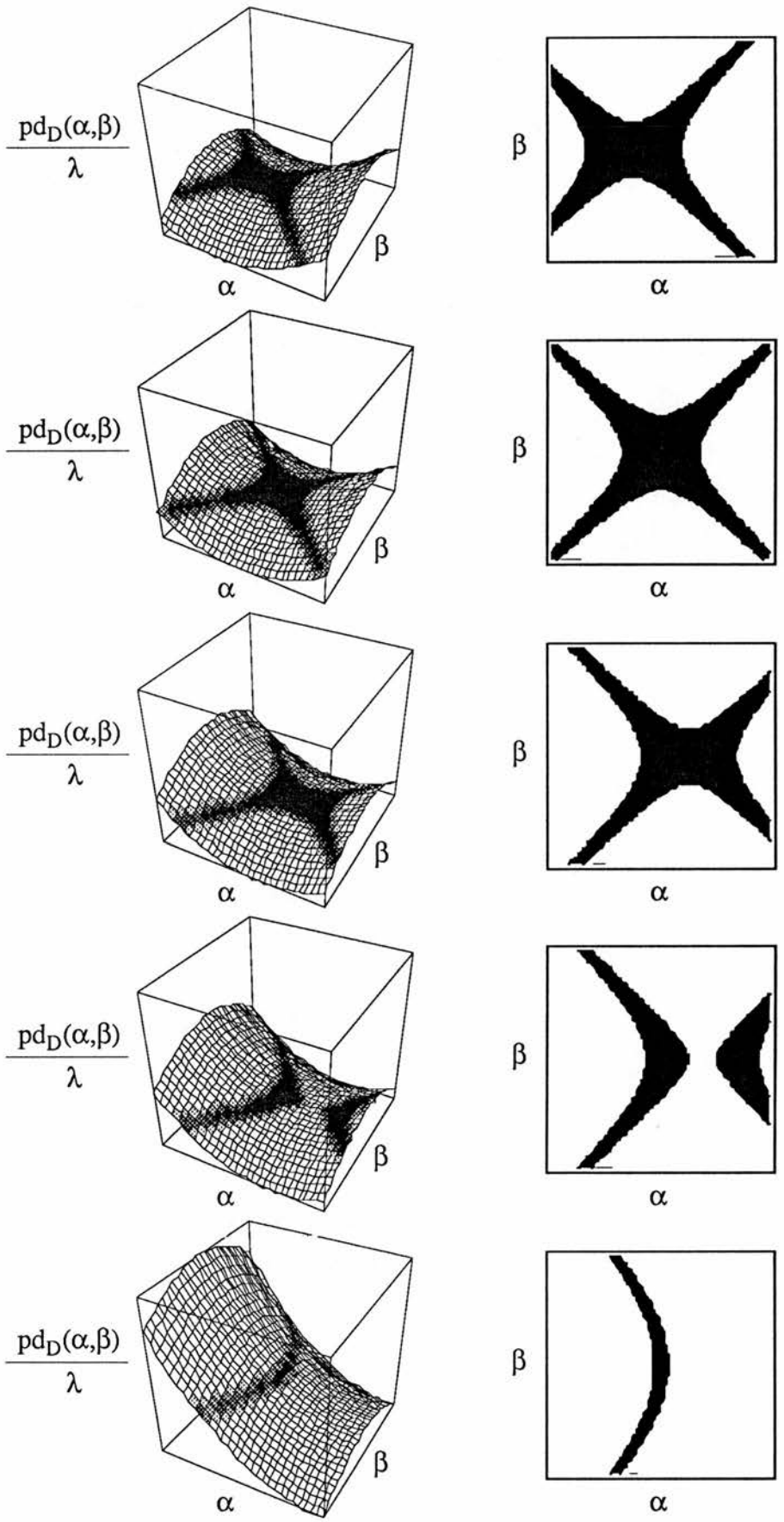


Figure 25: The shift of a saddle shaped path difference relation and the corresponding shape of the field of view.

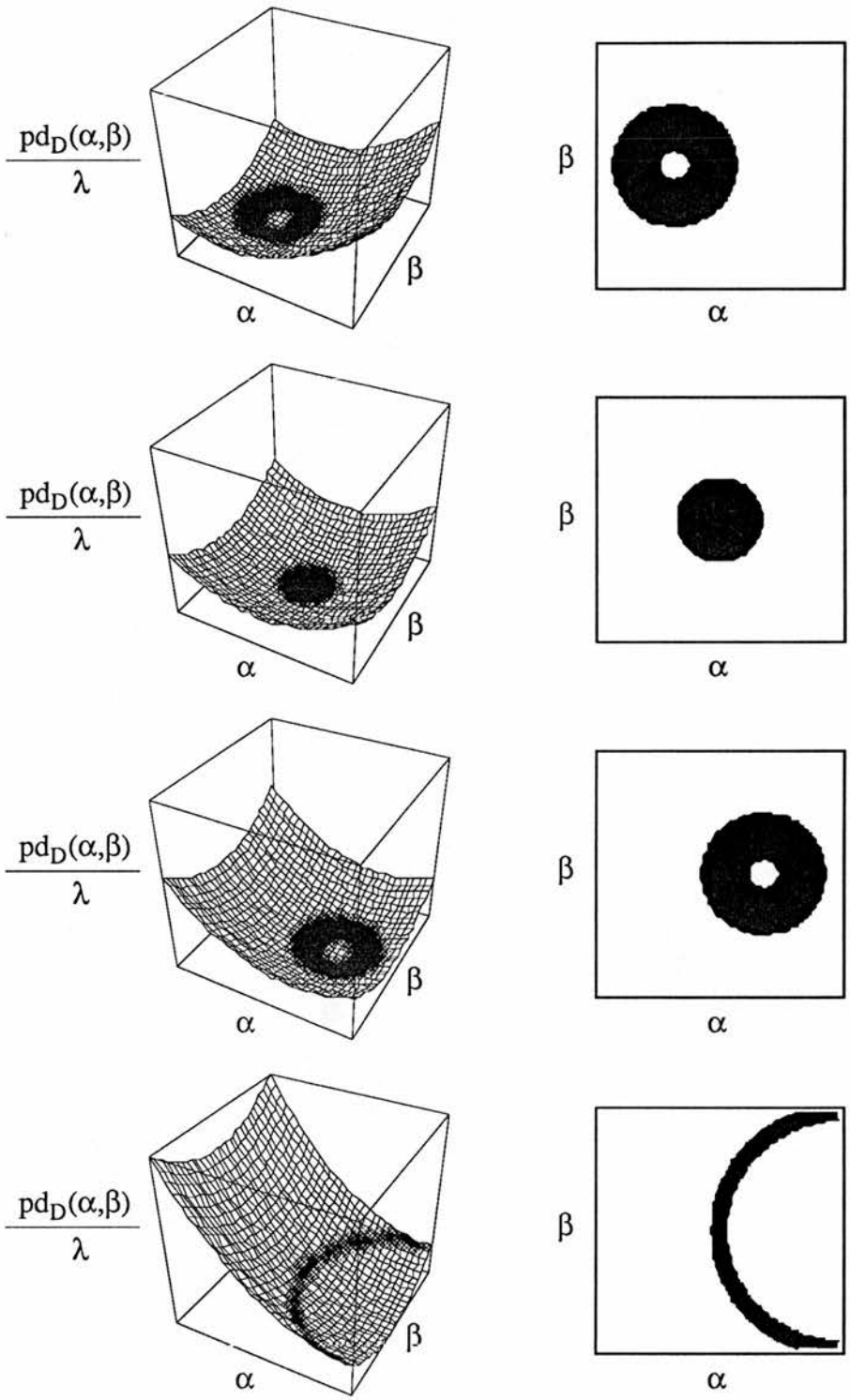


Figure 26: The shift of a paraboloid path difference relation and the corresponding shape of the field of view.

2.6. The path difference as a function of lateral position

The output of a Wollaston prism spectrometer is the intensity recorded for different stationary points along a line, which can be characterised by specifying the displacement along this line. The change of path difference between the rays as their intersection moves along this line is equivalent to the change of path difference within the Michelson interferometer as one of the mirrors is moved.

2.6.1. Detector positions D(l)

The spectrometer is usually configured so that the intensity distribution is sampled with a plane array detector, e.g. a CCD array. The detector is positioned so as to coincide with a plane $\mathfrak{D}$ that contains all stationary points $D_{stationary}(\alpha_0,\beta_0)$. For a light source that is centred around input angles (α_0,β_0) this normally ensures highest possible fringe visibility.

To calculate the signal recorded by the detector array one would consider the angular dependent path difference relations corresponding to points $D \in \mathfrak{D}$. In every design that is considered in this work, the intensity distribution is invariant to a shift in y-direction over the width of the prisms, i.e. the intensity at all positions D that differ only in their y coordinate is identical. Therefore only projections of points D into a plane y=const. need to be considered.

The projection of a point $D \in \mathfrak{D}$ can be characterised by its distance l along $\mathfrak{D}$ to the optical axis (see fig. 27).

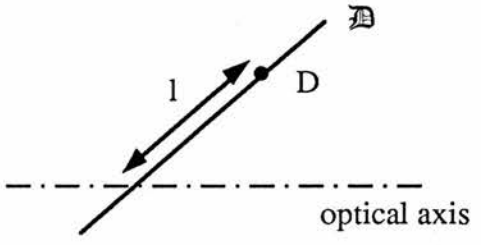


Figure 27: Projection of the optical axis, the plane $\mathfrak{D}$ and a point $D \in \mathfrak{D}$ into a plane y=const.

The projected detector position is called D(l).

2.6.2. Linearity of $pd_{D(l)}(\alpha,\beta)$

Similar to the discussion regarding the positions of stationary points $D_{stationary}(\alpha,\beta)$, consider a pair of light rays that propagate with arbitrary directions in medium 1. They intersect at a point A′ and, after being refracted at the plane interface $\mathfrak{I}$ between medium 1 and medium 2, intersect again at another point B′

(see fig. 24). Both intersections may be real, virtual or positioned at infinity (parallel rays).

According to the lemma discussed previously, if A′ is now moved within a plane $\mathfrak{A}$, B′ moves within a corresponding plane $\mathfrak{B}$ that intersects $\mathfrak{A}$ in a line $S \in \mathfrak{I}$.

Obviously, the lengths of the optical paths between A′ and B′ that are taken by the two light rays, scale with the distance between either A′ or B′ and S.

If more than one plane interface is involved, the path differences between consecutive intersections add up. The total optical path lengths are no longer proportional to the geometrical distances like $\overline{A'S}$ and $\overline{B'S}$, but still vary linearly with them.

The difference between the optical path lengths likewise varies linearly with geometrical distances like $\overline{A'S}$ and $\overline{B'S}$. If the last intersection lies at a detector position D(l) and the initial pair of light rays is parallel and impinging on the front face of the first Wollaston prism from input angles (α,β), the path difference between the two rays equals $pd_{D(l)}(\alpha,\beta)$.

2.6.3. The effective path difference $\Delta(\lambda,l)$ as a function of lateral position

As previously, the spectral intensity at a point D(l) is the superposition of the intensities resulting from light from input angles (α, β) (equation (3)) over the extent of the light source, i.e.

$$i_\lambda(D(l)) = \iint i_{\lambda,(\alpha,\beta)}(D(l))\,d\alpha d\beta$$

$$\propto \iint i_\lambda^{lamp}(\alpha,\beta)\left(1+\cos\frac{2\pi\cdot pd_{D(l)}(\alpha,\beta)}{\lambda}\right)d\alpha d\beta.$$

Using the definitions (4) and (5), this can be expressed as

$$i_\lambda(D(l)) \propto i_{\lambda,total}^{lamp} + Re\left(t_{interference}(D(l))\right).$$

Assuming a quadratic path difference relation

$$pd_{D(l)} = a(l)\alpha^2 + b(l)\beta^2 + c(l),$$

the interference term in the spectral intensity at D(l) becomes

$$t_{interference}(D(l)) = e^{i\frac{2\pi c(l)}{\lambda}}\,FT\left\{k_\lambda^{lamp}(\alpha,\beta)\right\}(2\pi a(l)/\lambda, 2\pi b(l)/\lambda),$$

where $k_\lambda^{lamp}(\alpha,\beta)$ is defined as in (6). $t_{interference}(D(l))$ is a complex number, whose argument is the phase $\phi(D(l))$ of the corresponding fringe pattern.

Further analysis is required to establish the exact behaviour of $\phi(D(l))$. In the following section it is assumed that ϕ varies linearly with l. This is equivalent to the existence of an effective path difference $\Delta(\lambda,l)$ that is a linear function of l. This assumption is supported by experimental observations.

2.7. Interferogram signal recorded by the camera

The intensity distribution I(l) in the plane of the camera gives rise to an energy

$$E_n = \Delta t \cdot h \int_{-\infty}^{\infty} I(l) p_n(l) dl$$

falling on pixel element n of height h during an integration time Δt. $p_n(x)$ is called *individual pixel function of the nth element* or its *aperture response function*, and is defined as

$$p_n(l) \equiv \frac{\frac{s_n}{E(l)}}{\max\left(\frac{s_n}{E(l)}\right)},$$

where s_n is the signal from the nth pixel when subject to a point illumination at coordinate l. If the pixel elements only differ in their displacement in l direction, the individual pixel functions can be written in terms of the *(global) pixel function*

$$p(l) \equiv p_n(l_n - l),$$

where l_n is the l coordinate of the centre of the nth pixel. Thus E_n becomes

$$E_n \propto \int_{-\infty}^{\infty} I(l) p(l_n - l) dl \equiv (I \otimes p)(l_n),$$

i.e. the convolution of I(l) with p(l), evaluated at l_n.

Rewriting this equation in terms of the energy due to light in the wavelength range $[\lambda, \lambda+d\lambda]$ falling on the nth pixel, $e_n(\lambda)d\lambda$, and the lateral distribution of the intensity in this wavelength range, $i(\lambda,l)d\lambda \equiv i_\lambda(\lambda)d\lambda$, gives

$$e_n(\lambda) \propto (i_\lambda \otimes p)(l_n).$$

Allowing for the spectral response of the camera, the signal S_n from the nth camera pixel is

$$S_n = \int_0^{\infty} \sigma(\lambda) e_n(\lambda) d\lambda,$$

where $\sigma(\lambda)$ is called *spectral responsitivity* of the pixel element. Substituting the expression for the $e_n(\lambda)$, it becomes

$$S_n \propto \int_0^{\infty} \sigma(\lambda)(i_\lambda \otimes p)(l_n) d\lambda.$$

In the case of an interferometer that introduces a path difference[1] $\Delta(\lambda,l)$ between two rays of wavelength λ whose paths intersect at a lateral position l, producing fringes of visibility $V(\lambda)$, the spectral intensity distribution at a displacement l is

$$i_\lambda(l) \propto i(\lambda) \cdot \left(1 + V(\lambda) \cos \frac{2\pi \cdot \Delta(\lambda,l)}{\lambda}\right).$$

---

[1] the introduction of an effective path difference might be necessary; see previous section

This equation is a modified version of (3) and takes into account the visibility $V(\lambda)$ of the fringes as a function of wavelength.

The recorded signal S_n has a constant term C, which is the same for each pixel and represents an offset to the signal, and a variable component F_n, i.e.

$$S_n \propto C + F_n$$

with

$$C := \int_0^\infty \sigma(\lambda) i(\lambda) d\lambda \cdot \int_{-\infty}^\infty p(l) dl$$

and

$$F_n = F(l_n)$$

where

$$F(l) = \int_0^\infty V(\lambda)\sigma(\lambda)i(\lambda) \cdot \left(\int_{-\infty}^\infty \cos \frac{2\pi \cdot \Delta(\lambda, l')}{\lambda} \cdot p(l - l') dl' \right) d\lambda.$$

Changing the order of the integration gives

$$F(l) = \int_{-\infty}^\infty \left(\int_0^\infty V(\lambda)\sigma(\lambda)i(\lambda) \cos\left(2\pi \frac{\Delta(\lambda, l')}{\lambda} \right) d\lambda \right) \cdot p(l - l') dl'.$$

If the path difference relation can be written as

$$\Delta(\lambda, l) = \Delta_\lambda(\lambda) \cdot l,$$

F(l) becomes

$$F(l) = \int_{-\infty}^\infty \left(\int_0^\infty V(\lambda)\sigma(\lambda)i(\lambda) \cos\left(2\pi \frac{\Delta_\lambda(\lambda)}{\lambda} l' \right) d\lambda \right) \cdot p(l - l') dl'.$$

It is convenient to perform the following calculation in terms of a new variable

$$\phi(\lambda) \equiv \frac{\Delta_\lambda(\lambda)}{\lambda} \qquad (9).$$

The integration over the wavelength range can be performed over $\phi(\lambda)$. Then $d\lambda$ has to be substituted by

$$d\lambda = \frac{d\phi}{\phi'(\lambda)},$$

the limits of the integration can be calculated on the assumption that

$$0 < \Delta_\lambda(\lambda) < \infty \quad \forall \lambda \geq 0,$$

whereupon they become

$$\phi(0) = \infty$$

and

$$\phi(\infty) = 0.$$

The expression for the variable part of the interferogram as recorded by the camera becomes

$$F(l)=\int_{-\infty}^{\infty}\left(\int_{0}^{\infty}\frac{V(\lambda(\phi))\sigma(\lambda(\phi))i(\lambda(\phi))}{\phi'(\lambda(\phi))}\cos(2\pi\cdot\phi\cdot l')d\phi\right)\cdot p(l-l')dl',$$

which, with the definition

$$F_{\phi}(\phi)\equiv\frac{V(\lambda(|\phi|))\sigma(\lambda(|\phi|))i(\lambda(|\phi|))}{\phi'(\lambda(|\phi|))},\quad \phi\in\Re,$$

and by writing

$$FCT\{f\}(\nu)\equiv\int_{0}^{\infty}f(t)\cdot\cos(2\pi\cdot t\cdot\nu)dt$$

for the Fourier cosine transform of the function f(t), yields

$$F(l)=\int_{-\infty}^{\infty}FCT\{F_{\phi}\}(l')\cdot p(l-l')dl'=$$

$$=\left(FCT\{F_{\phi}\}\otimes p\right)(l).$$

It is useful to write this in terms of the Fourier transform

$$FT\{f\}(\nu)\equiv FT\{f(t)\}(\nu)\equiv\int_{-\infty}^{\infty}f(t)\cdot e^{2\pi i\cdot\nu\cdot t}dt.$$

The corresponding inverse transformation is defined as

$$FT^{-1}\{\tilde{f}\}(t)\equiv FT^{-1}\{\tilde{f}(\nu)\}(t)\equiv\int_{-\infty}^{\infty}\tilde{f}(\nu)\cdot e^{-2\pi i\cdot\nu\cdot t}d\nu.$$

For an even and real function f(t)

$$2\cdot FCT\{f\}=FT\{f\}=FT^{-1}\{f\},\qquad(10)$$

and so F(l) becomes

$$F(l)=\frac{1}{2}\left(FT\{F_{\phi}\}\otimes p\right)(l).$$

Application of the convolution theorem

$$FT\{f\otimes g\}=FT\{f\}\cdot FT\{g\}$$

yields

$$FT\{F\}=\frac{1}{2}FT\{FT\{F_{\phi}\}\}FT\{p\}.$$

As can be seen from the definition, $F_{\phi}(\phi)$ is an even and real function. Consequently, according to (10), its Fourier transform and its inverse Fourier transform are identical. Therefore

$$FT\{FT\{F_{\phi}\}\}=FT\{FT^{-1}\{F_{\phi}\}\}=F_{\phi},$$

so that

$$FT\{F\}=\frac{1}{2}F_{\phi}\cdot FT\{p\}.$$

It can be seen from the definition of F(l), that if p(l) and $F_{\phi}(\phi)$ are real, so is F(l). Therefore, as used above, the Fourier transform of the last equation is

$$F = \frac{1}{2}FT\left\{F_\phi \cdot FT\{p\}\right\}.$$

Substituting the definition of $F_\phi(\phi)$ into the last equation and the result into the definition of S_n yields

$$S_n \propto \int_0^\infty \sigma(\lambda) i(\lambda) d\lambda \cdot \int_{-\infty}^{\infty} p(l) dl + \\ + \frac{1}{2} FT\left\{ \frac{V\left(\lambda(|\phi|)\right) \cdot \sigma\left(\lambda(|\phi|)\right) \cdot FT\{p\}(\phi)}{\phi'\left(\lambda(|\phi|)\right)} \cdot i\left(\lambda(|\phi|)\right) \right\}(l_n) \qquad (11).$$

This is the result for the recorded interferogram. It takes account of the camera characteristics and optical limitations of the design in the form of a wavelength-dependent visibility.

2.8. Calculation of the spectral intensity distribution of the input light from the camera signal

Equation (10) is now inverted to extract the spectral distribution $i(\lambda)$ of the incoming light from the recorded interferogram, S_n.

The data set, $\{S_n: n \in \{0,1,...,N-1\}\}$, can be considered to be a discretely sampled form of a signal function

$$S(x): S(x_n) = S_n, \; S(x) \propto C + F(x).$$

The offset S_{offset} of the signal function $S(x)$ is the sum of C and the zero frequency component of the argument of the Fourier transform,

$$F_\phi(0) \cdot FT\{p\}(0) = \frac{V(\lambda(0)) \cdot \sigma(\lambda(0)) \cdot FT\{p\}(0)}{\phi'(\lambda(0))} \cdot i(\lambda(0)). \qquad (12)$$

This offset is proportional to the total light intensity normalised with respect to the spectral response of the detector. Within this work this term can be ignored since it is solely the spectral distribution of the light which is of interest. S is even and real, therefore its inverse Fourier transform is the same as its Fourier transform, so

$$\frac{V\left(\lambda(|\phi|)\right) \cdot \sigma\left(\lambda(|\phi|)\right) \cdot FT\{p\}(\phi)}{\phi'\left(\lambda(|\phi|)\right)} \cdot i\left(\lambda(|\phi|)\right) \propto FT^{-1}\{S\}(\phi) = FT\{S\}(\phi)$$

for $\phi \neq 0$.

If

$$x_n := x_0 + n\delta, \; n \in \{0,1,...,N-1\},$$

i.e. the N camera pixels are equidistant,

$$S_{x_0}(x') \equiv S(x' + x_0)$$

defines a discrete function that has values

$$S_{x_0}(x'_n) = S_n$$

at points

$$x'_n := n\delta, \; n \in \{0,1,...,N-1\}.$$

The Fourier transform of $S(x'+x_0)$ can be approximated as the discrete Fourier transform of the discrete function S_n, i.e.

$$\left(\mathrm{DFT}\{(S_n)\}\right)_m \equiv \sum_{n=0}^{N-1} S_n \cdot e^{2\pi i \cdot \frac{nm}{N}} \approx \frac{1}{\delta} \mathrm{FT}\left\{S_{x_0}\right\}(\phi_m).$$

The mth element of the discrete Fourier transform of S_n corresponds to an argument

$$\phi_m \equiv \frac{m}{N\delta}, \quad m \in \{0,1,\ldots,N-1\}$$

of the Fourier transform of $S(x'+x_0)$.

Often one does not know the exact value of x_0, i.e. the exact displacement of the camera field, but the shifting property of the Fourier transform,

$$\mathrm{FT}\left\{S_{x_0}\right\}(\phi) = e^{-2\pi i \cdot \phi \cdot x_0} \mathrm{FT}\{S\}(\phi)$$

together with the fact that $\mathrm{FT}\left\{S_{x_0}\right\}(\phi)$ is purely real, yields

$$\left|\mathrm{FT}\{S\}(\phi)\right| = \left|\mathrm{FT}\left\{S_{x_0}\right\}(\phi)\right|.$$

Solving (12) for $i(|\lambda(\phi)|)$, taking absolute values on both sides, and substituting the last results yields the good approximation

$$i(\lambda(\phi_m)) \approx \frac{\left|\left(\mathrm{DFT}\{(S_n)\}\right)_m\right| \cdot \left|\phi'(\lambda(\phi_m))\right|}{\left|\mathrm{FT}\{p\}(\phi_m)\right| \cdot V(\lambda(\phi_m)) \cdot \sigma(\lambda(\phi_m))} \qquad (13).$$

This is the key result of this section and is used within the software developed to control the UV spectrometer described in the next chapter. It is also the basis of the modelling of the spectrometer response in the next chapter.

3. Detection of hydrogen sulphide

3.1. Theoretical modelling of observed interferogram and inferred spectrum

The operational characteristics of a Fourier-transform spectrometer and factors governing its performance are not obvious. The computer model described in this section was written using the application *Mathcad* [14]. It takes the real spectrum obtained from passing a well-defined light source through a particular gas and calculates the perfect interferogram produced by the twin Wollaston prisms and polarisers. The model proceeds to calculate the interferogram as recorded by the detector array, allowing for such factors as spectral sensitivity and aperture response function of the pixels of the detector array, as described by equation (11). Having calculated the recorded interferogram, the measured spectrum can be inferred, using equation (13), and compared to the original spectrum. Using this model, the observable spectrum arising from any particular gas can be calculated and the influence of camera specifications assessed.

The figures shown below relate to the computer modelling of the spectrometer performance.

3.1.1. Input data for gas spectra

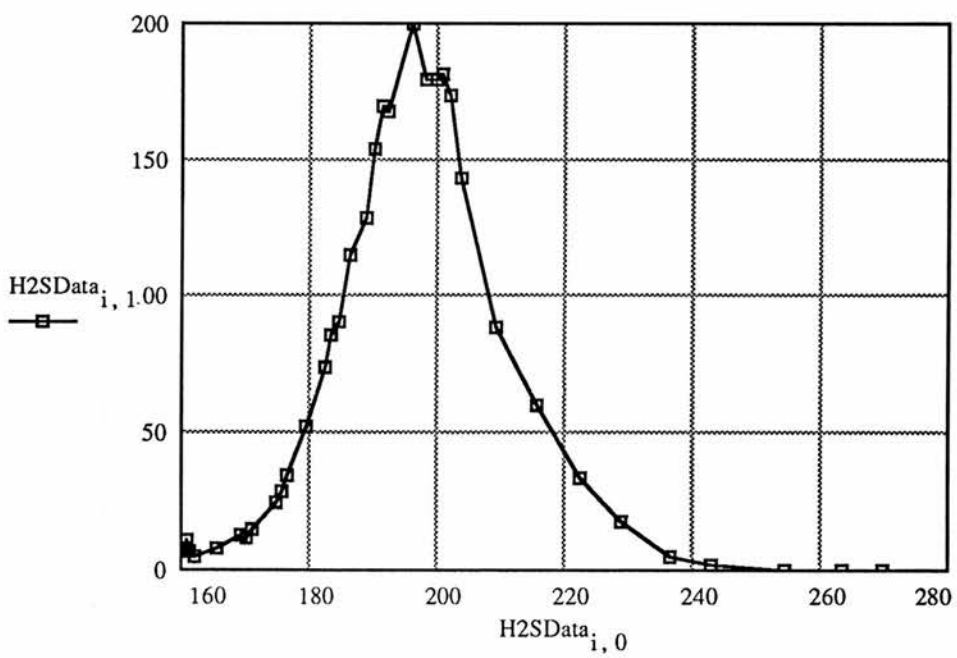


Figure 28: Input data for the UV absorption coefficient of hydrogen sulphide as taken from the literature [15; 19].

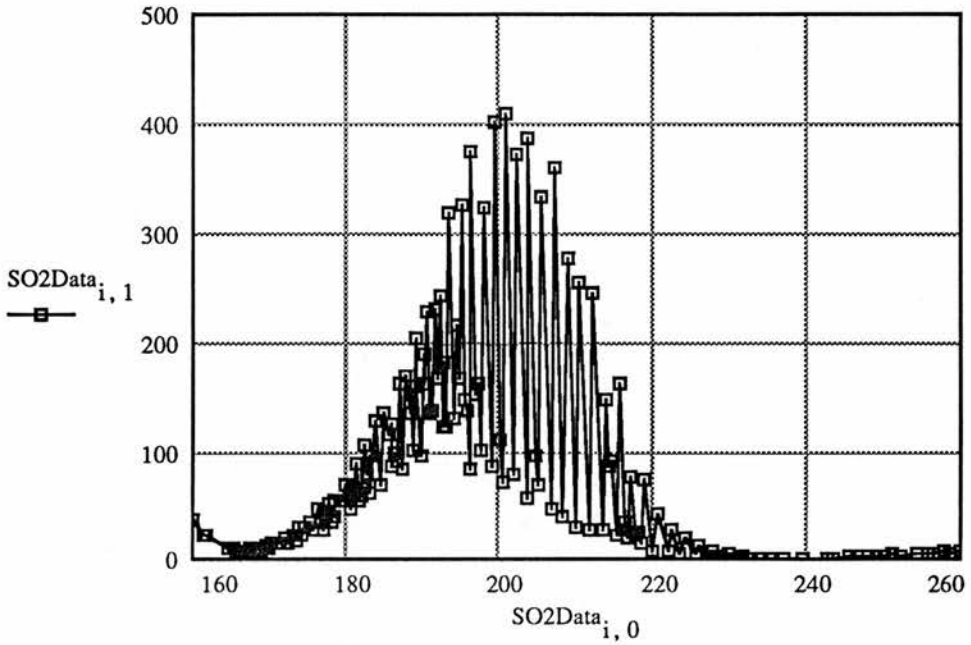


Figure 29: Input data for the UV absorption coefficient of sulphur dioxide.

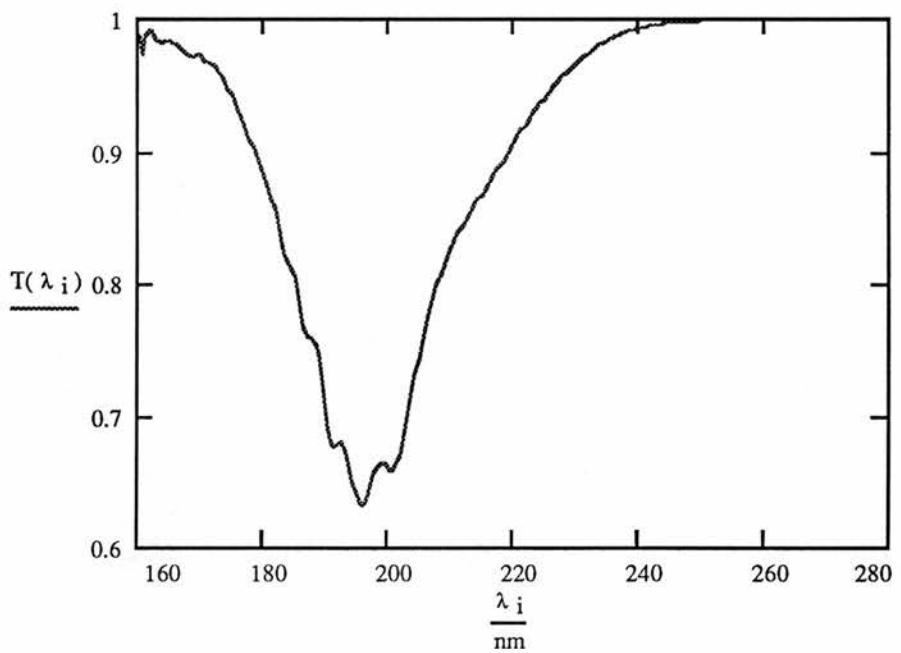


Figure 30: Transmission coefficient of a 100mm long cell filled to 0.0001atm of hydrogen sulphide.

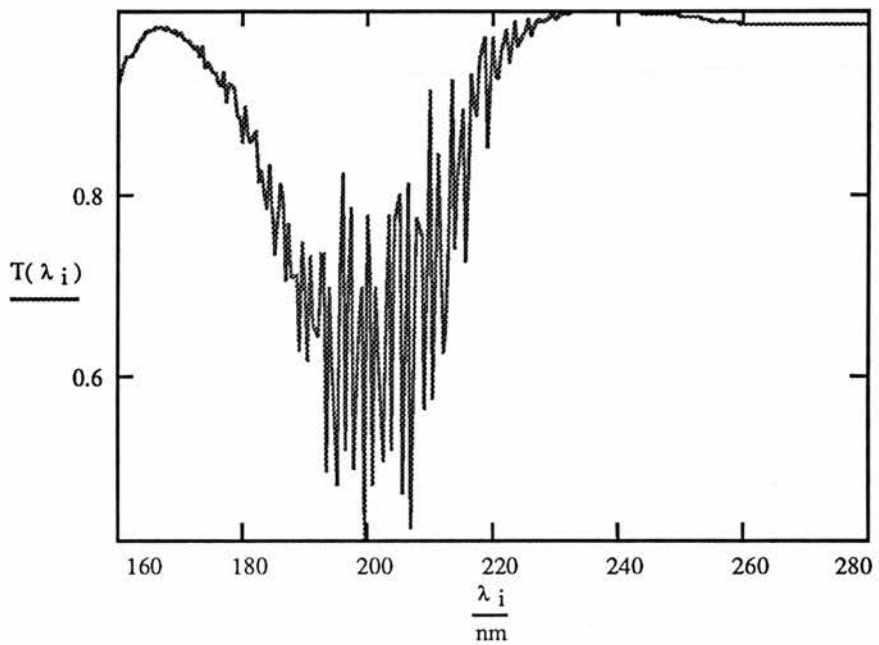


Figure 31: Transmission coefficient of a 100mm long cell filled to 0.0001atm of sulphur dioxide.

3.1.2. Input data for lamp, filter and camera response

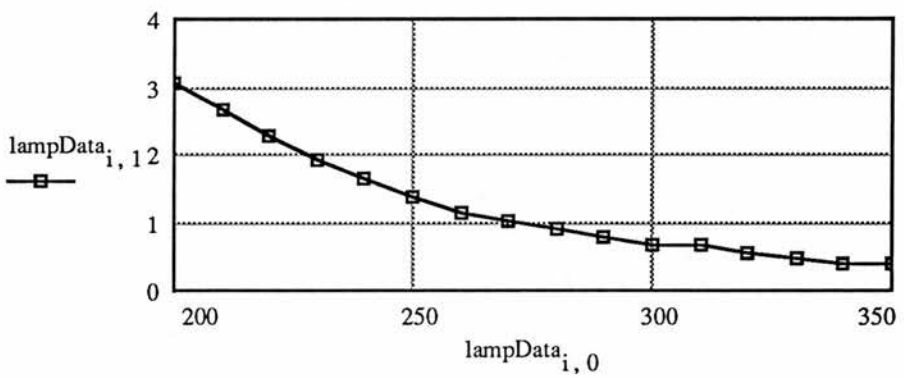


Figure 32: Input data for the spectral intensity distribution of light from the deuterium lamp.

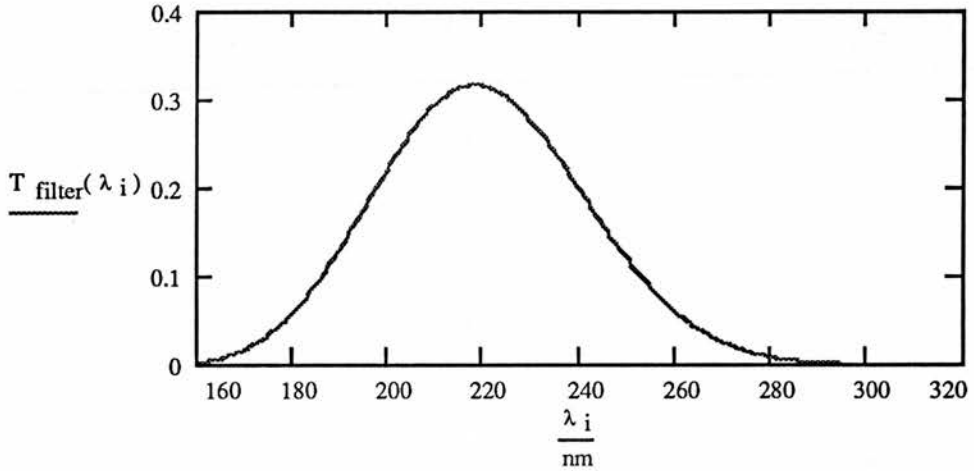


Figure 33: Input data for the transmission of the 220nm band-filter.

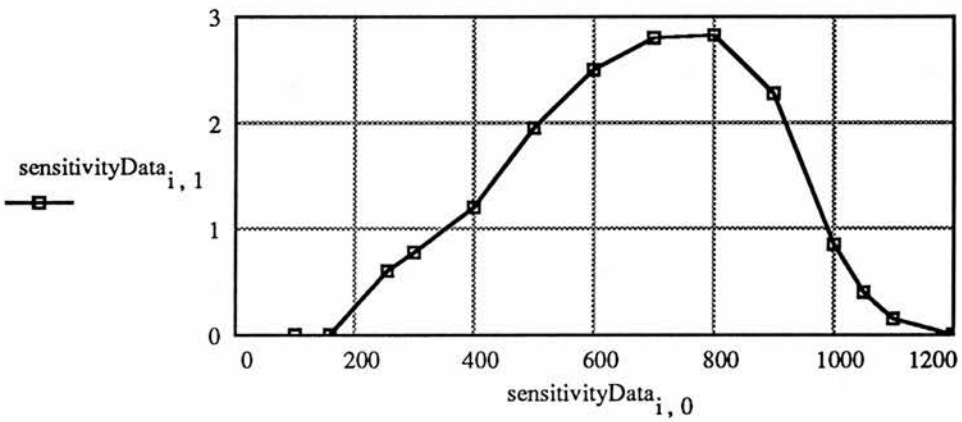


Figure 34: Input data for the spectral sensitivity of the Cronin camera.

3.1.3. Camera readout as simulated by the computer model

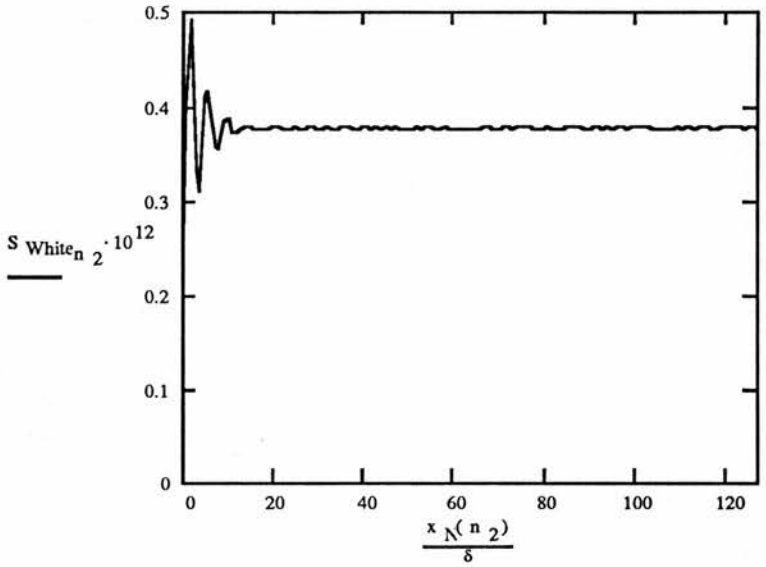


Figure 35: Modelled interferogram, prior to absorption by the gas, allowing for camera characteristics and detector noise, s/n=1000.

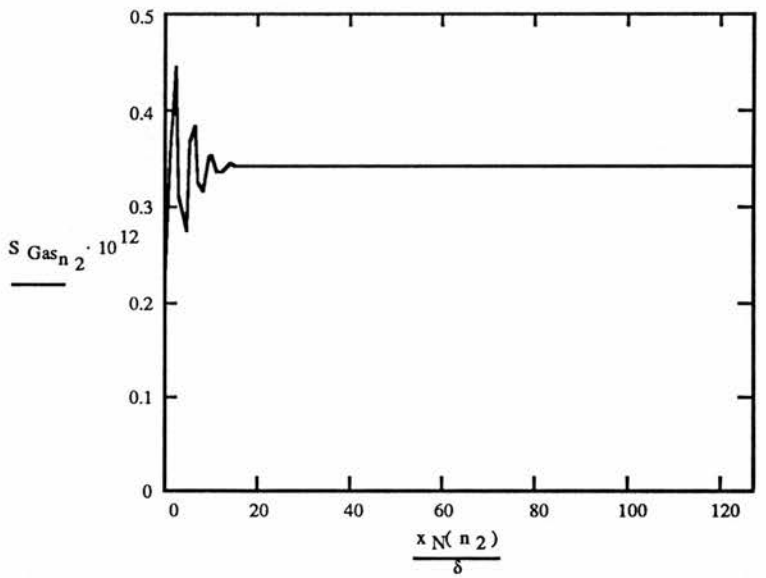


Figure 36: Modelled interferogram, after absorption by the hydrogen sulphide, allowing for camera characteristics and detector noise, s/n=1000.

3.1.4. Inferred spectral light distributions as simulated by the computer model

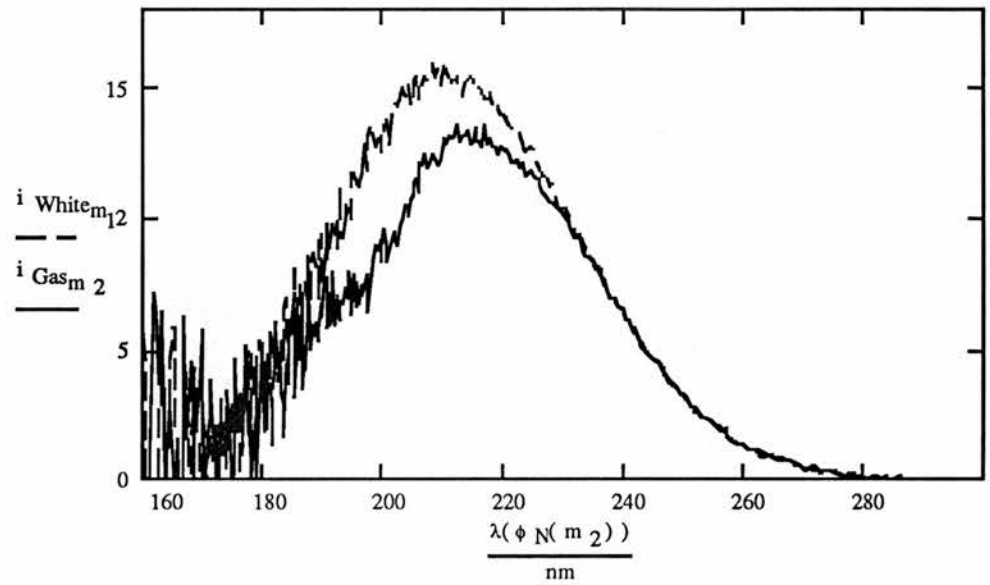


Figure 38: Spectral distribution of the light before, i_{White}, and after, i_{Gas}, transmission through the hydrogen sulphide cell, as inferred from the modelled interferograms.

3.1.5. Transmission spectrum of hydrogen sulphide as inferred from the modelled camera readout

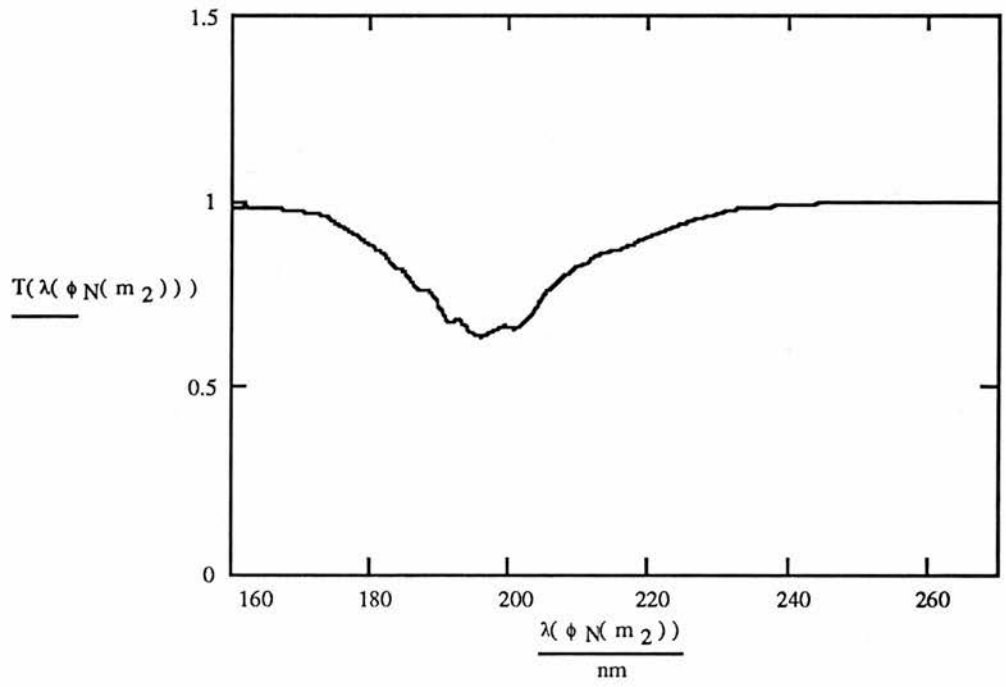


Figure 39: Actual transmission coefficient of the hydrogen sulphide cell.

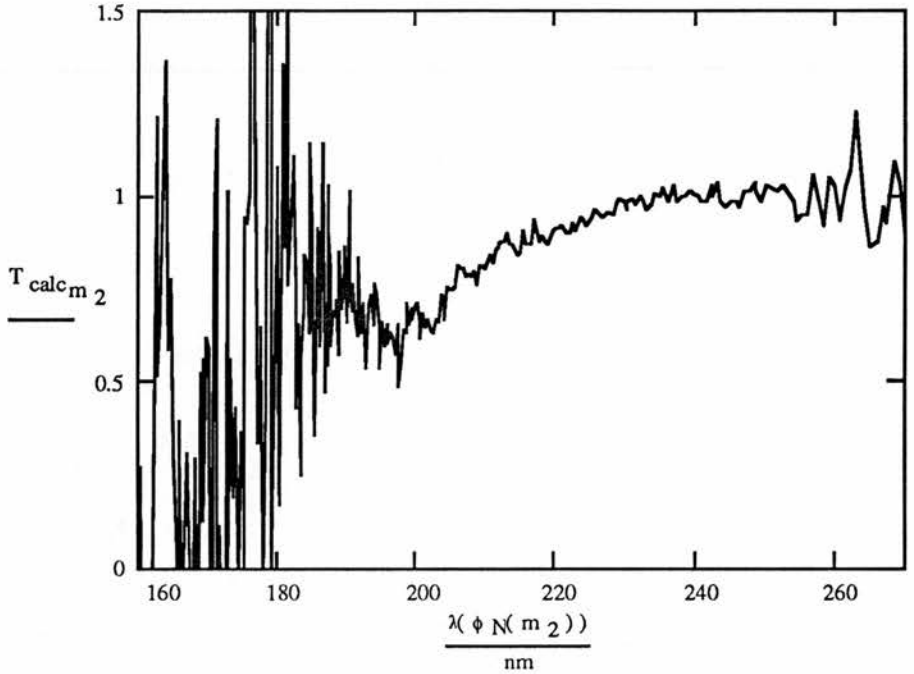


Figure 40: Measured transmission coefficient of the hydrogen sulphide cell as simulated by the computer model, allowing for both camera characteristics and detector noise. Note the increased noise level in regions of low light output either side of the filter peak.

3.1.6. Transmission spectrum of sulphur dioxide as inferred from the modelled camera readout

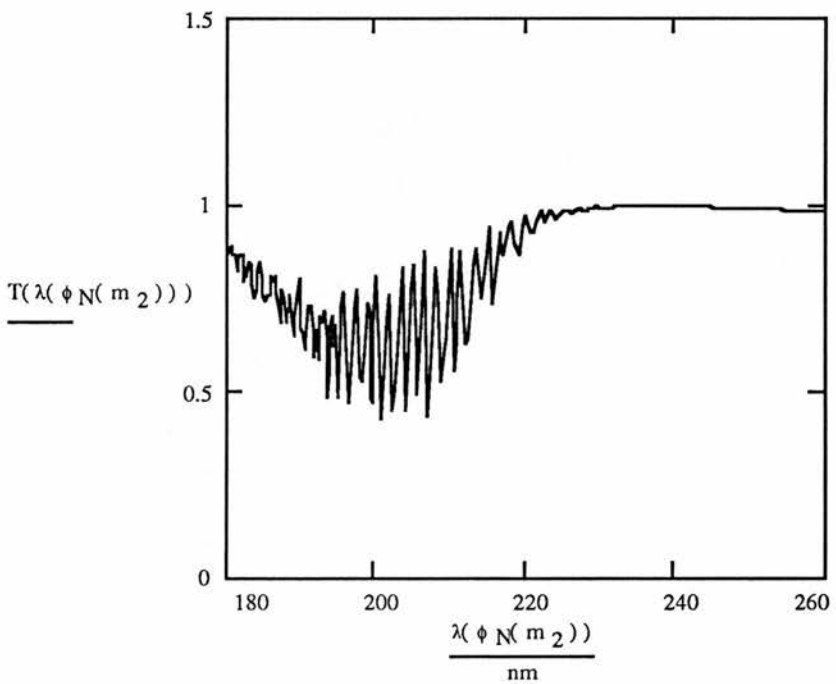


Figure 41: Actual transmission coefficient of the sulphur dioxide cell.

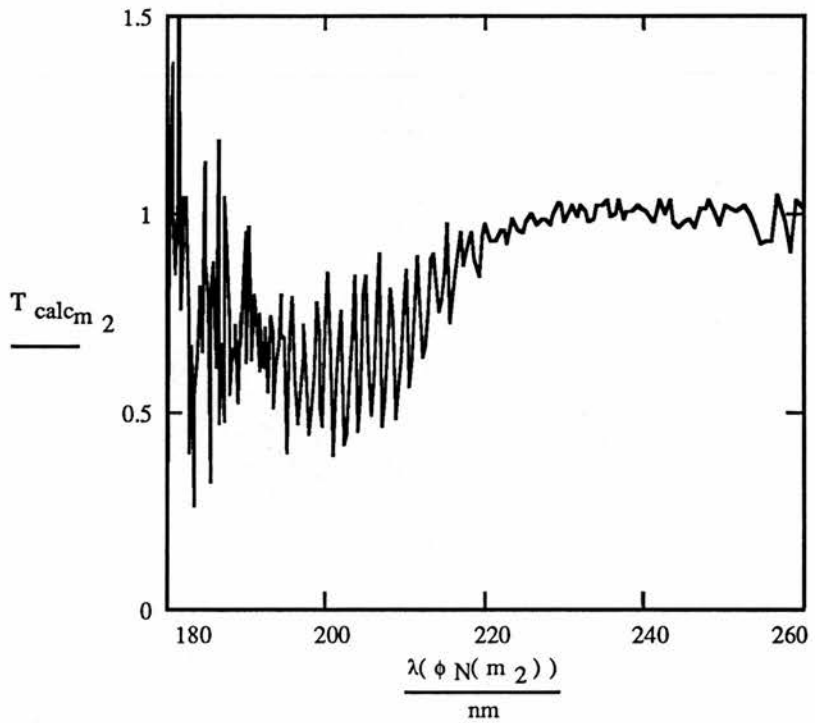


Figure 42: Transmission coefficient of the sulphur dioxide cell as inferred from the modelled camera readout, allowing for both camera characteristics and detector noise.

3.2. Choice of operating wavelength and resolution of the spectrometer

3.2.1. Absorption bands of interest

Hydrogen sulphide (H_2S) has absorptions in both the near infra-red and ultra-violet region of the spectrum.

The main absorption bands of interest are:

(a) E-series and A series lines between 120 nm and 160 nm with a peak absorption coefficients of about 200 $atm^{-1}cm^{-1}$ [15];

(b) dissociation continuum between 160 nm and 270 nm peaking at about 200 atm^{-1} cm^{-1} [19]. Other authors give 330 atm^{-1} cm^{-1} at 196 nm) [15];

(c) rotational spectrum (~20 lines) between 1.29 μm and 1.33 μm [19] with peak absorption of 1.26 x 10^{-3} $atm^{-1}cm^{-1}$;

(d) rotational spectrum (~25 lines) between 1.56 μm and 1.6 μm with a peak absorption coefficient of 3.9 x 10^{-3} $atm^{-1}cm^{-1}$;

(e) three absorption bands in the fundamental region; rotational transitions at 7-9 μm, its overtone 4.25 μm and one at 3.84 μm [16; 17]. There is an additional continuum absorption close to 2.6 μm with absorption coefficients of $0.01 \cdot atm^{-1}cm^{-1}$ [18].

Previously work has been reported using two tone frequency modulation spectroscopy (TTFMS) for the detection of hydrogen sulphide at ≈1.6μm [19]. Although capable of detecting hydrogen sulphide at low concentrations, this technique is felt to be too expensive for manufacture.

In relation to the static Fourier-transform instrument operation was considered in both the ultra-violet and infra-red regions of the spectrum. The advantage/disadvantages of hydrogen sulphide detection in the two regions are summarised in the table below.

spectral range	advantages	disadvantages
ultra-violet (160-270mnm)	large absorption coefficient (10^5 greater than IR); other gases can be detected with same instrument (e.g. SO_2)	potential UV blocking by oil film; potential problems with source lifetime; oxygen block below 190nm
infra red (1.3-1.6μm)	clearly resolvable spectral lines	very weak absorption; needs infra-red detector array

3.2.2. Operating wavelength range of the spectrometer

Given the 10^5 higher absorption strength, it was decided that the ultra-violet spectral region presented the obvious choice for a demonstration instrument. The short wavelength cut-off for useful atmospheric measurements is effectively imposed by the oxygen absorptions at 190nm. Given the absorption profile of hydrogen sulphide (see fig. 43) [15; 19], the target operating range for the spectrometer is 190-270nm.

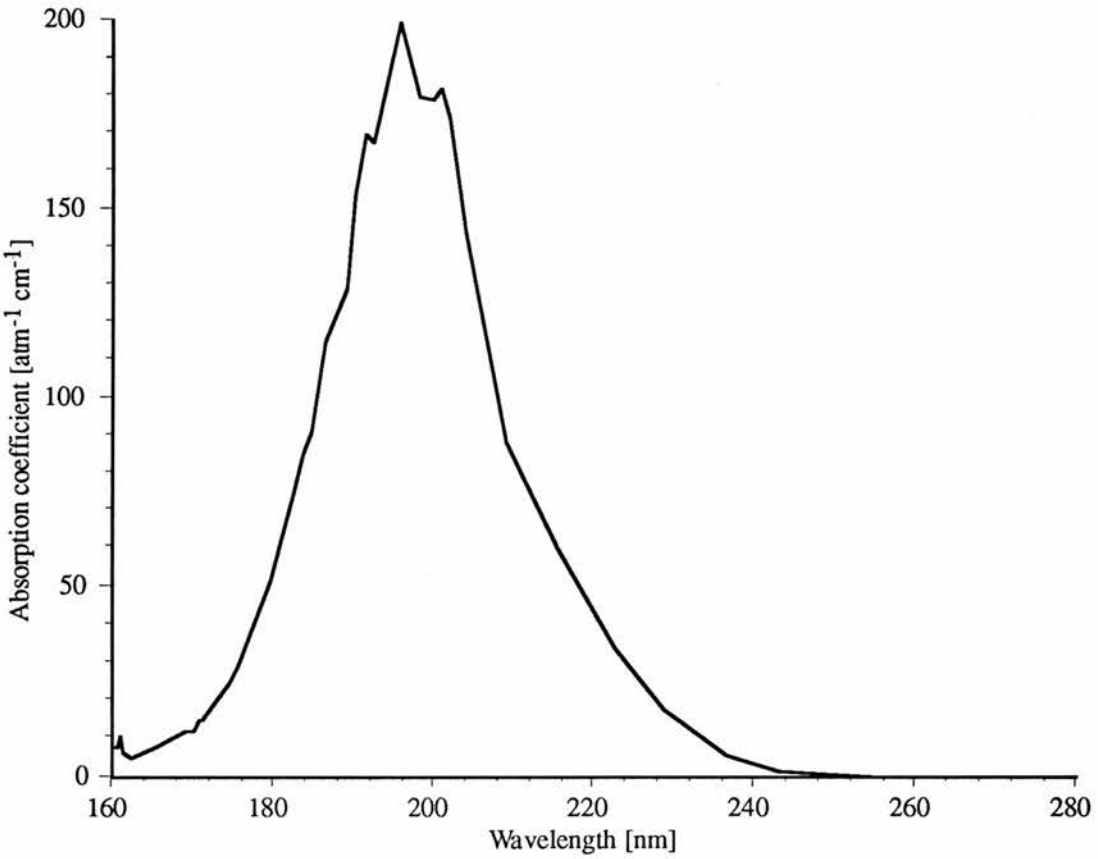


Figure 43: UV spectrum of hydrogen sulphide.

3.2.3. Interfering gases

The principal interfering gas is sulphur dioxide (SO_2), which absorbs over a similar range and with a similar strength to hydrogen sulphide but has a distinct line spectrum (fig. 44).

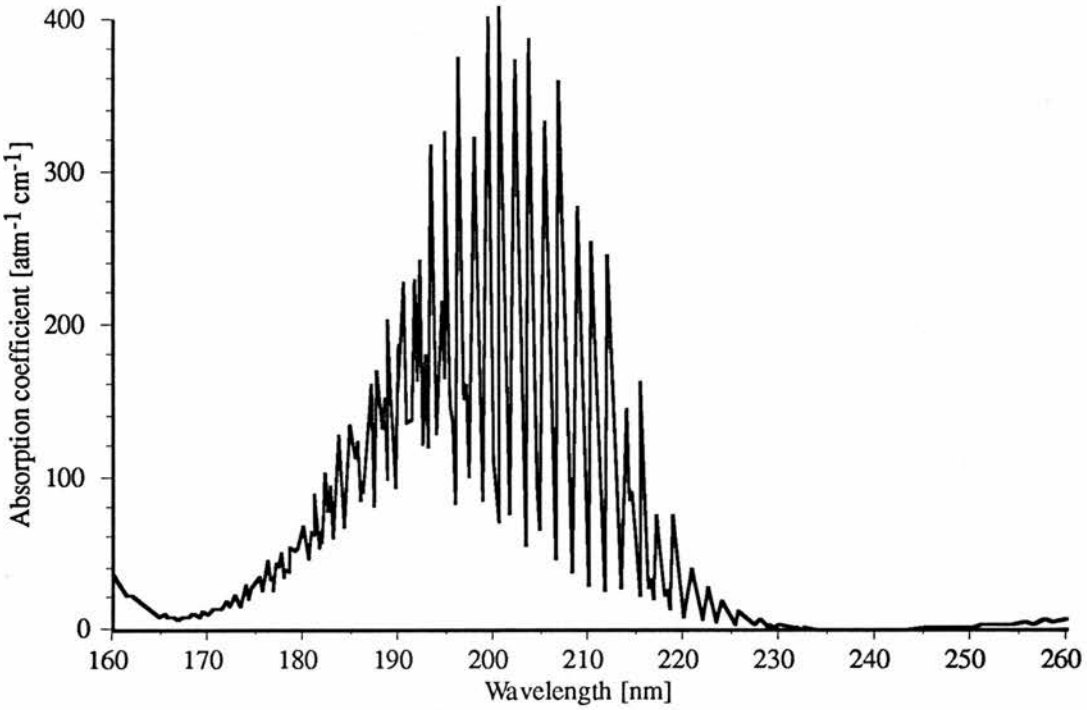


Figure 44: UV spectrum of sulphur dioxide.

3.2.4. Resolution requirement for spectrometer

To resolve the line structure of sulphur dioxide and hence differentiate it from hydrogen sulphide a resolution of approximately 1nm is required. A resolution of 1nm at 200nm implies a maximum path difference within a Fourier-transform spectrometer of approximately 40μm.

3.2.5. Operation in the UV and attenuation due to oil mist

During the project, concerns were encountered relating to the viability of optical detection in the UV spectral region. The chief concern is one of absorption of the light by oil mist either in the atmosphere itself or due to the build up of an oil film on any window or mirrors exposed to the environment. Independent work by St. Andrews University, Dräger and Shell has shown that a residual oil film on any of the optical components in the beam path would effectively block all light below 240nm, thereby preventing detection of hydrogen sulphide. Although such a loss of light is a detectable failure mode, it is believed that in oil production environments the requirement for regular cleaning would be excessive. This means that the detection of hydrogen sulphide in the UV is restricted to sites which do not have oil mist problems, e.g. refineries and gas-only production platforms. At wavelengths above 240nm, the absorption of light by oil mist is less of a problem and it may prove possible to operate the spectrometer at these wavelengths for the detection of other gases such as benzene and the other aromatics.

3.3. Design of the UV spectrometer based on Wollaston prisms

The key objective within this project was to design and construct a Wollaston prism based Fourier-transform spectrometer and demonstrate its application to the detection of hydrogen sulphide.

3.3.1. Optical layout of the UV static Fourier-transform spectrometer

The basic design of the spectrometer is centred around two Wollaston prisms (wedge angles 6° and 11.8°) fabricated from synthetic quartz (having useful transmission down to 190nm). The detector is a 1024 element clocked photodiode array (active area 25mm × 3mm). The polariser between the second Wollaston prism and the detector is an inclined plate (angle of incidence 70°), coated to enhance the reflection of the s-polarisation. A drawing using the software described in appendix B is shown in fig. 45; note the inclination of the localised fringe plane.

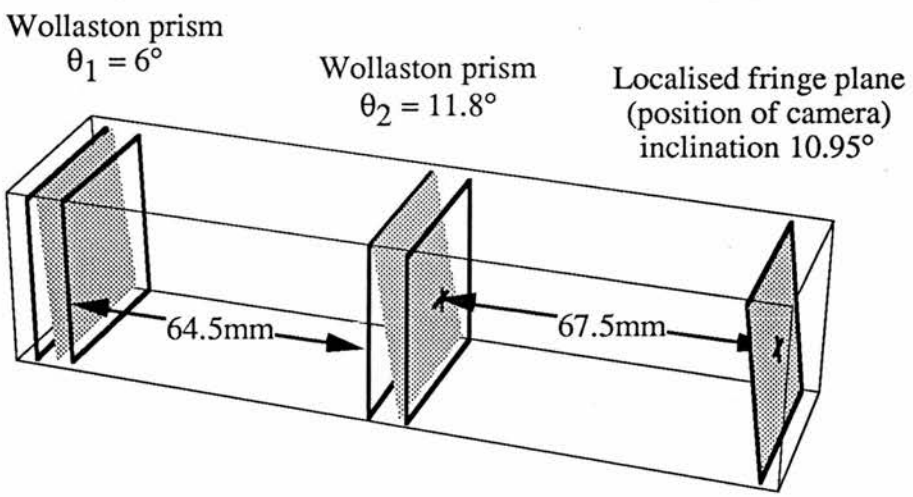


Figure 45: The Wollaston prisms and the position of the localised fringe plane in the UV static Fourier-transform spectrometer.

The opto-mechanical layout of the spectrometer is shown in figure 46.

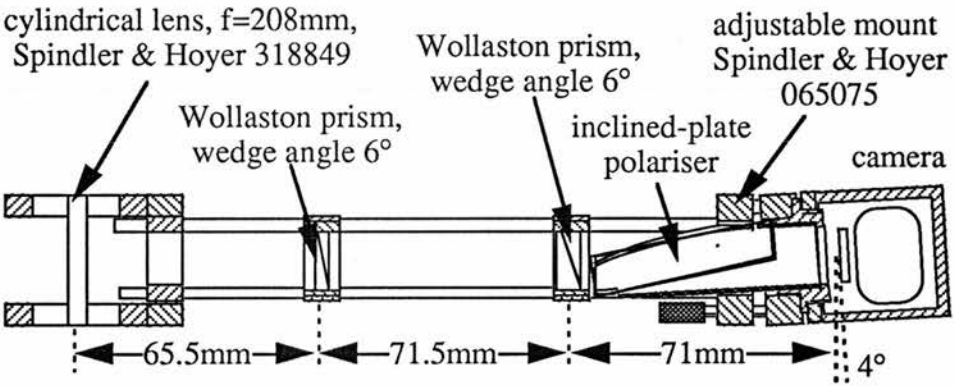


Figure 46: The opto-mechanical layout of the UV static Fourier-transform spectrometer.

The software described in the previous section can be used to predict the field of view of the spectrometer. Figure 47 shows the predicted interferogram as measured in an experiment as described in fig. 13 with the two Wollaston prisms used in the UV spectrometer. The extent of the central fringe corresponds to the field of view of the instrument.

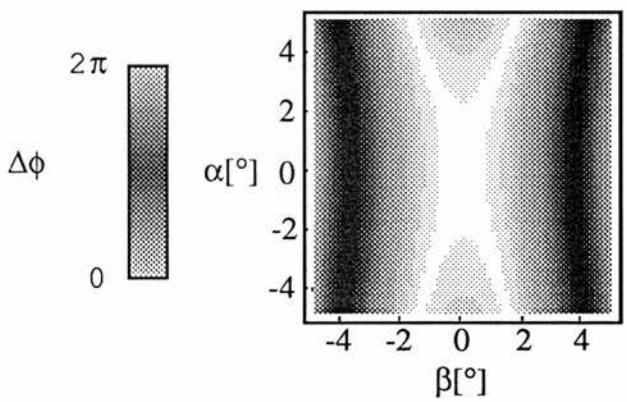


Figure 47: The predicted interferogram as generated by an experiment as described in fig. 13.

3.3.2. Optical design of the UV deuterium light source

The light source used within the demonstration UV spectrometer is a deuterium arc lamp. The lamp assembly also includes a three element collimation lens and a UV band pass filter centred at 220nm. The opto-mechanical layout is shown in fig. 48.

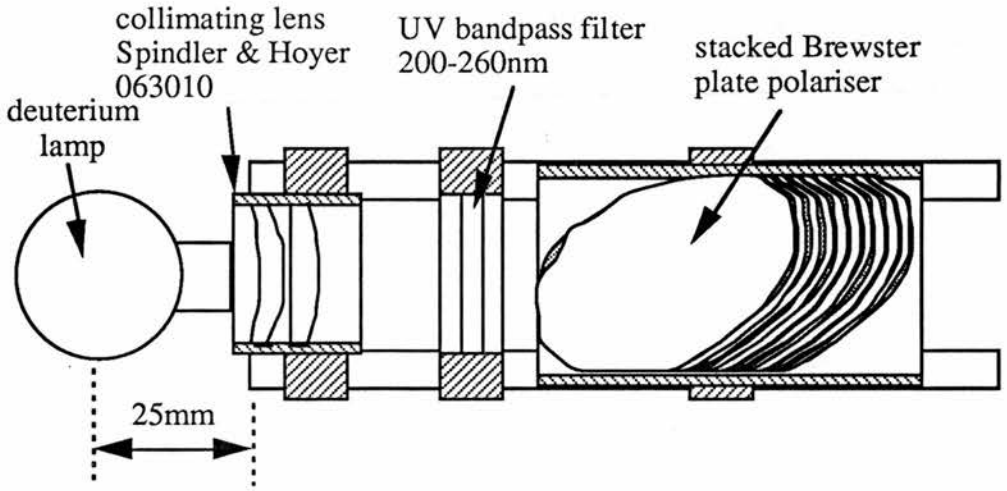


Figure 48: The opto-mechanical layout of the UV light source used within the static Fourier-transform spectrometer.

3.4. Component selection

3.4.1. Camera

The main factor governing the choice of camera is the target spectral range of the spectrometer, i.e. 200-300nm. This rules out conventional CCD detectors which typically have a short wavelength cut-off at approximately 400nm. Two choices exist, firstly a CCD coated with a UV sensitive phosphor and secondly a silicon based photodiode array.

The CCD option requires a UV sensitive coating which can only be obtained from a small number of suppliers. For example EG&G use a "Lumogen" coating which is sensitive down to 120nm with a quantum-efficiency approaching 40%. The RA1024J (EG&G) array comprises 1024 × 1024 pixels and can be obtained in coated form at an acceptable cost (≈£1000 1 of).

The alternative is a clocked photodiode array which can be configured in a similar fashion to a CCD but has an inherent sensitivity from 200nm to 1100nm. For example, Cronin Electronics package an 1024 element, EG&G "S" series array with an analogue to digital converter to produce a linescan camera with digital output (≈£3500 1 of).

The table below summarises the advantages of each detector type.

detector	advantages	disadvantages
2D CCD array with UV coating	high sensitivity; small, square aperture (14mm × 14mm), reduces cost for other optical components	places stringent requirements on interface electronics
1D clocked photodiode array with digital interface	easy to interface; previous experience (low risk)	large, rectangular aperture (25mm × 3mm) implying increased cost for other optical components

For the purpose of the demonstration spectrometer we selected the clocked photodiode array since this gave the low risk route to implementation. However, the significantly lower cost of the 2D CCD array coupled with the lower associated component cost make this array the target choice for the prototype instruments. The detailed specification for the Cronin camera (marketed in the UK by Laser 2000) is given below.

number of pixels	1024
pixel dimensions	3mm × 25μm
spectral range	190-1100nm
digitisation	16 bit
max integration time	≈2sec
max data rate	≈50 frames/sec
interface requirement	digital I/O

An application program was written in THINK C to interface the camera to a Macintosh computer. It enables the user to read in data from different camera types and calculate the spectral distribution of the incoming light. The various correction terms in equation (13) can be turned on or off as appropriate. The program allows the spectrum to be plotted over various scales and with respect to a previously recorded reference spectrum.

3.4.2. Wollaston prisms

The main factor governing the choice of material from which to fabricate the Wollaston prism is the requirement for UV transparency. Three of the commonly used birefringent optical crystals are transparent in this region: magnesium fluoride, ammonium dihydrogen phosphate (ADP) and quartz. Of these, magnesium fluoride is difficult to polish and therefore expensive, ADP is hydroscopic and needs protection from air moisture. Consequently quartz was the preferred material. In its

water-free synthetic form the spectral range covers 180-3500nm. The birefringence of the quartz is nominally 0.009, but this increases rapidly as the short-wavelength cut-off is approached, and allowance needs to be made for this fact within the spectrometer software.

The specifications of the Wollaston prisms used within the UV spectrometer were calculated using the *Mathematica* program described in appendix B and are given below.

specification	prism 1	prism 2
material	synthetic quartz	synthetic quartz
dimensions	height 12mm width 27mm thickness 7mm	height 12mm width 27mm thickness 7mm
internal wedge angle	6°	11.8°

3.4.3. Polarisers

Again, the key factor in polariser selection is the requirement for UV transmission. Below approximately 240nm the polaroid sheet type polariser is no longer available. Stacked Brewster plates offer a wide bandwidth but are expensive. An alternative is a single Brewster plate coated to enhance the reflection of the s-polarisation. A number of optical component companies offer polarisers of this type, e.g. Laser Optik. The specifications of these polarisers are given in the following table.

wavelength range	≈200-260nm
angle of incidence	70°
clear aperture	25mm

The use of a non-normal angle of incidence polariser extends the length of the spectrometer, consequently the aperture of the instrument also dominates its length.

3.4.4. Light source

The light source is of one of the most critical components within the spectrometer. Obviously, a key requirement is a high output in the ultra-violet spectral region, but this needs to be accompanied with a long operating life. In the UV spectral region, the most efficient source of light is a deuterium arc lamp. The longest lifetime lamp we have identified is supplied by Cathodeon (>8000 hrs). However, Hamamatsu claim to be launching a new ceramic based product with increased lifetime specification. The specifications for the Cathodeon arc lamp used in the spectrometer are given below.

product number	Y02
spectral output	>180nm
lifetime	>8000 hr
power requirements	≈300mA at 10V

3.4.5. Optical filter to restrict operating wavelength

Although possessing a high ultra-violet output, the light source also emits strongly in the visible region of the spectrum. Given the higher sensitivity of the camera to light in the visible region, steps need to be taken to shield the camera from the visible emission to avoid saturation of the detector array. One solution is to place an ultra-violet band pass filter at the entrance aperture of the spectrometer. For example, Ealing Electro Optic make a range of metallic-dielectric filters with a ≈50nm bandwidth centred at wavelengths in the 200-300nm region. The specification of the filter used in the UV demonstration instrument is given below.

centre wavelength	220nm
bandwidth	45nm
peak transmission	30%

3.4.6. Optical assembly

The optical assembly of the demonstration UV spectrometer was based upon Spindler & Hoyer opto-mechanical components, thus maintaining a flexible design while imparting a degree of robustness to the overall instrument.

3.5. Experimental results

3.5.1. Hydrogen sulphide

The main target gas for the demonstration of the UV static Fourier-transform spectrometer was hydrogen sulphide. Due to the high toxicity of the gas it was necessary to perform all the gas measurements at Shell Research (Thornton), where gas handling facilities and associated safety monitoring equipment is available.

A 100mm long sample cell with a fused silica window was inserted into the optical path. The cell could be evacuated and filled with any desired mixture of nitrogen and hydrogen sulphide. The effective gas concentration can be related to the partial pressure of gas in the cell, the length of the sample cell and the proposed length over which the open path instrument would be operated.

The traces below show the recorded spectra for four different partial pressures of hydrogen sulphide contained within the sample cell. To obtain a figure of merit for the change in signal arising from a given gas concentration, the data has been fitted (using a simple least squares algorithm) to an absorption of Gaussian profile centred at 210nm with a full-width-at-half-maximum of 40nm. The fitted parameters include a variable base line, to allow for changes in source brightness, and the centre height of the Gaussian absorption. As can be seen from figures 49-52, the change in signal can be observed for a partial pressure corresponding to a detection level of 1 ppm over a 6 metre path difference.

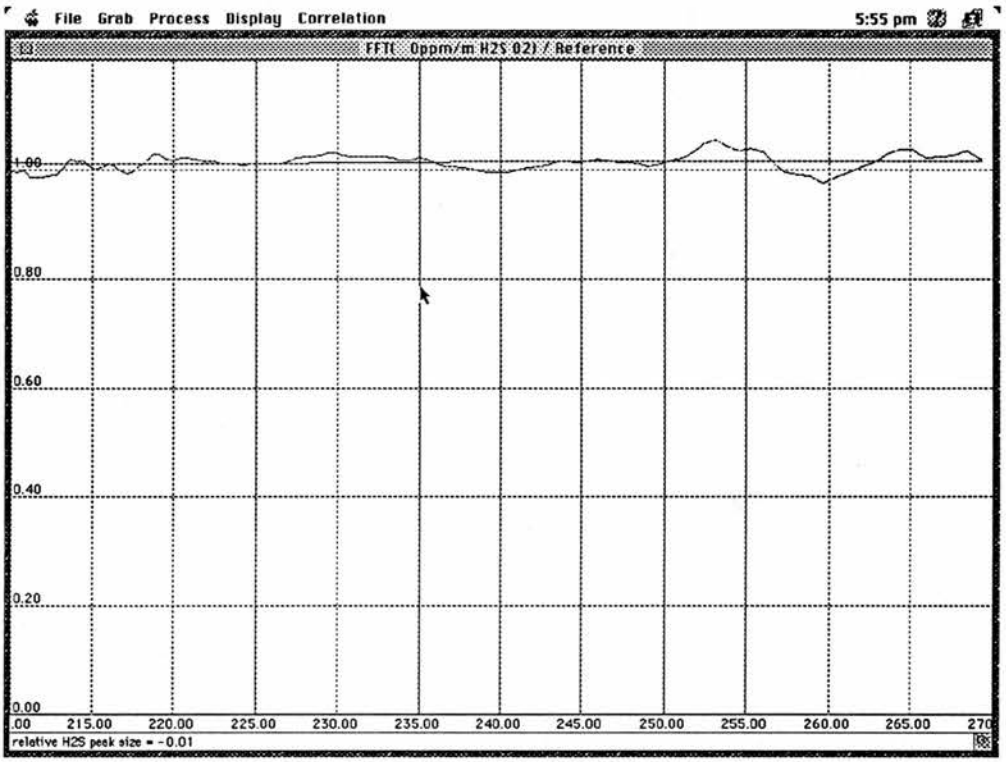


Figure 49: The transmission coefficient of the evacuated gas cell and a fitted Gaussian with an offset. The size of the Gaussian peak in arbitrary units is 0.01±0.01.

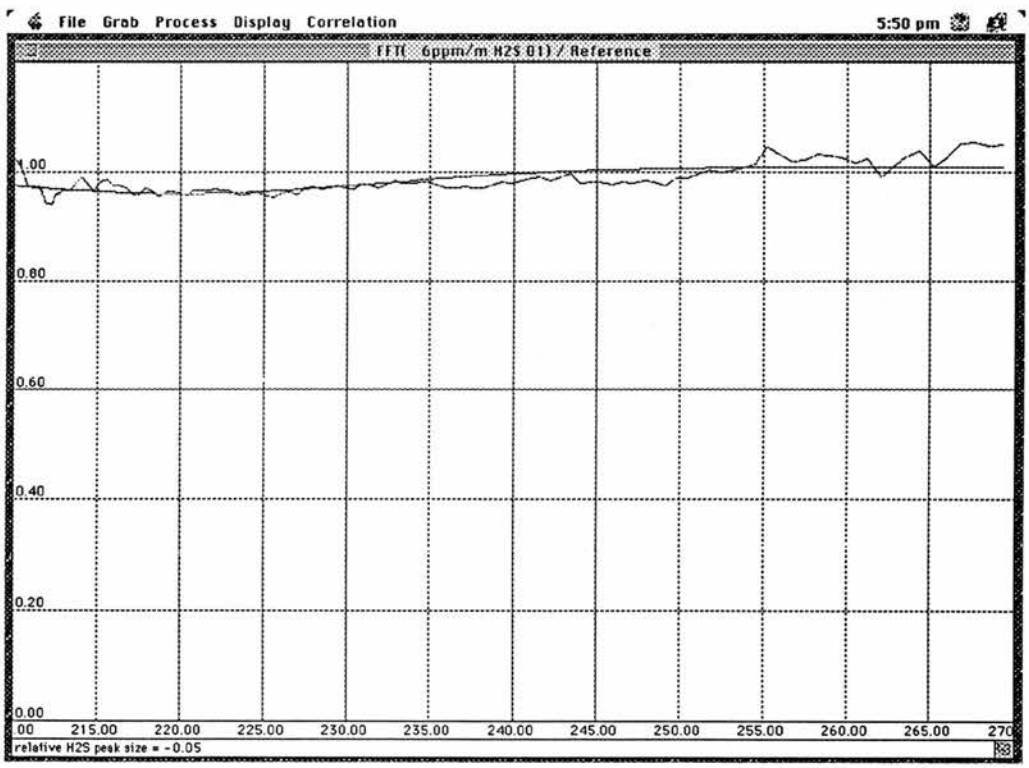


Figure 50: The transmission coefficient of the gas cell filled with a concentration of H_2S equivalent to 1ppm over a 6 metre path length. The size of the fitted Gaussian peak in arbitrary units is -0.05±0.01.

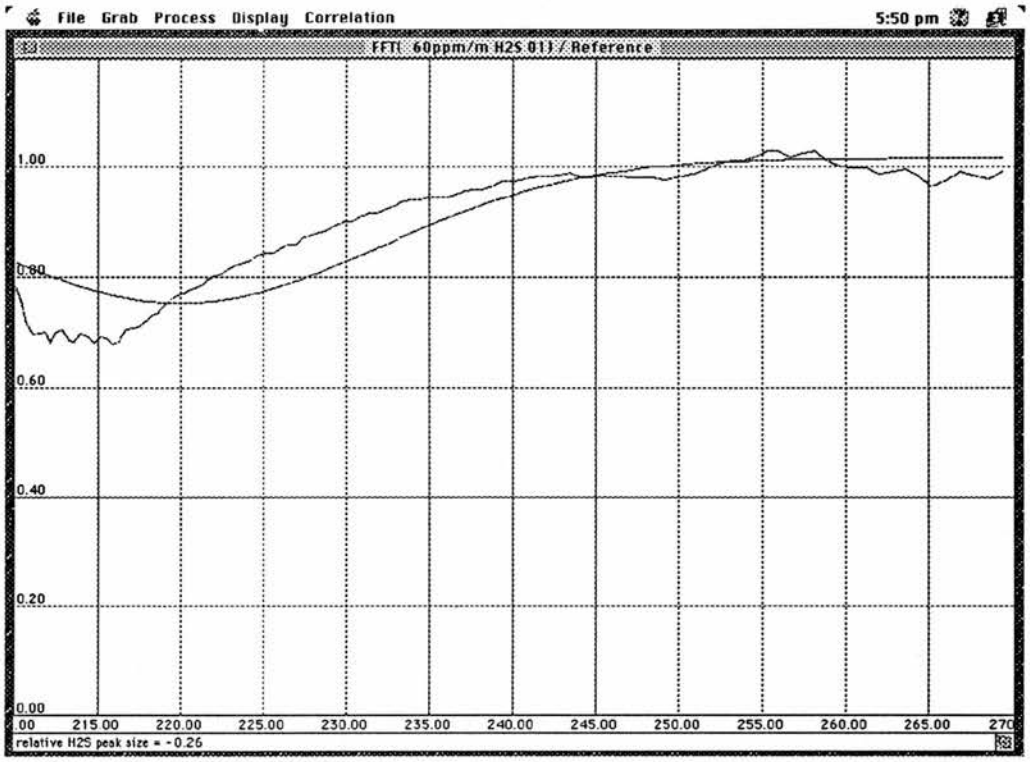


Figure 51: The transmission coefficient of the gas cell filled with a concentration of H_2S equivalent to 10ppm over a 6 metre path length. The size of the fitted Gaussian peak in arbitrary units is -0.26±0.01.

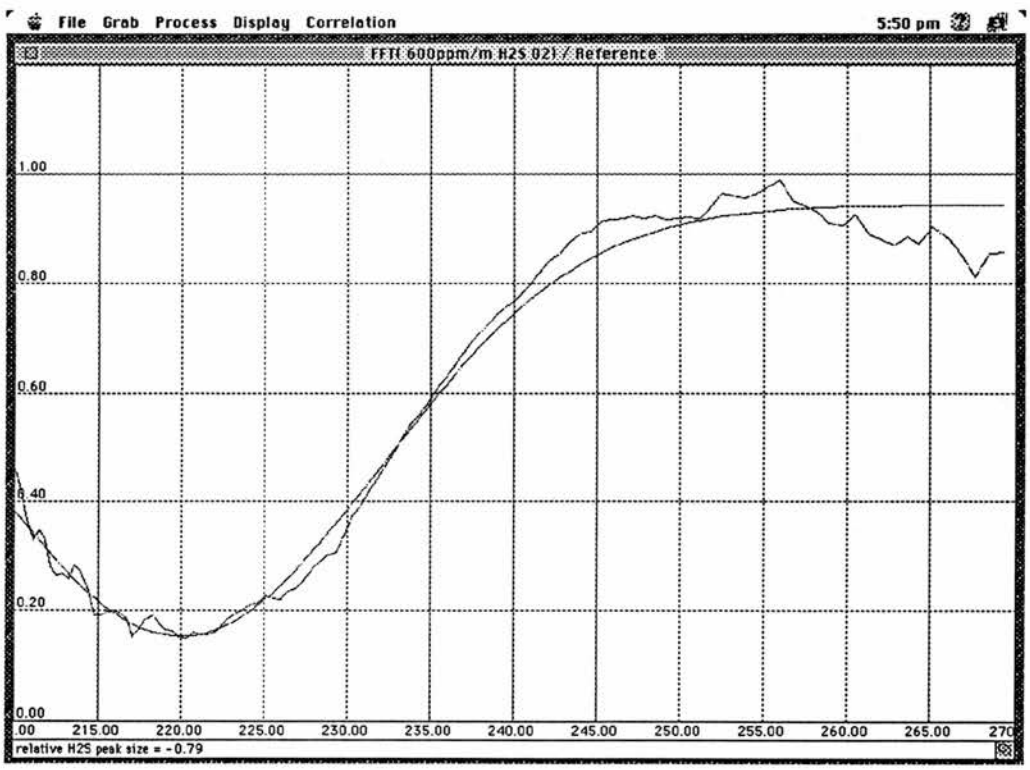


Figure 52: The transmission coefficient of the gas cell filled with a concentration of H_2S equivalent to 100ppm over a 6 metre path length. The size of the fitted Gaussian peak in arbitrary units is -0.79±0.01.

These initial results are extremely encouraging in that the required sensitivity for the detection of hydrogen sulphide has been demonstrated within the laboratory. The simple curve-fitting algorithm could be extended to relate the absorption strength to the amount of gas present. However, it is believed that alternative algorithms may offer more sensitive and more selective identification of gas species.

3.5.2. Other gases

The main advantage of a spectrometer over a single wavelength measuring device (such as a filter instrument) is that the availability of the full spectrum allows the identification of different gas species. To this end it was hoped within the initial testing phase to compare the signals arising from hydrogen sulphide with those from sulphur dioxide. Unfortunately, due to safety considerations, this was not possible within the initial gas trials. However, it should be emphasised that as can be seen in the previous section, the absorption spectra of hydrogen sulphide and sulphur dioxide differ significantly. There is little doubt that an appropriately designed instrument of this type will be able to distinguish between these gases. the only question is what degree of discrimination will be possible and this depends largely on the data processing algorithms employed.

Conclusions

In this work a theoretical study of the interferogram generated by a Wollaston prism based spectrometer has been presented. The techniques and results can be used to optimise the properties of Wollaston prism based spectrometers and predict their behaviour. In addition, correction factors to the calculation of the spectrum have been derived for the dispersion in birefringence, fringe visibility, pixel function and the spectral response of the camera.

Following the theoretical study, a UV spectrometer has been designed and successfully demonstrated. Special emphasis was paid to the detection of hydrogen sulphide. Using an enclosed gas cell, a detection limit better than 1ppm over 6m path length was demonstrated.

Acknowledgements

Firstly, I would like to thank my supervisors, Miles Padgett and Wilson Sibbett. Miles not only provided invaluable guidance and friendship, he also bore the brunt of making this work more accessible for native English readers. Also involved were Brett Patterson and Gregory Morris. Additionally, Brett was always available to discuss problems and install system extensions, and is, incidentally, the only person I have ever won a round of golf against who had seen a golf ball before.

I am grateful to Siemens UK, British Gas, Dräger and Shell Research for funding this work. In particular I would like to thank Bill Hirst, who organised the gas testing facility at Shell and looked after us so well.

Part of this thesis was written under the late-summer Italian sun. This enforced holiday was thanks to the fact that I had nowhere to stay in St. Andrews and to Sylvia Mieszkowski, who was staying in Florence at this time and prepared to put up with me so patiently.

I am also grateful to Neil Simpson, who shared his flat with all my possessions while I had to work in Italy, and provided musical entertainment and golf statistics in the lab.

My special thanks go to my friends and family back in Germany who kept me busy with correspondence and visits.

References

[1] M J Padgett, A R Harvey, A J Duncan, W Sibbett, *Single-pulse, Fourier-transform spectrometer having no moving parts,* Applied Optics **33**, 6035 (1994)

[2] M J Padgett, A R Harvey, *A static Fourier-transform spectrometer based on Wollaston prisms*, Rev. Sci. Instrum. **66**, 2807 (1995)

[3] A R Harvey, M Begbie, M J Padgett, *Stationary Fourier-transform spectrometer for use as a teaching tool*, Am. J Phys. **62**, 1033 (1994)

[4] A R Harvey, D Burns, A Duncan, K Lamb, W Sibbett, M J Padgett, *Fourier Domain Parallel Optical Spectral Processing*, Euro CLEO'94, CPD1.5 (1994)

[5] J Courtial, M J Padgett, A R Harvey, W Sibbett, *The Use of a Static Fourier-transform Spectrometer for Pollution Monitoring*, project report for Siemens (October 1994)

[6] J Courtial, B Patterson, A R Harvey, W Sibbett, M J Padgett, *An investigation into the application of a static Fourier-transform spectrometer based on Wollaston prisms for the detection of hydrogen sulphide and other gases*, project report for Siemens, British Gas, Dräger, Shell (July 1995)

[7] R J Bell, *Introductory Fourier-Transform Spectroscopy*, Academic, New York (1972)

[8] J Courtial, B A Patterson, W Sibbett, A R Harvey, M J Padgett, *Design of a static Fourier-transform spectrometer with increased field of view*, submitted to Applied Optics (September 1995)

[9] B A Patterson, M Antoni, J Courtial, W Sibbett, M J Padgett, *An ultra-compact static Fourier-transform spectrometer based on a single birefringent component*, in preparation

[10] M C Simon, R M Echarri, *Ray tracing formulas for monoaxial optical components: vectorial formulation*, App. Opt. **25**, 1935-9 (1986)

[11] M Françon, S Mallick, *Polarization Interferometers*, Wiley-Interscience, London

[12] E Hecht, A Zajac, *Optics*, Addison-Wesley, Reading (1974)

[13] Wolfram Research, Inc., *Mathematica*, Version 2.2, Champaign, Illinois, USA (1994)

[14] MathSoft Inc., *MathCad*, Version 3.1

[15] K Watanabe, A S Jursa, *Absorption and Photoionization Cross Sections of H_2O and H_2S*, J. Chem. Phys. **41**, 1650-2 (1964)

[16] M T Emerson, D F Eggers, *Effect of Centrifugal Distortion on the Shape of the Hydrogen Sulfide Fundamental Infrared Bands*, J. Chem. Phys. **37**, 251-9 (1962)

[17] H C Allen, E K Plyler, *Infrared Spectrum of Hydrogen Sulfide*, J. Chem. Phys. 25, 1132-6 (1956)

[18] R H Pierson, A N Fletcher, E S C Gantz, *Catalog of Infrared Spectra for Qualitative Analysis of Gases,* Analytical Chemistry **28**, 1218-39 (1956)

[19] C F Goodeve, N O Stein, *The Absorption Spectra and the Optical Dissociation of the Hydrides of the Oxygen Group*, (1931), transcribed by S A Reid, S Gillespie, *Open Path Detection of Hydrogen Sulphide*, International Symposium on Optical Remote Sensing for Environmental Monitoring, Atlanta, Georgia (1993)

A The ray tracing package

A.1. *Mathematica* code for the ray tracing package

The following *Mathematica* program was written within this work as the basis for programs which then perform calculations related to the optical performance of a Wollaston prism based spectrometer. It can be used as a general ray tracing tool. See also the demonstration program at the end of this appendix.

```
(* References: [1] M C Simon, R M Echarri, "Ray tracing formulas
                   for monoaxial optical components: vectorial
                   formulation",  Applied Optics 25, 1935-9
                   (1986)
               [2] Quan-Ting Liang, "Simple ray tracing formulas
                   for uniaxial optical crystals", Applied
                   Optics 29, 1008-10 (1990)
               [3] S C McClain, L W Hillman, R A Chipman,
                   "Polarization ray tracing in anisotropic
                   optically active media. I. Algorithms",
                   J.Opt.Soc.Am.A 10, 2371-82 (1993)
 *)
Clear["RayTrace`*"];
BeginPackage[ "RayTrace`",
              { "LinearAlgebra`CrossProduct`",
                "LinearAlgebra`Orthogonalization`",
                "Graphics`Graphics3D`",
                "Graphics`ContourPlot3D`"
              }
            ];
```

usages

```
(* public data structures *)
intersection::usage =
"intersection[ <vector to position of the intersection>,\n"<>
"              <unit vector normal to surface>,\n"<>
"              <unit vector normal to incoming wave front>,\n"<>
"              <refractive index incoming wave sees>,\n"<>
"              <cumulated optical path till intersection>\n"<>
"              <list of all former intersections>\n"<>
"            ] is a data structure holding information about an
 intersection between a light ray and a discontinuity
 surface.\n"<>
"intersection[ evanescent, <list of intersection points> ] holds
 information about an intersection between an exponentially
 decaying light ray and a discontinuity surface.\n"<>
"intersection[ $Failed, <list of intersections before error
 occured> ]";
ray::usage =
"ray[ <starting point>,\n"<>
"     <ray direction>,\n"<>
"     <wave direction>,\n"<>
"     <refractive index the wave sees>,\n"<>
"     <cumulated optical path>,\n"<>
"     <list of intersection points so far>\n"<>
"   ] is a data structure holding information about a light "<>
"ray. For non-axial ray tracing an additional argument "<>
"which holds the E-vector (which is pointing in the "<>
```

```
"direction of polarisation and whose modulus being "<>
"greater or equal to DarkE is the criterion for a ray not "<>
"to be considered dark) has to be supplied\n"<>
"ray[ evanescent, <list of intersection points> ] describes "<>
"an exponentially decaying ray\n"<>
"ray[ $Failed, <list of intersection points until error "<>
"occured> ].";
surface::usage =
"surface[ <equation in x, y, z> (, <condition for
 correct part of surface>) ]
 is a data structure used to describe an interface
 between two different materials; an isotropicMaterial[...]
 or birefringentMaterial[...] structure should follow.
 If the <equation> can have more than one solutions for
 an intersection with a ray, provide as second argument
 a condition in x, y, z which identifies the correct part
 of the surface.\n"<>
"surface[ <equation of surface>,\n"<>
"         <limiting conditions for the surface>,\n"<>
"         <material on -ve gradient side of surface>,\n"<>
"         <material on +ve gradient side of surface>\n"<>
"       ]
 is used to describe surfaces through which
 light can pass in both directions.";
x::usage =
y::usage =
z::usage =
"x, y, z are the coordinates used in the equations "<>
"describing surfaces. The optic axis of the system "<>
"is parallel to the z axis.";
birefringentMaterial::usage =
"birefringentMaterial[ <ordinary refractive index>,\n"<>
"                      <extraordinary refractive index>,\n"<>
"                      <crystal axis>\n"<>
"                    ] is a data structure used by
 rayTrace[...]";
isotropicMaterial::usage =
"isotropicMaterial[ <refractive index> ] is a data
 structure used by rayTrace[...]";
o::usage =
e::usage =
b::usage =
"rayTrace[...][[o,e]] picks the component which "<>
"is ordinary in the first birefringent medium and "<>
"extraordinary in the second.\n"<>
"rayTrace[..., Component->{o,b,e}] returns a list "<>
"whose first part is the component which is "<>
"ordinary in the first and 2nd birefringent medium "<>
"and extraordinary in the 3rd one and whose 2nd "<>
"part is the component which is ordinary in the "<>
"first birefringent medium and extraordinary in "<>
"the 2nd and 3rd.";
(* public functions *)
rayTrace::usage =
"rayTrace[ <ray[...] or intersection[...]>,\n"<>
"          { optical system }\n"<>
"        ]
 -> <resulting ray[...] or intersection[...]>\n"<>
"The first argument 'hits' the optical system. "<>
"If the first argument is a list, each element of the list "<>
"hits the optical system. The optical system is a list of "<>
"surfaces (see \"surface\") and materials (see "<>
"\"isotropicMaterial\" and \"birefringentMaterial\")."<>
```

```
"The components of the optical system are hit one after "<>
"another. A ray hitting a surface generates an intersection, "<>
"an intersection hitting an isotropicMaterial gives a ray, "<>
"and an intersection impinging on a birefringentMaterial "<>
"gives a list of two rays, the ordinary ray and the "<>
"extraordinary ray. If the optical system contains "<>
"more than one interface followed by a birefringent "<>
"material, "<>
"rayTrace[[<o|e>.1,<o|e>.2,...]] "<>
"picks out that component which is "<>
"o(rdinary)|e(xtraordinary) in the "<>
"first(.1)/2nd(.2)/... birefringent material "<>
"following an interface. "<>
"The results from hitting the first "<>
"component then hit the rest of the optical system.\n"<>
"Options:\n"<>
"* Component -> <list describing the component> "<>
"returns only the described component without "<>
"calculating the others; the list describing the "<>
"component has the format {<o|e|b>, <o|e|b>, ...}.\n"<>
"* Trace -> Axial or NonAxial";
snellsLaw::usage =
"snellsLaw[ <unit vector normal to incident wave front>,\n"<>
"           <refractive index incoming wave sees>,\n"<>
"           <unit vector normal to discontinuity surface>,\n"<>
"           <refractive transmitted wave sees>\n"<>
"         ]
 -> <unit vector normal to transmitted wave front>
 or \"evanescent\" (if total reflection occurs)";
toIsotropic::usage =
"toIsotropic[ <unit vector normal to incident wave front>,\n"<>
"             <refractive index incoming wave sees>,\n"<>
"             <cumulated optical path>,\n"<>
"             <list of intersection points so far>,\n"<>
"             <vector to intersection point between incoming\n"<>
"              ray and discontinuity surface>,\n"<>
"             <unit vector normal to discontinuity surface\n"<>
"              at intersection point>,\n"<>
"             <refractive index transmitted wave sees>\n"<>
"           ]
 -> <ray[...] structure corresponding to refracted ray>";
toMonoaxial::usage =
"toMonoaxial[ <unit vector normal to incident wave front>,\n"<>
"             <refractive index incoming wave sees>,\n"<>
"             <cumulated optical path till intersection>,\n"<>
"             <list of intersection points so far>,\n"<>
"             <vector to intersection point>,\n"<>
"             <unit vector normal to surface at intersection>,\n"<>
"             <ordinary refractive index>,\n"<>
"             <extraordinary refractive index>,\n"<>
"             <unit vector parallel to crystal axis>\n"<>
"           ]\n"<>
" -> { <ray[...] structure corresponding to ordinary ray>,\n"<>
"      <ray[...] structure corresponding to extraordinary ray>\n"<>
"    }";
normalToSurface::usage =
"normalToSurface[ <equation of surface in x, y, z>,\n"<>
"                 <point on surface where unit vector\n"<>
"                  normal to surface is to be computed>\n"<>
"               ]
 -> <unit vector normal to surface at given point>";
rayInAir::usage =
"rayInAir[ <vector pointing to starting point>,\n"<>
```

```
"          <vector parallel to ray direction>\n"<>
"          <E-vector>\n"<>
"        ]
 -> <ray[...] structure corresponding to ray in air>";
draw::usage =
"draw[ <any sequence of possible arguments to rayTrace[...]> ]
 -> <a graphics structure, after it was drawn>\n"<>
"Options: see Options[draw]; any options for the functions "<>
"Plot3D, ContourPlot3D, Show and Graphics3D can be added";
evanescent::usage =
"ray[evanescent, ...] and intersection[evanescent, ...] "<>
"indicate that total reflection occured.";
dark::usage =
"ray[dark, ...] and intersection[dark, ...] "<>
"indicate that the modulus of the E vector of the ray "<>
"is less than the lower limit, which is given by the option "<>
"DarkE.";
world::usage =
"world is used instead of a material to indicate "<>
"the outer boundary of an object.";
```

options

```
Options[rayTrace] = { Trace -> Axial,
                      Component -> {},
                      DarkE -> 10^-3 };
Options[draw] = { Range -> { Automatic,
                             Automatic,
                             Automatic },
                  StandardRangeWidth -> {-1, 1},
                  (* options which are passed to other
                     functions *)
                  Axes          -> True,
                  AxesLabel     -> {"x", "y", "z"},
                  BoxRatios     -> {1, 1, 4},
                  Mesh          -> False,
                  ViewPoint     -> {-1.578, -2.914, 1},
                  ViewVertical -> {-1, 0, 0}
                };
```

private parts

```
Begin["`Private`"];
constants
(* constants *)

c = 2.998*10^8;    (* m / s *)
o = 1;
e = 2;
```

print formats

```
(* some print formats *)
Form[ ray[P_, R_, K_, n_, d_, {i___}, r___] ] :=
  ray[P, R, K, N, d, "-intersectionList-", r];
Form[ intersection[ p_,eta_,ki_,ni_,d_,{i___}] ] :=
  intersection[p, eta, ki, ni, d, "-intersectionList-"];
```

messages

```
pickSolution::noSolution =
"Warning: no intersection which satisfies the "<>
"condition for the surface validity found.";
rayTrace::noIntersection =
"Warning: no intersection between ray and "<>
"surface found to satisfy the surface condition.";
rayTrace::evanescent =
```

```
"Warning: evanescent ray occurs";
rayTrace::backwards =
"Warning: ray travels backwards; you might want to "<>
"check the order of the discontinuity surfaces;"<>
"rayTrace failed";
rayTrace::rayHitsMaterial =
"Warning: ray hits material instead of surface; "<>
"ignored";
```

vectors

```
(* useful vector function *)
norm[v_] := Sqrt[v.v];
```

refraction

```
(* some optical formulas from [1] *)

(* see eqs. (5) and (6) in [1] *)
snellsLaw[ s_,          (* unit vector normal to incident wave
                           front
                        *)
           ni_,         (* refractive index incoming wave sees *)
           eta_,        (* unit vector normal to discontinuity
                           surface
                        *)
           nt_          (* refractive index for transmitted wave *)
         ] :=
  Module[ {n, r},
          n = Sign[s.eta] eta;
          r = (nt/ni)^2-1+(s.n)^2;
          If[ (r >= 0) && (s.n != 0),
              Normalize[ s + ( Sqrt[r] - (s.n) ) * n ],
              evanescent      (* [3], p. 2374;
                                 total reflection occurs
                              *)
            ]
        ];

toIsotropic[ s_,  (* unit vector normal to incident wave front *)
             ni_, (* refractive index for incoming wave *)
             d_,  (* cumulated optical path between
                     first starting point and p
                  *)
             i_,  (* list of intersection points *)
             p_,  (* point of intersection between incoming
                     ray and discontinuity surface
                  *)
             eta_,      (* unit vector normal to discontinuity
                           surface at intersection point
                        *)
             nt_  (* refractive index for transmitted wave *)
           ] :=
  Module[ { st },
          st = snellsLaw[ s, ni, eta, nt ];
          If[ st =!= evanescent,
              ray[ p,   (* starts from intersection point *)
                   st,  (* ray direction is the same as normal
                           to wavefront in isotropic medium
                        *)
                   st,  (* normal to wavefront *)
                   nt,  (* refractive index of medium the
                           refracted ray travels in
                        *)
                   d,   (* already calculated *)
                   i
```

```
                ],
              Message[rayTrace::evanescent];
              ray[ evanescent, i ]
            ]
        ];

ComplexQ[a_] := Head[a] === Complex;
toMonoaxialO[ s_,          (* unit vector normal to incident wave
                              front
                           *)
              ni_,         (* refractive index incoming ray sees *)
              d_,          (* cumulated optical path between
                              first starting point and p
                           *)
              i_,          (* list of intersection points so far *)
              p_,          (* point of intersection between incoming
                              ray and discontinuity surface
                           *)
              eta_,        (* unit vector normal to discontinuity
                              surface at intersection point
                           *)
              no_,         (* ordinary refractive index in
                              birefringent medium after surface
                           *)
              _,           (* extraordinary refractive index after
                              surface
                           *)
              _            (* unit vector parallel to crystal axis *)
            ] :=
  toIsotropic[s, ni, d, i, p, eta, no];
toMonoaxialE[ s_,          (* unit vector normal to incident wave
                              front
                           *)
              ni_,         (* refractive index incoming ray sees *)
              d_,          (* cumulated optical path between
                              first starting point and p
                           *)
              i_,          (* list of intersection points so far *)
              p_,          (* point of intersection between incoming
                              ray and discontinuity surface
                           *)
              eta_,        (* unit vector normal to discontinuity
                              surface at intersection point
                           *)
              no_,         (* ordinary refractive index in
                              birefringent medium after surface
                           *)
              ne_,         (* extraordinary refractive index after
                              surface
                           *)
              z3_          (* unit vector parallel to crystal axis *)
            ] :=
  Module[ {n, u, uo, ue, b, Z, A, B, C, w, nt, N},

          (* make n point into same half space as s *)
          n = Sign[s.eta] eta;

          (* phase velocities *)
          u  = c / ni;
          uo = c / no;
          ue = c / ne;

          (* calculate unit vector normal N to e-wavefront *)
```

```
b = (uo^2 - ue^2) / u^2;  (* see Eqn. (12) in [1] *)
(* see Eqns. (19) - (21) *)
A = ( 1 + b * ( 1 - (s.n)^2 -
                  (s.Cross[n, z3])^2
                )
    )^2 -
    4 * b * ( (1 - (s .n)^2) *
              (1 - (z3.n)^2) -
              (s.Cross[n, z3])^2
            );
B = 2 * ( 1 + b * ( 1 - (s.n)^2 -
                      (s.Cross[n, z3])^2
                    )
        ) *
    ( b * (z3.n)^2 + (ue / u)^2 ) -
    4 * b * (ue / u)^2 * ( (1 - (s .n)^2) *
                           (1 - (z3.n)^2) -
                           (s.Cross[n, z3])^2
                         );
C = ( b * (z3.n)^2 + (ue / u)^2 )^2;
(* Z is defined in equation (18) *)
If[ norm[Cross[n, z3]] == 0,
    w = Sqrt[B / (2*A)],
    Z = Normalize[ Cross[ n, Cross[n, z3] ] ];
    (* Eqns. (22) and (23) *)
    w = Sqrt[ ( B + Sign[ (uo - ue) *
                          (z3.Z) *
                          (z3.n) *
                          (s .Z)
                        ] * Sqrt[B^2 - 4*A*C]
              ) / (2*A)
            ]
  ];

(* check if transmitted e-ray occurs at all *)
If[ ComplexQ[w] || ni Sqrt[1-(s.n)^2] > ni / w,

    (* internal reflection occurs for e-ray *)
    Message[rayTrace::evanescent];
    { toIsotropic[s, ni, d, i, p, n, no],    (* o-ray *)
      ray[ evanescent, i ]
    },

    (* calculate transmitted e-ray *)
    (* compare (25) and (6) to find w = ni/nt, where
       nt is the refractive index the transmitted
       extraordinary ray sees
     *)
    nt = ni / w;
    N = snellsLaw[ s, ni, n, nt ];
      (* [1](24), effectively *)
    ray[ p,    (* starts from intersection point *)
         Normalize[ N  * ue^2 +
                    z3 * (uo^2 - ue^2) * (N.z3)
                  ], (* ray direction, calculated
                        with equation (32)
                     *)
         N,          (* normal to wavefront *)
         nt,         (* refractive index of medium the
                        refracted ray travels in
                     *)
         d,          (* already calculated *)
```

```
                    i
                ]
            ]
        ];
toMonoaxial[ a__ ] :=
  { toMonoaxialO[a], toMonoaxialE[a] };
```

Fresnel coefficients (non-axial ray trace only)

```
Fresnel[ nI_,      (* refractive index incoming and
                      reflected rays see *)
         (* incoming ray *)
           kI_,    (* wave direction (normalised) *)
           EI_,    (* E field amplitude vector *)
         kR_,      (* reflected ray *)
         eta_,     (* normal to surface (normalised) *)
         nT_,      (* refractive index transmitted ray sees *)
         kT_       (* transmitted ray *)
       ] :=
  Module[ { eta1, cosI, cosR, cosT,
            EPer, ePerI, eParI, ePerR, ePerT, eParR, eParT},

          (* make kI and eta point into the same half space *)
          eta1 = eta Sign[eta.kI];

          (* calculate the cosini of the angles with eta *)
          cosI = cosR = eta1.kI;
          cosT = eta1.kT;

          (* calculate the direction of the E-field
             component perpendicular to the
             plane of incidence *)
          If[ N[kI.eta1] == 1.,

              (* normal incidence *)
              EPer = Normalize[ EI ],

              (* non-normal incidence *)
              EPer = Normalize[ Cross[kI, eta1] ]
            ];

          (* split the incoming E-field into a perpendicular
             and a parallel component *)
          ePerI = EI.EPer;
          eParI = Sqrt[ EI.EI + ePerI^2 ];

          (* calculate the amplitudes of the E-field
             components; see Hecht, (4.34), (4.35),
             (4.40) and (4.41) *)
          ePerR = ePerI (nI cosI - nT cosT) /
                        (nI cosI + nT cosT);
          ePerT = ePerI 2 nI cosI / (nI cosI + nT cosT);
          eParR = eParI (nT cosI - nI cosT) /
                        (nI cosT + nT cosI);
          eParT = eParI 2 nI cosI / (nI cosT + nT cosI);

          (* return the E-vectors for the reflected and
             transmitted rays *)
          { ePerR EPer + eParR Normalize[ Cross[kI, EPer] ],
            ePerT EPer + eParT Normalize[ Cross[kT, EPer] ]
          }
        ];
```

2-way surfaces (non-axial ray trace only)

```
(* surfaceDistanceAndIntersectionPosition *)
```

```
surfaceDistanceAndIntersectionPosition[ P_, Rho_,
                                        eqn_, condition_
                                      ] :=
  Module[ {s, si, d, dMin, PMin, i},

          (* look for solutions of the set of equations
             describing intersections between the ray
             starting at P with direction Rho and the
             surface
           *)
          s = Solve[ Append[ P + d Rho /.
                             {xR_, yR_, zR_} ->
                               {xR==x, yR==y, zR==z},
                             eqn
                           ]
                   ] //N;

           (* check if the solutions satisfy the
              condition imposed on the surface boundaries
              and look for the closest intersection *)
           dMin = Infinity;
           For[ i = Length[s], i > 0, i = i-1,

                si = s[[i]];

                If[ ((condition) /. si) &&
                    ((d /. si) > 0.001) &&
                    ((d /. si) < dMin),

                    (* new smallest distance *)
                    dMin = (d /. si);
                    PMin = ({x, y, z} /. si)
                  ]
              ];
           If[ dMin == Infinity,

               (* no hits *)
               {noHits},

               (* return the nearest distance *)
               {dMin, PMin}
             ]
         ];
(* findSurfaceDistance *)
findSurfaceDistance[ P_, Rho_,
                     surface[ eqn_, condition_, ___ ]
                   ] :=
  surfaceDistanceAndIntersectionPosition[ P, Rho,
                                          eqn, condition
                                        ][[1]];
(* traceThroughSurface *)
traceThroughSurface[ ray[ P_, (* starting point *)
                          Rho_,     (* ray direction *)
                          K_, (* wave direction *)
                          n_, (* refractive index in
                                 wave direction *)
                          d_, (* cumulated optical
                                 path *)
                          i_, (* intersection points
                                 so far *)
                          E_, (* E field amplitude
                                   vector *)
                          od___     (* other data *)
```

```
                    ],
                surface[ eqn_,
                         condition_,
                         isotropicMaterial[nNeg_],
                         isotropicMaterial[nPos_]
                       ]
              ] :=
Module[ {iP, eta, deltaD, n1, n2, KR, KT, ERT, rays},

        (* calculate intersection position *)
        iP = surfaceDistanceAndIntersectionPosition[
               P, Rho, eqn, condition][[2]];

        (* calculate unit vector normal to discontinuity
           surface at iP
         *)
        eta = normalToSurface[ eqn, iP ];

        (* increase cumulated optical path;
           see equation (60) in [McClain] *)
        deltaD = n*norm[P-iP]*(K.Rho);

        (* find out which direction the ray is going
           through the surface *)
        If[ eta.Rho > 0,

            (* ray is coming from the negative gradient
               side of the surface *)
            n1 = nNeg;
            n2 = nPos,

            (* ray is coming from pos. grad side *)
            n1 = nPos;
            n2 = nNeg
          ];

        (* make K and eta point into the same half
           space *)
        eta = eta Sign[eta.K];

        (* calculate the directions of the reflected
           and the transmitted rays *)
        (*********************************
         works only for isotropic media !!!
         *********************************)
        KR = K - 2 eta (K.eta);    (* K vector of
                                      reflected ray *)
        KT = snellsLaw[K, n1, eta, n2];

        If[ KT === evanescent,

            (* only temporary; I'll have to see how large
               the amplitude of the reflected ray is if
               'total reflection' occurs *)
            Message[rayTrace::evanescent];
            ERT = {{0,0,0},{0,0,0}},

            (* calculate E-vectors for the refracted
               and transmitted rays *)
            ERT = Fresnel[ n1, K, E, KR, eta, n2, KT ]
          ];

        (* prepare return value *)
```

```
            rays = {};
            If[ norm[ERT[[1]]] >= darkModE,

                rays = N[ Append[ rays,
                                  ray[ iP, KR, KR, n1,
                                       d+deltaD,
                                       Append[i, iP],
                                       ERT[[1]], od
                                     ]
                                ]
                        ],
                rays = Append[ rays,
                               ray[ dark, Append[i, iP] ]
                             ]
              ];
            If[ norm[ERT[[2]]] >= darkModE,

                rays = N[ Append[ rays,
                                  ray[ iP, KT, KT, n2,
                                       d+deltaD,
                                       Append[i, iP],
                                       ERT[[2]], od
                                     ]
                                ]
                        ],
                 rays = Append[ rays,
                                ray[ dark, Append[i, iP] ]
                              ]
              ];

            If[ Length[rays] == 1,
                rays[[1]],
                rays
              ]
          ];
```

axial ray trace: ray hits optical system

normal to surface

```
normalToSurface[ (* equation of surface in x, y, z *)
                 l_ == r_,
                 (* vector to point on surface *)
                 {xS_, yS_, zS_}
               ] :=
  Module[ {f},
          f = (l - r);  (* equation of surface
                           becomes f == 0 *)
          (* calculate the gradient of f *)
          Normalize[ { D[f, x], D[f, y], D[f, z] }
                     /. { x -> xS, y -> yS, z -> zS}
                   ]
        ];
```

curved surfaces

```
(* picks out the solution which describes the given vector *)
pickSolution[ {s_, r___}, cond_ ] :=
  Module[ {},
          If[ cond /. s,
              s,
              pickSolution[ {r}, cond ]
            ]
        ];
pickSolution[ {}, _ ] :=
  Module[ {},
          Message[ pickSolution::noSolution ];
```

```
        {}
      ];

ray hits flat surface
(* ray hits a discontinuity surface;
   calculate intersection point and trace further
 *)
aRayTrace[ ray[ {xP_, yP_, zP_},          (* starting point *)
                {xRho_, yRho_, zRho_},    (* ray direction *)
              K_,                         (* wave direction *)
              n_,                         (* refractive index in
                                             wave direction
                                          *)
              d_,                         (* cumulated optical
                                             path
                                          *)
              i_                          (* intersection points
                                             so far
                                          *)
         ],
       { surface[sEquation_],
         rest1___          (* rest of the optical system *)
       },
       rest2___
     ] :=
  Module[ {p, k, eta, deltaD},

       (* calculate intersection between ray and
          discontinuity surface
        *)
       p = ( {x, y, z} /. Solve[ { xP + k * xRho == x,
                                   yP + k * yRho == y,
                                   zP + k * zRho == z,
                                   sEquation
                                   (* equation in x, y, z *)
                                 }
                               ]
           )[[1]];

       (* calculate unit vector normal to discontinuity
          surface at p
        *)
       eta = normalToSurface[ sEquation, p ];

       (* increase cumulated optical path *)
       deltaD = n*norm[{xP, yP, zP}-p]*(K.{xRho, yRho, zRho});
              (* see equation (60) in [McClain] *)
       If[ ((p-{xP, yP, zP}).K)<0,

           Message[rayTrace::backwards];
           aRayTrace[ intersection[$Failed, i],
                      {rest1}, rest2
                    ],

           (* perform next step *)
           aRayTrace[ intersection[ p,       (* point of
                                               intersection *)
                                    eta,     (* normal to
                                               surface *)
                                    K, (* wave direction *)
                                    n, (* refractive index *)
                                    d+deltaD,
                                       (* cumulated optical
```

```
                                                    path *)
                                                Append[i, p]
                                                   (* intersections *)
                                             ] // N,
                                {rest1},
                                rest2
                             ]
                ]
        ];
ray hits curved surface
aRayTrace[ ray[ {xP_, yP_, zP_},          (* starting point *)
                {xRho_, yRho_, zRho_},    (* ray direction *)
                K_,             (* wave direction *)
                n_,             (* refractive index in
                                   wave direction *)
                d_,             (* cumulated optical path *)
                i_              (* intersection points *)
              ],
           { surface[sEquation_, cond_],
             rest1___           (* rest of the optical system *)
           },
           rest2___
         ] :=
  Module[ {p, k, eta, deltaD},

          p = {x, y, z} /.
                pickSolution[
                    Solve[ { xP + k * xRho == x,
                             yP + k * yRho == y,
                             zP + k * zRho == z,
                             sEquation
                           }
                         ],
                    cond
                  ];

          If[ p === {},

              Message[rayTrace::noIntersection];
              aRayTrace[ intersection[$Failed, i],
                         {rest1}, rest2
                       ],

              (* calculate unit vector normal to discontinuity
                 surface at p
               *)
              eta = normalToSurface[ sEquation, p ];

              (* increase cumulated optical path *)
              deltaD = n*norm[{xP, yP, zP}-p]*(K.{xRho, yRho,
zRho});
                    (* see equation (60) in [McClain] *)
              If[ ((p-{xP, yP, zP}).K)<0,

                  Message[rayTrace::backwards];
                  aRayTrace[ intersection[$Failed, i],
                             {rest1}, rest2
                           ],

                  (* perform next step *)
                  aRayTrace[
                      intersection[ p,     (* point of
                                              intersection *)
```

```
                                        eta,  (* normal to
                                                 surface *)
                                        K,    (* wave direction *)
                                        n,    (* refractive index *)
                                        d+deltaD,   (* cumulated
                                                       optical
                                                       path *)
                                        Append[i, p]
                                          (* intersections *)
                                      ] // N,
                                  {rest1},
                                  rest2
                               ]
                        ]
                    ]
            ];
deal with evanescent rays and rays which couldn't be computed
(* an exponentially decaying ray hits a surface;
   trace further to be able to pick special rays generated
   by ray doubling out by rayTrace[...][[(o or e)]],
   where o=1 and e=2 pick out the result of what became
   ordinary or extraordinary ray after hitting a birefringent
   medium respectively
 *)
aRayTrace[ ray[ evanescent, i_ ],
           { surface[ __ ], rest1___ },
           rest2___
         ] :=
  aRayTrace[ intersection[ evanescent, i ],
             { rest1 }, rest2 ];

aRayTrace[ ray[ $Failed, i_ ],
           { surface[ __ ], rest1___ },
           rest2___
         ] :=
  aRayTrace[ intersection[ $Failed, i ],
             { rest1 }, rest2 ];
ray hits material
(* ray hits material: ignore *)
aRayTrace[ ray[a__],
           {birefringentMaterial[__], rest1___},
           rest2___
         ] :=
  Module[ {},
          Message[rayTrace::rayHitsMaterial];
          aRayTrace[ ray[a], {rest1}, rest2 ]
        ];

aRayTrace[ ray[a__],
           {isotropicMaterial[_], rest1___},
           rest2___
         ] :=
  Module[ {},
          Message[rayTrace::rayHitsMaterial];
          aRayTrace[ ray[a], {rest1}, rest2 ]
        ];
axial ray trace: intersection 'hits' optical system
isotropic material follows
(* isotropic material follows after intersection;
   calculate new ray characteristics and trace further
 *)
aRayTrace[ intersection[ p_,
                         eta_,
```

```
                        ki_,
                        ni_,
                        d_,
                        i_
                      ],
          { isotropicMaterial[ n_ ],
            rest1___
          },
          rest2___
        ] :=
  aRayTrace[ toIsotropic[ ki, ni, d, i, p, eta, n ] // N,
            {rest1}, rest2
          ];
birefringent material follows
(* birefringent material follows after intersection;
   calculate new ray characteristics and trace further
 *)
aRayTrace[ intersection[ p_,
                        eta_,
                        ki_,
                        ni_,
                        d_,
                        i_
                      ],
          { birefringentMaterial[ no_,     (* ordinary refractive
                                              index
                                           *)
                                  ne_,     (* extraordinary
                                              refractive index
                                           *)
                                  c_       (* direction of crystal
                                              axis
                                           *)
                                ],
            rest1___
          },
          {o, rest2___},
          rest3___
        ] :=
  aRayTrace[ toMonoaxialO[ ki, ni, d, i, p, eta, no, ne, c ] // N,
             {rest1}, {rest2}, rest3
           ];
aRayTrace[ intersection[ p_,
                        eta_,
                        ki_,
                        ni_,
                        d_,
                        i_
                      ],
          { birefringentMaterial[ no_,     (* ordinary refractive
                                              index
                                           *)
                                  ne_,     (* extraordinary
                                              refractive index
                                           *)
                                  c_       (* direction of crystal
                                              axis
                                           *)
                                ],
            rest1___
          },
          {e, rest2___},
          rest3___
```

```
        ] :=
  aRayTrace[ toMonoaxialE[ ki, ni, d, i, p, eta, no, ne, c ] // N,
             {rest1}, {rest2}, rest3
           ];
aRayTrace[ intersection[ p_,
                         eta_,
                         ki_,
                         ni_,
                         d_,
                         i_
                       ],
           { birefringentMaterial[ no_,     (* ordinary refractive
                                               index
                                            *)
                                   ne_,     (* extraordinary
                                               refractive index
                                            *)
                                   c_       (* direction of crystal
                                               axis
                                            *)
                                 ],
             rest1___
           },
           {b, rest2___},
           rest3___
         ] :=
  aRayTrace[ toMonoaxial[ ki, ni, d, i, p, eta, no, ne, c ] // N,
             {rest1}, {rest2}, rest3
           ];
aRayTrace[ intersection[ p_,
                         eta_,
                         ki_,
                         ni_,
                         d_,
                         i_
                       ],
           { birefringentMaterial[ no_,     (* ordinary refractive
                                               index
                                            *)
                                   ne_,     (* extraordinary
                                               refractive index
                                            *)
                                   c_       (* direction of crystal
                                               axis
                                          *)
                                 ],
             rest1___
           },
           {},
           rest3___
         ] :=
  aRayTrace[ toMonoaxial[ ki, ni, d, i, p, eta, no, ne, c ] // N,
             {rest1}, {}, rest3
           ];
deal with intersections from evanescent rays and failed
computations
aRayTrace[ intersection[ a_, i_ ],
           { birefringentMaterial[ ___ ], rest1___},
           {o, rest2___}, rest3___
         ] :=
  aRayTrace[ ray[ a, i ],
             {rest1}, {rest2}, rest3 ];
aRayTrace[ intersection[ a_, i_ ],
```

```
          { birefringentMaterial[ ___ ], rest1___},
          {e, rest2___}, rest3___
        ] :=
  aRayTrace[ ray[ a, i ],
             {rest1}, {rest2}, rest3 ];
aRayTrace[ intersection[ a_, i_ ],
          { birefringentMaterial[ ___ ], rest1___},
          {b, rest2___}, rest3___
        ] :=
  aRayTrace[ { ray[ a, i ],
               ray[ a, i ] },
             {rest1}, {rest2}, rest3 ];
aRayTrace[ intersection[ a_, i_ ],
          { birefringentMaterial[ ___ ], rest1___},
          {}, rest2___
        ] :=
  aRayTrace[ { ray[ a, i ],
               ray[ a, i ] },
             {rest1}, {}, rest2 ];
aRayTrace[ intersection[ a_, i_ ],
          { isotropicMaterial[ ___ ], rest1___},
          rest2___
        ] :=
  aRayTrace[ ray[ a, i ], {rest1}, rest2 ];
axial ray trace finished
(* trace finished, nothing more to follow *)
aRayTrace[ something_, {}, __ ] := something;
```

non-axial ray trace

```
nothing to hit (finished)
nRayTrace[ r_, {}, ___ ] := r;
(* no objects to trace through *)
start with ray
nRayTrace[ ray[ P_,      (* starting point *)
                Rho_,    (* ray direction *)
                od___    (* other data *)
              ],
           surfaces_,
           ___
         ] :=
  Module[ {d, dMin, s, sNearest, i},

          (* find nearest surface that is hit *)
          dMin = Infinity;
          For[ i = Length[surfaces], i > 0, i = i-1,

               s = surfaces[[i]];
               d = findSurfaceDistance[ P, Rho, s ];
               If[ (d =!= noHits) && (d < dMin),

                   (* d is nearer than previous nearest
                      found surface *)
                   dMin = d;
                   sNearest = s
                 ]
             ];

          If[ dMin < Infinity,

              (* a surface was hit, trace through it
                 and then trace on *)
              nRayTrace[ traceThroughSurface[ ray[P, Rho, od],
                                              sNearest
```

```
                                                    ],
                        surfaces
                      ],

              (* no hits *)
              ray[ P, Rho, od ]      (* return unaltered ray *)
            ]
        ];
deal with evanescent & dark rays
nRayTrace[ ray[ evanescent, i___ ], ___ ] :=
  ray[ evanescent, i ];
nRayTrace[ ray[ dark, i___ ], ___ ] :=
  ray[ dark, i ];
more than one impinging ray
(* more than one ray hits the optical system *)
aRayTrace[ {l_, rest1___}, system_, rest2__ ] :=
  Join[ { aRayTrace[ l, system, rest2 ]},
          aRayTrace[ {rest1}, system, rest2 ] ];
(* more than one ray hits the optical system *)
nRayTrace[ {l_, r___}, system_ ] :=
  Join[ {nRayTrace[ l, system ]}, nRayTrace[ {r}, system ] ];
nothing hits the optical system
(* nothing at all happens *)
aRayTrace[ {}, __ ] := {};
(* nothing at all happens *)
nRayTrace[ {}, _ ] := {};
```

high level rayTrace routine

```
(* this routine normalises all vectors which are supposed to be
   unit vectors by aRayTrace[...], to which it passes the
   arguments on
 *)
rayTrace[ incoming_, system_, options___ ] :=
  Module[ {tr, pc, args},

          tr = Trace /.
                 {options} /. Options[rayTrace];
          pc = Component /.
                 {options} /. Options[rayTrace];
          darkModE = DarkE /.
                 {options} /. Options[rayTrace];

          args = {
              incoming /.
                  { ray[ p_,          (* starting point *)
                         rho_,        (* ray direction *)
                         k_,          (* wave direction *)
                         n_,          (* refractive index in
                                         wave direction
                                      *)
                         d_,          (* cumulated optical
                                         path
                                      *)
                         i_           (* intersection points *)
                       ] ->
                    ray[ p,
                         Normalize[rho],
                         Normalize[k],
                         n, d, i
                       ],
                    intersection[ p_,
                                  eta_,
                                  ki_,
```

```
                                    ni_,
                                    d_,
                                    i_
                                  ] ->
                      intersection[ p,
                                    Normalize[eta],
                                    Normalize[ki],
                                    ni, d, i
                                  ]
                    } // N,
                system /.
                    ( birefringentMaterial[ no_, ne_, z3_ ] ->
                      birefringentMaterial[ no, ne,
                                            Normalize[z3]
                                          ]
                    ) // N
              };
              args = Join[ args,
                           If[ tr === Axial,
                               {pc, options},
                               {options}
                             ]
                         ];

              If[ tr === Axial,
                  aRayTrace @@ args,
                  nRayTrace @@ args
                ]
            ];
rayInAir function
rayInAir[ P_,      (* starting point *)
          Rho_     (* ray direction *)
          ] :=
  ray[ P,          (* starting point *)
       Rho,        (* ray direction *)
       Rho,        (* wave direction, same as ray direction
                      (isotropic medium)
                   *)
       1,          (* refractive index *)
       0,          (* cumulation of optical path starts at P *)
       {P}         (* list of intersection points *)
     ] // N;
rayInAir[ P_,      (* starting point *)
          Rho_,    (* ray direction *)
          E_       (* E vector *)
          ] :=
  ray[ P,          (* starting point *)
       Rho,        (* ray direction *)
       Rho,        (* wave direction, same as ray direction
                      (isotropic medium)
                   *)
       1,          (* refractive index *)
       0,          (* cumulation of optical path starts at P *)
       {P},        (* list of intersection points *)
       E
     ] // N;
```

drawing package

```
updateDrawExtend[ {{x1_, y1_, z1_}, r___} ] :=
  Module[ {},

          If[ drawRange[[1]] === Automatic,
```

```
            If[ drawExtend[[1,1]] > x1,
                drawExtend[[1,1]] = x1
              ];
            If[ drawExtend[[1,2]] < x1,
                drawExtend[[1,2]] = x1
              ]
          ];
        If[ drawRange[[2]] === Automatic,

            If[ drawExtend[[2,1]] > y1,
                drawExtend[[2,1]] = y1
              ];
            If[ drawExtend[[2,2]] < y1,
                drawExtend[[2,2]] = y1
              ]
          ];
        If[ drawRange[[3]] === Automatic,

            If[ drawExtend[[3,1]] > z1,
                drawExtend[[3,1]] = z1
              ];
            If[ drawExtend[[3,2]] < z1,
                drawExtend[[3,2]] = z1
              ]
          ];

        updateDrawExtend[{r}]
      ];

updateDrawExtend[ Graphics3D[l_, r__] ] :=
  ( updateDrawExtend[ Graphics3D[l] ];
    updateDrawExtend[ Graphics3D[r] ]
  );

updateDrawExtend[ Graphics3D[{l_, r___}] ] :=
  ( updateDrawExtend[ Graphics3D[l] ];
    updateDrawExtend[ Graphics3D[r] ]
  );

updateDrawExtend[ Graphics3D[ Polygon[points_] ] ] :=
  updateDrawExtend[ points ];

updateDrawExtend[ Graphics3D[ Line[points_] ] ] :=
  updateDrawExtend[ points ];

updateDrawExtend[ Graphics3D[ _ ] ] := 0;

updateDrawExtend[{}] := drawExtend;

drawLines[ points_ ] :=
  ( updateDrawExtend[points];
    Line[points]
  );

drawRays[ {ray[_, _, _, _, _, i_, ___], r___} ] :=
  Append[ drawRays[{r}], drawLines[i] ];

drawRays[ {ray[ _, i_], r___} ] :=
  Append[ drawRays[{r}], drawLines[i] ];

drawRays[ {intersection[_, _, _, _, _, i_], r___} ] :=
  Append[ drawRays[{r}], drawLines[i] ];
```

```
drawRays[ {intersection[ _, i_ ], r___} ] :=
  Append[ drawRays[{r}], drawLines[i] ];

drawRays[ {isotropicMaterial[ __ ], r___} ] :=
  drawRays[{r}];

drawRays[ {birefringentMaterial[ __ ], r___} ] :=
  drawRays[{r}];

drawRays[ {surface[ ___ ], r___} ] :=
  drawRays[{r}];

drawRays[ {Graphics3D[ ___ ], r___} ] :=
  drawRays[{r}];

drawRays[ {{l___}, r___} ] :=
  drawRays[ {l, r} ];

drawRays[{}] := {};

drawSurfaces[ {ray[ ___ ], r___}, o___ ] :=
  drawSurfaces[{r}, o];

drawSurfaces[ {intersection[ ___ ], r___}, o___ ] :=
  drawSurfaces[{r}, o];

drawSurfaces[ {surface[ equation_ ], r___}, co_, {po___} ] :=
  Prepend[ drawSurfaces[{r}, co, {po}],
           Plot3D[ z /. Solve[ equation, z ][[1]],
                   Join[{x}, drawExtend[[1]]] // Evaluate,
                   Join[{y}, drawExtend[[2]]] // Evaluate,
                   po,
                   DisplayFunction :> Identity,
                   PlotPoints -> {2, 2},
                   Mesh-> False
                 ]
         ];

drawSurfaces[ {surface[ l__ == r__, cond_, ___ ], rest___},
              {co___}, po_ ] :=
  Prepend[ drawSurfaces[{rest}, {co}, po],
           ContourPlot3D[ If[ cond,
                              l-r,
                              0.1
                            ],
                          Join[{x}, drawExtend[[1]]] // Evaluate,
                          Join[{y}, drawExtend[[2]]] // Evaluate,
                          Join[{z}, drawExtend[[3]]] // Evaluate,
                          co,
                          Contours->{0.},
                          DisplayFunction :> Identity
                        ]
         ];

drawSurfaces[ {isotropicMaterial[ __ ], r___}, o___ ] :=
  drawSurfaces[{r}, o];

drawSurfaces[ {birefringentMaterial[ __ ], r___}, o___ ] :=
  drawSurfaces[{r}, o];

drawSurfaces[ {Graphics3D[ __ ], r___}, o___ ] :=
  drawSurfaces[{r}, o];
```

```
drawSurfaces[ {{l___}, r___}, o___ ] :=
  drawSurfaces[ {l, r}, o ];

drawSurfaces[{}, ___] := {};

drawGraphics3D[{Graphics3D[a___], rest___}] :=
  ( updateDrawExtend[Graphics3D[a]];
    Prepend[ drawGraphics3D[{rest}],
             Graphics3D[a] ]
  );

drawGraphics3D[{ray[___], rest___}] :=
  drawGraphics3D[{rest}];

drawGraphics3D[{intersection[___], rest___}] :=
  drawGraphics3D[{rest}];

drawGraphics3D[{surface[___], rest___}] :=
  drawGraphics3D[{rest}];

drawGraphics3D[{isotropicMaterial[___], rest___}] :=
  drawGraphics3D[{rest}];

drawGraphics3D[{birefringentMaterial[___], rest___}] :=
  drawGraphics3D[{rest}];

drawGraphics3D[{{l___}, r___}] :=
  drawGraphics3D[{l, r}];

drawGraphics3D[{}] := {};

draw[ {a___}, o___ ] :=
  Module[ { s, sXYRange, co, po, go, so, dr },

          (* Extract the Options *)
          s = Options[draw];  (* standard options *)
          drawRange = Range  /.{o}/.s;
          sXYRange = StandardRangeWidth /.{o}/.s;
          co = Select[ Join[{o}, s],
                       ( MemberQ[ Options[ContourPlot3D],
                                  #[[1]] -> _
                                ] )&
                     ];
          po = Select[ Join[{o}, s],
                       ( MemberQ[ Options[Plot3D],
                                  #[[1]] -> _
                                ] )&
                     ];
          go = Select[ Join[{o}, s],
                       ( MemberQ[ Options[Graphics3D],
                                  #[[1]] -> _
                                ] )&
                     ];

          so = Sequence @@
                    ( Select[ Join[{o}, s],
                              ( MemberQ[ Options[Show],
                                         #[[1]] -> _
                                       ] )&
                            ]
                    );
          (* define drawExtend, which is updated
             during the execution of drawRays *)
```

```
            drawExtend = drawRange
                /. Automatic -> {Infinity, -Infinity};

            (* update drawExtend *)
            dr = Join[ { Graphics3D[ drawRays[{a}], go ] },
                       drawGraphics3D[{a}]
                     ];
            (* if no rays are drawn, draw in standard
               range *)
            drawExtend = drawExtend
                /. {Infinity, -Infinity} -> sXYRange;

            (* if width of extend is zero, draw in
               standard range *)
            drawExtend = drawExtend
                           /.{b_, b_}->({b,b}+sXYRange);
            Show[ Join[ dr, drawSurfaces[{a}, co, po] ],
                  so,
                  DisplayFunction :> $DisplayFunction
                ]
        ];
End[];
```

units & materials

units

```
m = 1;       (* all lengths in meters *)
mm = 10^-3 m;
cm = 10 mm;
um = 10^-3 mm;
nm = 10^-6 mm;
```

Calcite

define nO[Calcite, <lambda>] and nE[Calcite, <lambda>]

```
(* refractive indices of Calcite;
   from G W C Kaye, T H Laby, "Tables of Physical
   and Chemical Constants" *)
nOCalciteTable = { { 200 nm, 1.90284},
                   { 303 nm, 1.71959},
                   { 410 nm, 1.68014},
                   { 508 nm, 1.66527},
                   { 643 nm, 1.65504},
                   { 706 nm, 1.65207},
                   { 801 nm, 1.64869},
                   { 905 nm, 1.64578},
                   {1042 nm, 1.64276},
                   {1159 nm, 1.64051},
                   {1229 nm, 1.63926},
                   {1307 nm, 1.63789},
                   {1396 nm, 1.63637} };
nECalciteTable = { { 200 nm, 1.57649},
                   { 303 nm, 1.51365},
                   { 410 nm, 1.49640},
                   { 508 nm, 1.48956},
                   { 643 nm, 1.48490},
                   { 706 nm, 1.48353},
                   { 801 nm, 1.48216},
                   { 905 nm, 1.48098},
                   {1042 nm, 1.47985},
                   {1159 nm, 1.47910},
                   {1229 nm, 1.47870},
                   {1307 nm, 1.47831},
                   {1396 nm, 1.47789} };
```

```
(* interpolate *)
nO[Calcite, lambda_] =
  Interpolation[ nOCalciteTable ][lambda];
nE[Calcite, lambda_] =
  Interpolation[ nECalciteTable ][lambda];
define Calcite[ <lambda>, <crystal axis>]
Calcite[ l_, cr_ ] :=
  birefringentMaterial[ nO[Calcite, l], nE[Calcite, l], cr ];
(* crystal axis parallel to x-axis *)
Calcite[ l_, x ] := Calcite[ l, {1, 0, 0} ];
(* crystal axis parallel to y-axis *)
Calcite[ l_, y ] := Calcite[ l, {0, 1, 0} ];
(* crystal axis parallel to z-axis *)
Calcite[ l_, z ] := Calcite[ l, {0, 0, 1} ];
```

Lithium Niobate

```
define nO[LiNbO3, <lambda>] and nE[LiNbO3, <lambda>]
(* from: E D Palik, "Handbook of Optical Constants
   of Solids" and W G Driscoll, W Vaughan,
   "Handbook of Optics" *)
nOLiNbO3Table = { { 450 nm, 2.3780},
                  { 500 nm, 2.3410},
                  { 508 nm, 2.3356},
                  { 600 nm, 2.2967},
                  { 700 nm, 2.2716},
                  { 800 nm, 2.2571},
                  { 1000 nm, 2.2370} };
nELiNbO3Table = { { 450 nm, 2.2772},
                  { 500 nm, 2.2457},
                  { 508 nm, 2.2448},
                  { 600 nm, 2.2082},
                  { 700 nm, 2.1874},
                  { 800 nm, 2.1745},
                  { 1000 nm, 2.1567} };
(* interpolate *)
nO[LiNbO3, lambda_] =
  Interpolation[ nOLiNbO3Table ][lambda];
nE[LiNbO3, lambda_] =
  Interpolation[ nELiNbO3Table ][lambda];
define LiNbO3[ <lambda>, <crystal axis>]
LiNbO3[ l_, cr_ ] :=
  birefringentMaterial[ nO[LiNbO3, l],
                        nE[LiNbO3, l], cr ];
(* crystal axis parallel to x-axis *)
LiNbO3[ l_, x ] := LiNbO3[ l, {1, 0, 0} ];
(* crystal axis parallel to y-axis *)
LiNbO3[ l_, y ] := LiNbO3[ l, {0, 1, 0} ];
(* crystal axis parallel to z-axis *)
LiNbO3[ l_, z ] := LiNbO3[ l, {0, 0, 1} ];
```

Quartz

```
define nO[Quartz, <lambda>] and nE[Quartz, <lambda>]
(* From W G Driscoll, W Vaughan,
   "Handbook of Optics" *)
nOQuartzTable = { { 185 nm, 1.67578},
                  { 198 nm, 1.65087},
                  { 231 nm, 1.61395},
                  { 340 nm, 1.56747},
                  { 394 nm, 1.55846},
                  { 434 nm, 1.55396},
                  { 508 nm, 1.54822},
                  { 589.3 nm, 1.54424},
                  { 768 nm, 1.53903},
                  { 832.5 nm, 1.53773},
```

```
                    { 991.4 nm, 1.53514},
                    { 1159.2 nm, 1.53283},
                    { 1307 nm, 1.53090},
                    { 1395 nm, 1.52977},
                    { 1479.2 nm, 1.52865} };
nEQuartzTable = { { 185 nm, 1.68988},
                    { 198 nm, 1.66394},
                    { 231 nm, 1.62555},
                    { 340 nm, 1.57737},
                    { 394 nm, 1.56805},
                    { 434 nm, 1.56338},
                    { 508 nm, 1.55746},
                    { 589.3 nm, 1.55335},
                    { 768 nm, 1.54794},
                    { 832.5 nm, 1.54661},
                    { 991.4 nm, 1.54392},
                    { 1159.2 nm, 1.54152},
                    { 1307 nm, 1.53951},
                    { 1395 nm, 1.53832},
                    { 1479.2 nm, 1.53716} };
(* interpolate *)
nO[Quartz, lambda_] =
  Interpolation[ nOQuartzTable ][lambda];
nE[Quartz, lambda_] =
  Interpolation[ nEQuartzTable ][lambda];
define Quartz[ <lambda>, <crystal axis>]
Quartz[ l_, cr_ ] :=
  birefringentMaterial[ nO[Quartz, l],
                        nE[Quartz, l], cr ];
(* crystal axis parallel to x-axis *)
Quartz[ l_, x ] := Quartz[ l, {1, 0, 0} ];
(* crystal axis parallel to y-axis *)
Quartz[ l_, y ] := Quartz[ l, {0, 1, 0} ];
(* crystal axis parallel to z-axis *)
Quartz[ l_, z ] := Quartz[ l, {0, 0, 1} ];
```

Rutile

```
define nO[Rutile, <lambda>] and nE[Rutile, <lambda>]
(* refractive indices of Rutile;
   from W G Driscoll, W Vaughan,
   "Handbook of Optics" *)
nORutileTable = { { 435.8 nm, 2.853},
                    { 491.6 nm, 2.725},
                    { 496 nm, 2.718},
                    { 546.1 nm, 2.652},
                    { 577 nm, 2.623},
                    { 579.1 nm, 2.621},
                    { 690.7 nm, 2.555},
                    { 708.2 nm, 2.548},
                    {1014 nm, 2.484},
                    {1529.6 nm, 2.454} };
nERutileTable = { { 435.8 nm, 3.216},
                    { 491.6 nm, 3.051},
                    { 496 nm, 3.042},
                    { 546.1 nm, 2.958},
                    { 577 nm, 2.921},
                    { 579.1 nm, 2.919},
                    { 690.7 nm, 2.836},
                    { 708.2 nm, 2.826},
                    {1014 nm, 2.747},
                    {1529.6 nm, 2.710} };
(* interpolate *)
nO[Rutile, lambda_] =
```

```
  Interpolation[ nORutileTable ][lambda];
nE[Rutile, lambda_] =
  Interpolation[ nERutileTable ][lambda];
define Rutile[ <lambda>, <crystal axis>]
Rutile[ l_, cr_ ] :=
  birefringentMaterial[ nO[Rutile, l], nE[Rutile, l], cr ];
(* crystal axis parallel to x-axis *)
Rutile[ l_, x ] := Rutile[ l, {1, 0, 0} ];
(* crystal axis parallel to y-axis *)
Rutile[ l_, y ] := Rutile[ l, {0, 1, 0} ];
(* crystal axis parallel to z-axis *)
Rutile[ l_, z ] := Rutile[ l, {0, 0, 1} ];
```

ADP

```
define nO[ADP, <lambda>] and nE[ADP, <lambda>]
(* refractive indices of ADP;
   from W G Driscoll, W Vaughan,
   "Handbook of Optics" *)
nOADPTable = {    { 404.7 nm, 1.53969},
                  { 407.7 nm, 1.53925},
                  { 435.8 nm, 1.53578},
                  { 546.1 nm, 1.52662},
                  { 576.9 nm, 1.52478},
                  { 579.1 nm, 1.52466},
                  { 632.8 nm, 1.52166},
                  {1013.9 nm, 1.50835},
                  {1128.7 nm, 1.50446},
                  {1152.3 nm, 1.50364},
                  {200 nm, 1.648},
                  {300 nm, 1.563},
                  {400 nm, 1.540},
                  {500 nm, 1.530},
                  {600 nm, 1.524},
                  {700 nm, 1.519},
                  {800 nm, 1.515},
                  {900 nm, 1.512},
                  {1000 nm, 1.509},
                  {1100 nm, 1.505},
                  {1200 nm, 1.502},
                  {1300 nm, 1.498},
                  {1400 nm, 1.495},
                  {1500 nm, 1.491} };
nEADPTable = { { 404.7 nm, 1.49159},
                  { 407.7 nm, 1.49123},
                  { 435.8 nm, 1.48831},
                  { 491.6 nm, 1.48390},
                  { 546.1 nm, 1.48079},
                  { 576.9 nm, 1.47939},
                  { 579.1 nm, 1.47930},
                  { 632.8 nm, 1.47685},
                  {1013.9 nm, 1.46895},
                  {1128.7 nm, 1.46704},
                  {1152.3 nm, 1.46666},
                  {200 nm, 1.587},
                  {300 nm, 1.512},
                  {400 nm, 1.492},
                  {500 nm, 1.483},
                  {600 nm, 1.478},
                  {700 nm, 1.475},
                  {800 nm, 1.473},
                  {900 nm, 1.471},
                  {1000 nm, 1.469},
                  {1100 nm, 1.467},
```

```
                    {1200 nm, 1.466},
                    {1300 nm, 1.464},
                    {1400 nm, 1.463},
                    {1500 nm, 1.461} };
(* interpolate *)
nO[ADP, lambda_] =
  Interpolation[ nOADPTable ][lambda];
nE[ADP, lambda_] =
  Interpolation[ nEADPTable ][lambda];
define ADP[ <lambda>, <crystal axis>]
ADP[ l_, cr_ ] :=
  birefringentMaterial[ nO[ADP, l], nE[ADP, l], cr ];
(* crystal axis parallel to x-axis *)
ADP[ l_, x ] := ADP[ l, {1, 0, 0} ];
(* crystal axis parallel to y-axis *)
ADP[ l_, y ] := ADP[ l, {0, 1, 0} ];
(* crystal axis parallel to z-axis *)
ADP[ l_, z ] := ADP[ l, {0, 0, 1} ];
```

KDP

```
define nO[KDP, <lambda>] and nE[KDP, <lambda>]
(* refractive indices of KDP;
   from W G Driscoll, W Vaughan,
   "Handbook of Optics" *)
nOKDPTable = { { 404.7 nm, 1.52341},
                    { 407.7 nm, 1.52301},
                    { 435.8 nm, 1.51990},
                    { 546.1 nm, 1.51152},
                    { 579.1 nm, 1.50977},
                    { 632.8 nm, 1.50737},
                    { 1013.9 nm, 1.49535},
                    { 1128.7 nm, 1.49205},
                    {1152.3 nm, 1.49135},
                    {200 nm, 1.621},
                    {300 nm, 1.545},
                    {400 nm, 1.524},
                    {500 nm, 1.514},
                    {600 nm, 1.509},
                    {700 nm, 1.505},
                    {800 nm, 1.502},
                    {900 nm, 1.499},
                    {1000 nm, 1.496},
                    {1100 nm, 1.493},
                    {1200 nm, 1.490},
                    {1300 nm, 1.487},
                    {1400 nm, 1.483},
                    {1500 nm, 1.480} };
nEKDPTable = { { 404.7 nm, 1.47927},
                    { 407.7 nm, 1.47898},
                    { 435.8 nm, 1.47640},
                    { 546.1 nm, 1.46982},
                    { 579.1 nm, 1.46856},
                    { 632.8 nm, 1.46685},
                    { 1013.9 nm, 1.46041},
                    { 1128.7 nm, 1.45917},
                    {1152.3 nm, 1.45893},
                    {200 nm, 1.563},
                    {300 nm, 1.498},
                    {400 nm, 1.480},
                    {500 nm, 1.472},
                    {600 nm, 1.468},
                    {700 nm, 1.465},
                    {800 nm, 1.463},
```

```
                    {900 nm, 1.462},
                    {1000 nm, 1.461},
                    {1100 nm, 1.459},
                    {1200 nm, 1.458},
                    {1300 nm, 1.457},
                    {1400 nm, 1.456},
                    {1500 nm, 1.455} };
(* interpolate *)
nO[KDP, lambda_] =
  Interpolation[ nOKDPTable ][lambda];
nE[KDP, lambda_] =
  Interpolation[ nEKDPTable ][lambda];
define KDP[ <lambda>, <crystal axis>]
KDP[ l_, cr_ ] :=
  birefringentMaterial[ nO[KDP, l], nE[KDP, l], cr ];
(* crystal axis parallel to x-axis *)
KDP[ l_, x ] := KDP[ l, {1, 0, 0} ];
(* crystal axis parallel to y-axis *)
KDP[ l_, y ] := KDP[ l, {0, 1, 0} ];
(* crystal axis parallel to z-axis *)
KDP[ l_, z ] := KDP[ l, {0, 0, 1} ];
```

air

```
air = isotropicMaterial[1];
```

A.2. Demonstration program

The following short *Mathematica* program demonstrates the application of the ray tracing package. For further information consult the *usages* section of the ray tracing program.

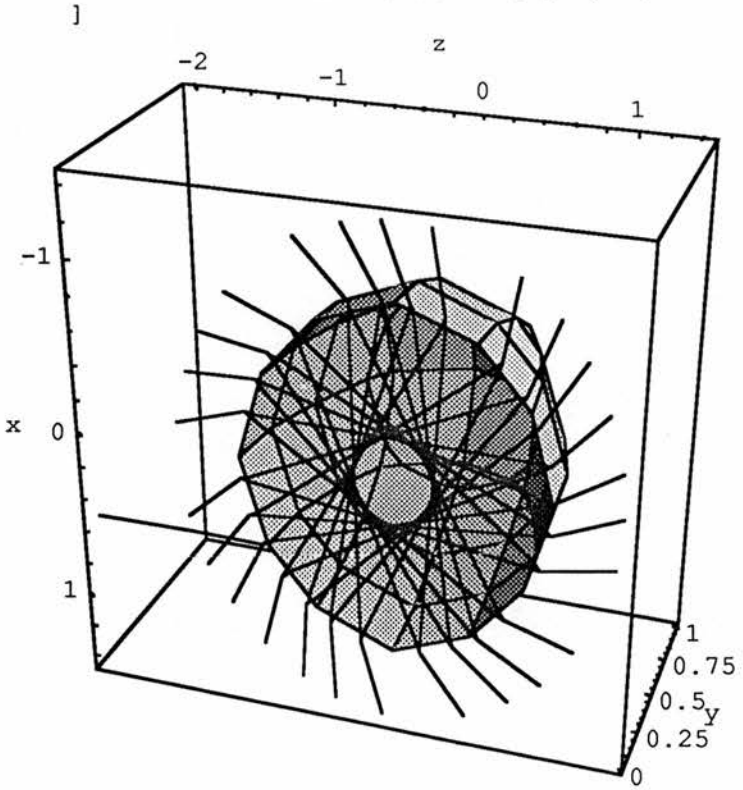
```
(* the ray tracing package has to be loaded *)

sphere = surface[ x^2+y^2+z^2==1, True,
                  isotropicMaterial[2],
                  isotropicMaterial[1]
                ];

largeSphere = surface[ x^2+y^2+z^2==2, True,
                       isotropicMaterial[1],
                       isotropicMaterial[1]
                     ];

(* the function rayTrace performs the actual calculations *)
r=rayTrace[ rayInAir[{0.5,0,-2},{0,0,1},{0,1,0}],
            {sphere, largeSphere},
            Trace->NonAxial,
            DarkE -> 10^-10
          ];

draw[{r, sphere}, Range -> {{-2,2},{0,2},{-2,2}},
                  BoxRatios -> {2, 1, 2}
    ]
```



B Application of the ray tracing package to Wollaston prism spectrometers

B.1. The spectrometer package

This *Mathematica* program uses the ray tracing package described in appendix A to supply functions which perform the calculations referred to in the main part of this work. The second part of this appendix contains a program that demonstrates its use.

```
inclinedRayInAir[ p_,    (* starting point *)
                  alpha_,     (* angle of inclination to x axis
                                 in x-z plane in rad *)
                  beta_       (* angle of inclination to x axis
                                 in y-z plane in rad*)
                ] :=
  rayInAir[ p,
            Normalize[ { Tan[alpha],
                         Tan[beta],
                         1 } ]
          ];
opticalPath[ray[_, _, _, _, p_, _]] := p;
opticalPath[intersection[_, _, _, _, p_, _]] := p;
startingPoint[ray[p_, _, __]] := p;
direction[ray[_, d_, __]] := d;
```

geometric tools

```
displace[ {dx_, dy_, dz_}, whatever_ ] :=
  whatever /. { x -> (x - dx),
                y -> (y - dy),
                z -> (z - dz) };
rayIntersection::noIntersection =
"warning: rays don't intersect; rayIntersection failed";
rayIntersection[ ray[ p1_, d1_, ___ ],
                 ray[ p2_, d2_, ___ ] ] :=
  Module[ {k, l, s},

          s = Solve[ p1 + k d1 == p2 + l d2 ];

          If[ Length[s] != 1,

              Message[rayIntersection::noIntersection];
              $Failed,

              p1 + k d1 /. s[[1]]
            ]
        ];
```

camera functions

```
setupCamera::failed =
"the neccessary ray trace has failed; setupCamera failed";
setupCamera::unknownOptionValue =
"unknown option value";
Options[ setupCamera ] =
```

```
  { position -> InFocus,
    pixels -> 1024,
    width -> 2.56 cm,
    spectrometerType -> doubleWollaston
  };
(* camera normal to optic axis *)
setupCamera[ spectrometer_, options___ ] :=
  Module[ { cPos, cPixels, cWidth, st, s1, s2,
            r11, r12, i1, r21, r22, i2, d },

          (* evaluate options *)
          cPos = position /.
                      {options} /.
                      Options[setupCamera];
          cPixels = pixels /.
                     {options} /.
                     Options[setupCamera];
          cWidth = width /.
                    {options} /.
                    Options[setupCamera];
          st = spectrometerType /.
                 {options} /.
                 Options[setupCamera];
          If[ st === doubleWollaston,

              s1 = {e,o,o,e};
              s2 = {o,e,e,o},

              If[ st === doubleWollastonWWP,

                  (* double Wollaston with wave plate *)
                  s1 = {e,o,e,o};
                  s2 = {o,e,o,e},

                  If[ st === singleWollaston,

                      s1 = {e,o};
                      s2 = {o,e}
                    ]
                ]
            ];

          If[ cPos === InFocus,


              (* setup camera in focal plane, defined
                 as the plane where the two
                 polarisation components of normally
                 incident rays intersect *)
              r11 = rayTrace[ rayInAir[{0,0,0},{0,0,1}],
                              Append[spectrometer, air],
                              Component->s1
                            ];
              r12 = rayTrace[ rayInAir[{0,0,0},{0,0,1}],
                              Append[spectrometer, air],
                              Component->s2
                            ];
              r21 = rayTrace[ rayInAir[{1 mm,0,0},{0,0,1}],
                              Append[spectrometer, air],
                              Component->s1
                            ];
              r22 = rayTrace[ rayInAir[{1 mm,0,0},{0,0,1}],
                              Append[spectrometer, air],
```

```
                                Component->s2
                              ];
                    If[ (Length[ r11 ] > 2) &&
                        (Length[ r12 ] > 2) &&
                        (Length[ r21 ] > 2) &&
                        (Length[ r22 ] > 2),

                        (* everything ok *)
                        i1 = rayIntersection[ r11, r12 ];
                        i2 = rayIntersection[ r21, r22 ];
                        d = Normalize[i2 - i1] cWidth;
                        camera[ i1-d/2.0,
                                N[d/(cPixels-1)],
                                cPixels ],

                        (* ray trace has failed *)
                        Message[setupCamera::failed];
                        camera[ $Failed ]
                      ],


                    If[ cPos === NormalToOpticalAxis,

                        (* setup camera in plane normal to
                           optic axis *)
                        r11 = rayTrace[ rayInAir[{0,0,0},{0,0,1}],
                                        Append[spectrometer, air],
                                        Component->s1
                                      ];
                        r12 = rayTrace[ rayInAir[{0,0,0},{0,0,1}],
                                        Append[spectrometer, air],
                                        Component->s2
                                      ];
                        If[ (Length[r11] > 2) &&
                            (Length[r12] > 2),

                            (* everything ok *)
                            i1 = rayIntersection[ r11, r12 ];
                            camera[ { -cWidth/2, 0, i1[[3]]},
                                    { cWidth / (cPixels - 1), 0, 0},
                                    cPixels
                                  ],

                            Message[setupCamera::failed];
                            camera[ $Failed ]
                          ],

                        Message[setupCamera::unknownOptionValue]
                      ]
                ]
            ];
inclination[ camera[ _, d_, _ ] ] :=
  (Pi/2 - ArcCos[ Normalize[d].{0,0,1} ]) //N;

noOfPixels[ camera[ _, _, n_ ] ] := n;
pixel[ camera[ p_, d_, _ ],
       n_    (* number of pixel element *)
     ] :=
  p + (n-1) d;
drawCamera[ camera[ pl_, d_, n_ ] ] :=
  Module[ {width},

          width = N[ Normalize[Cross[{0,0,1},d]] *
```

```
                    Sqrt[d.d] * n / 20 ];
          Graphics3D[Polygon[{ p1    -width, p1    +width,
                               p1+n*d+width, p1+n*d-width
                             }]
                    ]
        ];
```

path difference and Nyquist wavelength

```
pixelPD::cameraProblem =
"the camera is not set up properly; pixelPD failed";
pixelPD::rayTraceFailed =
"the neccessary ray trace failed; pixelPD failed";
pixelPD[ _, camera[ $Failed ], ___ ] :=
  Module[ {},

          Message[pixelPD::cameraProblem];
          $Failed
        ];
Options[pixelPD] =
  { whichPixel -> Centre,
    spectrometerType -> doubleWollaston,
    drawRays -> False };
pixelPD::usage =
"Options: whichPixel -> Centre\n"<>
"                       <pixel no>\n"<>
"         spectrometerType -> doubleWollaston\n"<>
"                             doubleWollastonWWP\n"<>
"                             singleWollaston\n"<>
"                               (* with wave plate *)";
(* optical path difference between parallel rays that meet
   in pixel element #n *)
pixelPD[ spectrometer_,
         camera_,
         alpha_,  (* angle of inclination to x axis
                     in x-y plane *)
         beta_,   (* angle of inclination to x axis
                     in x-z plane *)
         opt___
       ] :=
(* pixelPD[ spectrometer, camera, alpha, beta, opt ] = *)
  Module[ { n, st, dr, s1, s2, s1r, s2r,
            r1, r2, d1, d2, d },

          n = whichPixel /.{opt}/.Options[pixelPD]
                /.Centre->N[(noOfPixels[camera]+1)/2];
          st = spectrometerType/.{opt}/.Options[pixelPD];
          dr = drawRays /. {opt}/.Options[pixelPD];

          If[ st === doubleWollaston,

              s1 = {e,o,o,e};
              s2 = {o,e,e,o},

              If[ st === doubleWollastonWWP,

                  (* double Wollaston with wave plate *)
                  s1 = {e,o,e,o};
                  s2 = {o,e,o,e},

                  If[ st === singleWollaston,

                      s1 = {e,o};
                      s2 = {o,e}
```

```
          ]
      ]
  ];
s1r = Reverse[s1];
s2r = Reverse[s2];

(* find directions at which rays, that fall on to
   the spectrometer from the left, come out at the
   other end with the camera on
 *)
r1 = rayTrace[ inclinedRayInAir[ {0, 0, 0},
                                 alpha,
                                 beta
                               ],
               Append[ spectrometer, air ],
               Component->s1
             ];
r2 = rayTrace[ inclinedRayInAir[ {0, 0, 0},
                                 alpha,
                                 beta
                               ],
               Append[ spectrometer, air ],
               Component->s2
             ];
If[ (Length[ r1 ] > 2) &&
    (Length[ r2 ] > 2),

    (*** everything ok so far ***)
    d1 = direction[ r1 ];
    d2 = direction[ r2 ];

    (* trace rays emerging from the pixel element in
       the opposite directions
     *)
    r1 = rayTrace[ rayInAir[ pixel[camera, n],
                             -d1 ],
                   Append[Reverse[spectrometer], air],
                   Component->s1r
                 ] (* // Chop *);
    r2 = rayTrace[ rayInAir[ pixel[camera, n],
                             -d2 ],
                   Append[Reverse[spectrometer], air],
                   Component->s2r
                 ] (* // Chop *);

    (* draw setup and rays *)
    If[ dr === True,

        draw[ { spectrometer, r1, r2,
                drawCamera[camera] } ]
      ];

    If[ (Length[ r1 ] > 2) &&
        (Length[ r2 ] > 2),

        (** still everything ok **)
        d = startingPoint[r1] - startingPoint[r2];

        (* return path difference *)
        opticalPath[r1] - opticalPath[r2] -
          direction[r2].d,

        (** problem **)
```

```
                Message[pixelPD::rayTraceFailed];
                $Failed
              ],

             (*** same problem ***)
             Message[pixelPD::rayTraceFailed];
             $Failed
           ]
        ];
brightness[lambda_, pd_] :=
  N[ (Cos[Pi pd/lambda])^2 ];
pixelPDAndGrayLevel[lambda_, a__] :=
  Module[ {pd},

          pd = pixelPD[a];
          { pd,
            GrayLevel[ brightness[lambda, pd] ]
          }
        ];
NyquistLambda::failed =
"neccessary path difference calculation failed;
 calculation of Nyquist wavelength failed";
NyquistLambda::usage =
"Options are passed on to pixelPD";
NyquistLambda[ spectrometer_, camera_, opt___ ] :=
  (* returns the Nyquist wavelength for the setup *)
  Module[ {pdLCentre, pdRCentre},

          pdLCentre =
            pixelPD[ spectrometer, camera, 0, 0,
                     whichPixel -> noOfPixels[camera]/2-1,
                     opt
                   ];
          pdRCentre =
            pixelPD[ spectrometer, camera, 0, 0,
                     whichPixel -> noOfPixels[camera]/2,
                     opt
                   ];
          If[ (pdLCentre =!= $Failed) &&
              (pdRCentre =!= $Failed),

              N[ 2 Abs[ pdLCentre - pdRCentre ] ],

              Message[NyquistLambda::failed];
              $Failed
            ]
        ];
```

shear

```
Options[shear] =
  { whichPixel -> Centre,
    spectrometerType -> doubleWollaston};
shear::usage =
"shear[ <spectrometer>, <camera>, <alpha>, <beta>, "<>
"<options>] returns the separation between ray "<>
"components in orthogonal polarisation states "<>
"which impinge from the same direction (i.e. originate "<>
"from the same infinitely distant point source) "<>
"and intersect in the camera pixel chosen with the "<>
"option 'whichPixel'.\n"<>
"Options: whichPixel -> Centre\n"<>
"                       <pixel no>\n"<>
"         spectrometerType -> doubleWollaston\n"<>
```

```
"                              doubleWollastonWWP\n"<>
"                              singleWollaston\n"<>
"                                (* with wave plate *)";
shear::cameraProblem =
"the camera is not set up properly; shear failed";
shear::rayTraceFailed =
"the neccessary ray trace failed; shear failed";
shear[ _, camera[ $Failed ], ___ ] :=
  Module[ {},

          Message[shear::cameraProblem];
          $Failed
        ];
shear[ spectrometer_,
       camera_,
       alpha_,     (* angle of inclination to x axis
                      in x-y plane *)
       beta_,      (* angle of inclination to x axis
                      in x-z plane *)
       opt___
     ] :=
  shear[ spectrometer, camera, alpha, beta, opt ] =
  Module[ { n, st, s1, s2, s1r, s2r,
            r1, r2, d1, d2, d },

          n = whichPixel /.{opt}/.Options[shear]
                /.Centre->N[(noOfPixels[camera]-1)/2];
          st = spectrometerType/.{opt}/.Options[shear];

          If[ st === doubleWollaston,

              s1 = {e,o,o,e};
              s2 = {o,e,e,o},

              If[ st === doubleWollastonWWP,

                  (* double Wollaston with wave plate *)
                  s1 = {e,o,e,o};
                  s2 = {o,e,o,e},

                  If[ st === singleWollaston,

                      s1 = {e,o};
                      s2 = {o,e}
                    ]
                ]
            ];
          s1r = Reverse[s1];
          s2r = Reverse[s2];

          (* find directions at which rays, that fall on to
             the spectrometer from the left, come out at the
             other end with the camera on
           *)
          r1 = rayTrace[ inclinedRayInAir[ {0, 0, 0},
                                           alpha,
                                           beta
                                         ],
                         Append[ spectrometer, air ],
                         Component->s1
                       ];
          r2 = rayTrace[ inclinedRayInAir[ {0, 0, 0},
                                           alpha,
```

```
                                        beta
                                      ],
                        Append[ spectrometer, air ],
                        Component->s2
                      ];
          If[ (Length[ r1 ] > 2) &&
              (Length[ r2 ] > 2),

              (*** everything ok so far ***)
              d1 = direction[ r1 ];
              d2 = direction[ r2 ];

              (* trace rays emerging from the pixel element in
                 the opposite directions
               *)
              r1 = rayTrace[ rayInAir[ pixel[camera, n],
                                       -d1 ],
                             Append[Reverse[spectrometer], air],
                             Component->s1r
                           ] // Chop;
              r2 = rayTrace[ rayInAir[ pixel[camera, n],
                                       -d2 ],
                             Append[Reverse[spectrometer], air],
                             Component->s2r
                           ] // Chop;

              If[ (Length[ r1 ] > 2) &&
                  (Length[ r2 ] > 2),

                  (** still everything ok **)
                  d = startingPoint[r1] - startingPoint[r2];

                  (* return ray separation *)
                  Sqrt[d.d - (d.direction[r2])^2],

                  (** problem **)
                  Message[shear::rayTraceFailed];
                  $Failed
                ],

              (*** same problem ***)
              Message[shear::rayTraceFailed];
              $Failed
            ]
        ];
```

optical components

```
wedge[ material_, theta_, thickness_ ] :=
  { surface[ z == 0],
    material,
    surface[ (z-thickness) == x * Tan[theta] ]
  } // N;
wedge::usage =
"wedge[ <material>, <theta>, <thickness> ] \n
 -> <a list for use in rayTrace[...], which\n
     describes a wedge made from <material>,\n
     with one face lying in the plane z = 0\n
     and the other one being rotated by an angle\n
     theta around the y axis and shifted through\n
     <thickness> to the right>\n";
wollaston[ material1_,
           theta_,
           material2_,
```

```
          thickness_
        ] :=
  Join[ wedge[ material1, theta, thickness/2 ],
        { material2,
          surface[ z == thickness ]
        }
      ] // N;
(* for some reason it's advisable to have the smaller ends of
   the wedges at least 0.5 mm thick, so another 1 mm has to
   be added to the thickness
 *)
minimumWollastonThickness[theta_, height_] :=
  Module[ {t},
          t = N[ height Tan[ theta ] + 1 mm ];
          If[ t < height (4. / 27.),
              height (4. / 27.),
              t
            ]
        ];
wollaston[ material1_, theta_, material2_,
           height_, thickness->Minimum
         ] :=
  wollaston[ material1, theta, material2,
             minimumWollastonThickness[ theta, height ]
           ];
```

spectrometer

double Wollaston spectrometer

```
Options[doubleWollastonSpectrometer] =
  { materials->{ Calcite[633 nm, x], Calcite[633 nm, y],
                 Calcite[633 nm, y], Calcite[633 nm, x] },
    internalAngles->{1.3 Degree, 2.1 Degree},
    WollastonSeparation->3. mm,
    thicknesses->{4. mm, 4. mm}
  };
doubleWollastonSpectrometer[ o___ ] :=
  Module[ {so, mTable, iTable, dWollastons, tTable},

          (* evaluate options *)
          so = Options[doubleWollastonSpectrometer];
          mTable      = materials           /. {o} /. so;
          iTable      = internalAngles      /. {o} /. so;
          dWollastons = WollastonSeparation /. {o} /. so;
          tTable      = thicknesses         /. {o} /. so;

          If[ Length[mTable] == 2,
              mTable = Join[ mTable, Reverse[mTable] ]
            ];

          Join[ wollaston[ mTable[[1]], iTable[[1]],
                           mTable[[2]], tTable[[1]] ],
                {air},
                displace[ { 0, 0,
                            tTable[[1]] + dWollastons
                          },
                          wollaston[ mTable[[3]],
                                     iTable[[2]],
                                     mTable[[4]],
                                     tTable[[2]]
                                   ]
                        ]
              ]
        ];
```

```
single Wollaston spectrometer
Options[singleWollastonSpectrometer] =
  { materials -> { Calcite[633 nm, x], Calcite[633 nm, y] },
    internalAngle -> 1.3 Degree,
    thickness -> 4. mm
  };
singleWollastonSpectrometer[ o___ ] :=
 Module[ {so, mTable, i, t},

          (* evaluate options *)
          so = Options[singleWollastonSpectrometer];
          mTable = materials       /. {o} /. so;
          i      = internalAngle   /. {o} /. so;
          t      = thickness       /. {o} /. so;

          wollaston[ mTable[[1]], i,
                     mTable[[2]], t ]
        ];
spectrometer design (lambdaNAndDCamera)
lambdaNAndDCamera::usage =
"Options are passed on to NyquistLambda";
lambdaNAndDCamera::failed =
"calculation of Nyquist wavelength and camera distance failed";
lambdaNAndDCamera[ s_, ca_, opt___ ] :=
lambdaNAndDCamera[ s, ca, opt ] =
  If[ Evaluate[ca] =!= camera[$Failed],

      (* return Nyquist lambda and camera distance *)
      { NyquistLambda[s, ca, opt],
        Min[ opticalPath[
                 rayTrace[ rayInAir[ pixel[ca, 1],
                                     {0, 0, -1} ],
                           {Reverse[ s ][[1]]}
                         ]
               ],
             opticalPath[
                 rayTrace[ rayInAir[ pixel[ca, noOfPixels[ca]],
                                     {0, 0, -1} ],
                           {Reverse[ s ][[1]]}
                         ]
               ]
           ]
      },

      (*** everyghing not ok ***)
      Message[deviation::failed];
      $Failed
    ];
```

alpha-dependence

```
alphaDependence returns the difference between the path differences
(between e- and o-ray) for alpha=1°, beta=0° and alpha=0°, beta=0°
alphaDependence::usage =
"Options are passed on to pixelPD";
(* s = description of spectrometer;
   cam = camera structure *)
alphaDependence[ s_, opt___ ] :=
 alphaDependence[ s, opt ] =
  Module[ { cam, pd0, pd1 },

          cam = setupCamera[s];
          If[ ((pd0 = pixelPD[ s, cam,        0, 0, opt ]) =!=
              $Failed) &&
```

```
        ((pd1 = pixelPD[ s, cam, 1 Degree, 0, opt ]) =!=
        $Failed),

        pd1 - pd0,

        $Failed
    ]
];
```

B.2. Demonstration program

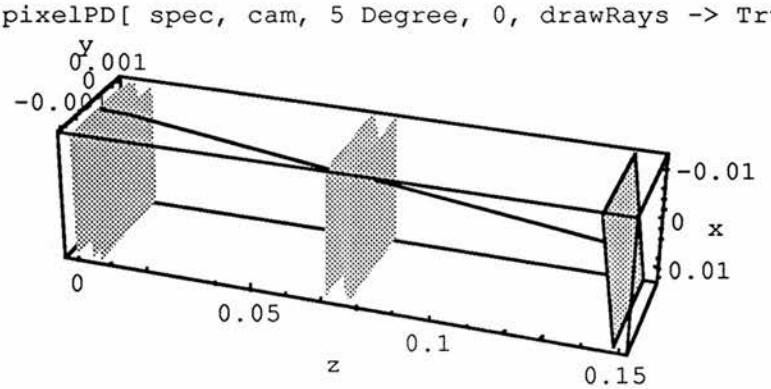
This *Mathematica* program gives an example of how some of the results represented in this work were calculated. It calculates the angular dependent path difference relation of the UV spectrometer.

```
(* the ray tracing package and the spectrometer package
   have to be loaded *)

spec = doubleWollastonSpectrometer[
    materials -> { Quartz[220. nm, x],
                   Quartz[220. nm, y],
                   Quartz[220. nm, y],
                   Quartz[220. nm, x] },
    internalAngles -> {6.0 Degree, 11.8 Degree},
    WollastonSeparation -> 64.5 mm,
    thicknesses -> {7. mm, 7. mm}
  ];

cam = setupCamera[ spec ];

pixelPD[ spec, cam, 5 Degree, 0, drawRays -> True ];
```




```
(* all lengths are in m *)

Plot3D[
    pixelPD[ spec, cam, alpha Degree, beta Degree ] / nm,
    {alpha, -5, 5},
    {beta, -5, 5}
  ];
```