Polarisation

July 8, 2024

1 Fakultät für Physik

1.1 Physikalisches Praktikum P2 für Studierende der Physik

Versuch P2-11 (Stand: April 2024)

Raum F1-14

2 Polarisation und Doppelbrechung

Name: Vrkic Vorname: Tin E-Mail: uyvpq@student.kit.edu

Name: Nock Vorname: Mika E-Mail: uttziqstudent.kit.edu

Gruppennummer: Mo-32

Betreuer: Kerstin Trost

Versuch durchgeführt am: 01.07.24

Beanstandungen zu Protokoll Version _____:

Testiert am: _____ Testat: _____

3 Durchführung

Die Anleitung zu diesem Versuch finden Sie hier.

[2]:
```
import pathlib
import pandas as pd
import numpy as np
import kafe2
import scipy as sc
import matplotlib.pyplot as plt
from uncertainties import ufloat, unumpy as unp
```

[3]:
```
# erstellen einer Funktion für kafe2 Fits
def fit_funktion(xy_data, model_function, xy_error, xy_label, title,␣
 ↪constraint=[], add_error=True):
    xy_data = kafe2.XYContainer(xy_data[0], xy_data[1])
    xy_data.label = title
    fit = kafe2.XYFit(xy_data = xy_data, model_function = model_function)
    if add_error:
        fit.add_error(axis = 'x', err_val = xy_error[0])
        fit.add_error(axis = 'y', err_val = xy_error[1])
    for i in range(len(constraint)):
        fit.add_parameter_constraint(name = constraint[i][0], value =␣
 ↪constraint[i][1], uncertainty = constraint[i][2])
    fit.do_fit()
    plot = kafe2.Plot(fit)
    plot.x_label, plot.y_label = xy_label[0], xy_label[1]
```

```
    return fit.parameter_values, fit.parameter_errors, plot

def weighted_mean_gauss(arr):
    res = np.sum( unp.nominal_values(arr) / unp.std_devs(arr) ) / np.sum( 1 /␣
 ↪unp.std_devs(arr) )
    return res

def std_weighted_mean_gauss(arr):
    N = unp.nominal_values(arr).size
    arr_bar = weighted_mean_gauss(arr)
    return np.sqrt( (N/(N-1)) * np.sum( (unp.nominal_values(arr)-arr_bar)**2 /␣
 ↪unp.std_devs(arr) ) / np.sum( 1/unp.std_devs(arr) ) )
```

3.1 Aufgabe 1: Polarisiertes Licht aus dem Wasserglas

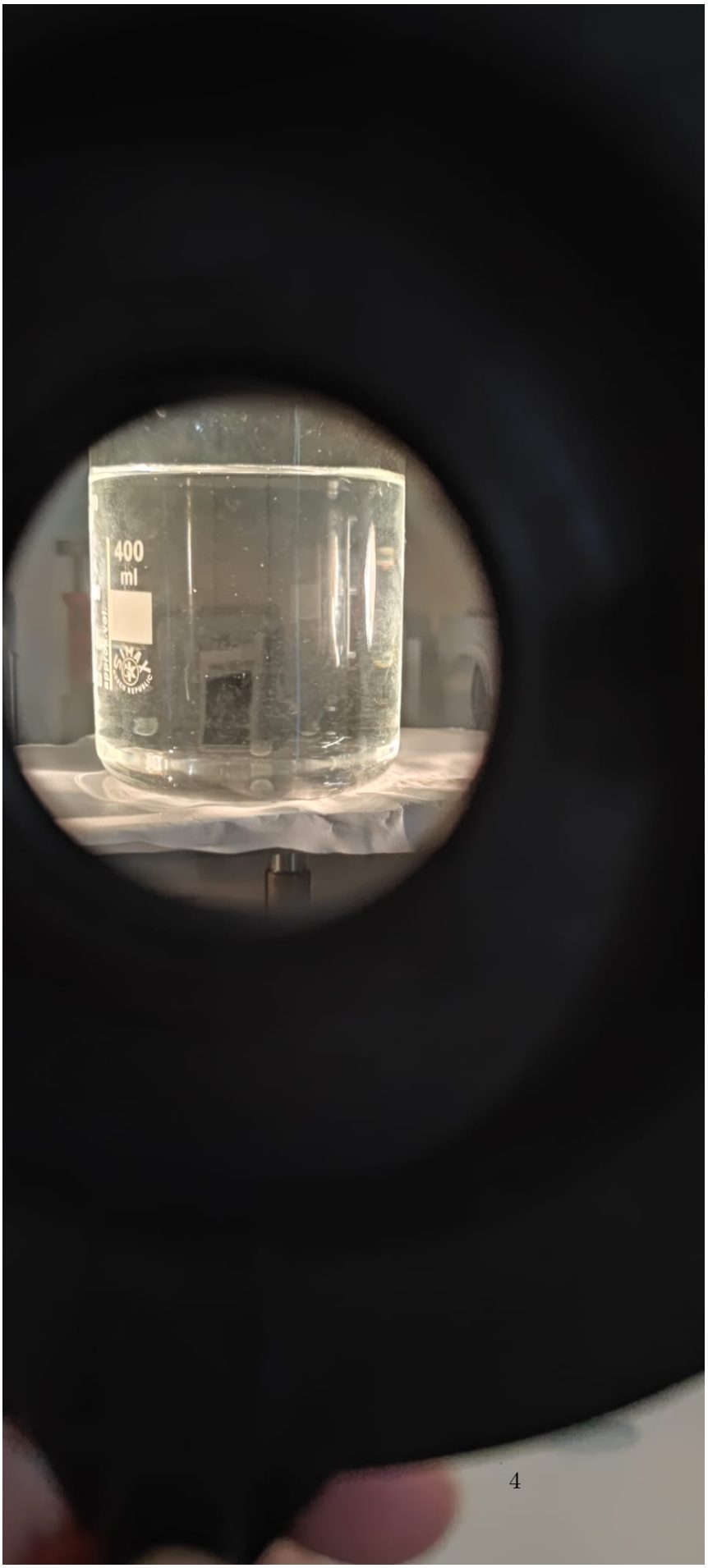
Hinweise zu Aufgabe 1 finden in der Datei Hinweise-Versuchsdurchfuehrung.md.

- Strahlen Sie Licht durch ein mit Wasser gefülltes Glas und beobachten Sie das austretende Streulicht aus verschiedenen Richtungen durch einen Polarisationsfilter.
- Beschreiben und erklären Sie, was Sie beobachten.

In diesem Versuch wurde ein Becherglas, welches mit Wasser gefüllt war, mittels einer Lampe beleuchtet. Beobachtet man diese Anordnung nun in einem Winkel von 90° durch einen Polaristaionsfilter, zeigt sich eine Helligkeitsveränderung je nach Orientierung des Filters.

Diese Helligkeitsveränderung ergibt sich durch die Wassermoleküle, die aufgrund der Elektronegativität des Sauerstoffs als Dipole agieren. Sie werden durch das Licht zu Schwingungen angeregt. Da diese Dipole hauptsächlich senkrecht zur Einfallrichtung des Lichtes abstrahlen ist der Effekt großteilig unter 90° beobachtbar. Außerdem ist das ausgesendete Licht polarisiert. Der Polarisationsfilter erlaubt somit je nach Orientierung die Beobachtung eines helleren bzw. eines dunkleren Bildes.

In den unten gezeigten Grafiken ist zuerst das hellere Wasserglas sichtbar, bei dem der Polfilter das abgestrahlte Licht transmittiert. Im zweiten, dunkleren Bild wird ein Teil des Lichtes herausgefiltert. Im dunkleren Bild ist ein Gelbstich erkennbar, der vermutlich aus der automatischen Bildbearbeitung der Kamera stammt.


400
ml

3.2 Aufgabe 2: Erzeugung und Untersuchung von Licht mit verschiedener Polarisation

Hinweise zu Aufgabe 2 finden in der Datei Hinweise-Versuchsdurchfuehrung.md.

- Erzeugen und untersuchen Sie die Intensitätsverteilungen von verschieden polarisiertem Licht.
- Bearbeiten Sie hierzu die folgenden Aufgaben.

3.2.1 Aufgabe 2.1: Aufbau des Strahlengangs

- Bauen Sie geeignete Strahlengänge zur Erzeugung von **linear**, **elliptisch** und **zirkular** polarisiertem Licht auf.

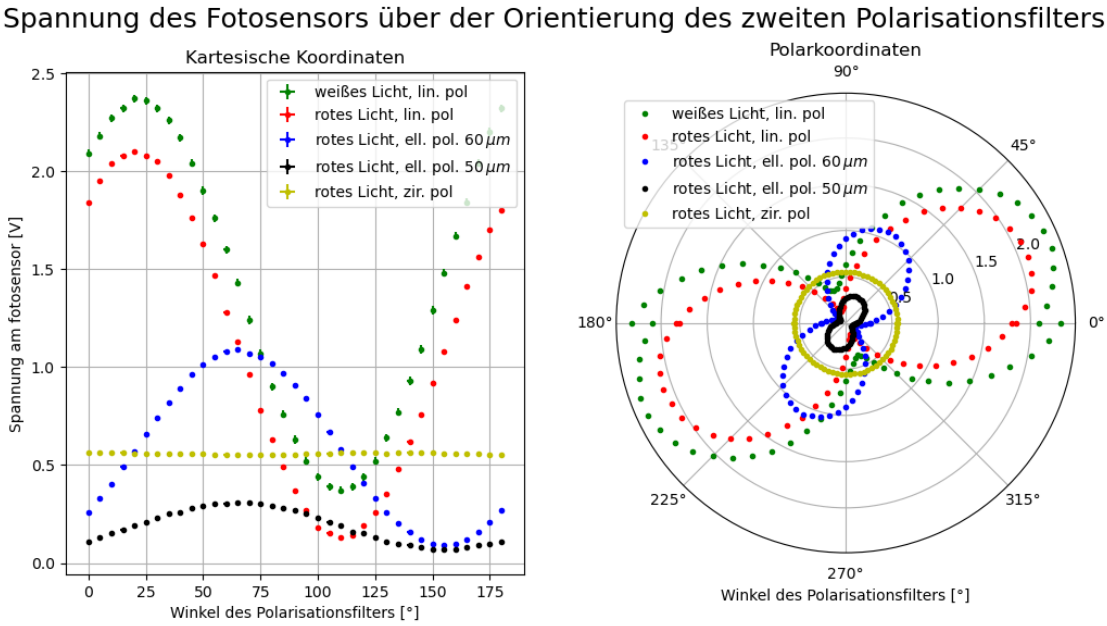
- Bestimmen Sie die Intensitätsverteilungen des Lichts jeweils als Funktion des Winkels φ eines zweiten linearen Polarisationsfilters.

---

[4]:
```
# Messungen
phi = unp.uarray(np.arange(0, 181, 5, dtype=np.float32), [1.])
# Weißes Licht, linear polarisiert
I_weiß_lin = unp.uarray([2.09, 2.18, 2.27, 2.32, 2.37, 2.36, 2.32, 2.26, 2.17,␣
 ↪2.04, 1.90, 1.76, 1.60, 1.43, 1.24, 1.07, 0.90, 0.76, 0.63, 0.52, 0.44, 0.
 ↪39, 0.37, 0.39, 0.44, 0.52, 0.64, 0.77, 0.93, 1.09, 1.29, 1.48, 1.67, 1.84,␣
 ↪2.04, 2.2, 2.32] , [0.02]) # in Volt
# Monochromatisches Rotes Licht, linear polarisiert
I_rot_lin = unp.uarray([1.84, 1.95, 2.04, 2.08, 2.10, 2.08, 2.05, 1.98, 1.88, 1.
 ↪76, 1.63, 1.47, 1.28, 1.13, 0.96, 0.78, 0.63, 0.49, 0.37, 0.27, 0.18, 0.15,␣
 ↪0.13, 0.14, 0.19, 0.26, 0.35, 0.48, 0.62, 0.76, 0.92, 1.08, 1.24, 1.41, 1.
 ↪56, 1.70, 1.80] , [0.01])
# Monochromatisches Rotes Licht, elliptisch polarisiert, Dicke Glimmerplättchen:
 ↪ 60 mikrometer
I_rot_ell_60 = unp.uarray([0.26, 0.33, 0.40, 0.49, 0.57, 0.66, 0.74, 0.82, 0.
 ↪89, 0.96, 1.01, 1.05, 1.08, 1.09, 1.07, 1.05, 1.02, 0.97, 0.91, 0.84, 0.76,␣
 ↪0.67, 0.58, 0.49, 0.41, 0.33, 0.26, 0.20, 0.16, 0.12, 0.10, 0.09, 0.10, 0.
 ↪12, 0.16, 0.21, 0.27] , [0.01])
# Monochromatisches Rotes Licht, elliptisch polarisiert, Dicke Glimmerplättchen:
 ↪ 50 mikrometer
I_rot_ell_50 = unp.uarray([0.11, 0.13, 0.15, 0.17, 0.19, 0.21, 0.23, 0.25, 0.
 ↪26, 0.28, 0.29, 0.30, 0.30, 0.31, 0.31, 0.30, 0.29, 0.28, 0.27, 0.25, 0.23,␣
 ↪0.21, 0.19, 0.16, 0.15, 0.13, 0.11, 0.10, 0.09, 0.08, 0.07, 0.07, 0.07, 0.
 ↪08, 0.09, 0.10, 0.11] , [0.01])
# Monochromatisches Rotes Licht, zirkular polarisiert, Lambda/4 Plättchen ->␣
 ↪Zirkular
I_rot_zir = unp.uarray([0.564, 0.562, 0.562, 0.562, 0.559, 0.560, 0.560, 0.559,␣
 ↪0.559, 0.558, 0.556, 0.553, 0.552, 0.552, 0.552, 0.551, 0.554, 0.553, 0.554,␣
 ↪0.556, 0.558, 0.560, 0.561, 0.561, 0.563, 0.563, 0.560, 0.561, 0.561, 0.561,␣
 ↪0.561, 0.560, 0.560, 0.559, 0.558, 0.554, 0.553] , [0.002])
```

```
# making a figure
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(121)
ay = fig.add_subplot(122,projection="polar")

# Fun with kartesian coordinates
ax.errorbar(unp.nominal_values(phi), unp.nominal_values(I_weiß_lin), xerr=unp.
 ↪std_devs(phi), yerr=unp.std_devs(I_weiß_lin), fmt="g.", label="weißes Licht,␣
 ↪lin. pol")
ax.errorbar(unp.nominal_values(phi), unp.nominal_values(I_rot_lin), xerr=unp.
 ↪std_devs(phi), yerr=unp.std_devs(I_rot_lin), fmt="r.", label="rotes Licht,␣
 ↪lin. pol")
ax.errorbar(unp.nominal_values(phi), unp.nominal_values(I_rot_ell_60), xerr=unp.
 ↪std_devs(phi), yerr=unp.std_devs(I_rot_ell_60), fmt="b.", label="rotes␣
 ↪Licht, ell. pol. 60$\,\mu m$")
ax.errorbar(unp.nominal_values(phi), unp.nominal_values(I_rot_ell_50), xerr=unp.
 ↪std_devs(phi), yerr=unp.std_devs(I_rot_ell_50), fmt="k.", label="rotes␣
 ↪Licht, ell. pol. 50$\,\mu m$")
ax.errorbar(unp.nominal_values(phi), unp.nominal_values(I_rot_zir), xerr=unp.
 ↪std_devs(phi), yerr=unp.std_devs(I_rot_zir), fmt="y.", label="rotes Licht,␣
 ↪zir. pol")

# Fun with polar coordinates
ay.plot(unp.nominal_values(phi) * np.pi/180, unp.nominal_values(I_weiß_lin), "g.
 ↪", label="weißes Licht, lin. pol")
ay.plot(unp.nominal_values(phi) * np.pi/180 - np.pi , unp.
 ↪nominal_values(I_weiß_lin), "g.")
ay.plot(unp.nominal_values(phi) * np.pi/180, unp.nominal_values(I_rot_lin), "r.
 ↪", label="rotes Licht, lin. pol")
ay.plot(unp.nominal_values(phi) * np.pi/180 - np.pi , unp.
 ↪nominal_values(I_rot_lin), "r.")
ay.plot(unp.nominal_values(phi) * np.pi/180, unp.nominal_values(I_rot_ell_60),␣
 ↪"b.", label="rotes Licht, ell. pol. 60$\,\mu m$")
ay.plot(unp.nominal_values(phi) * np.pi/180 - np.pi , unp.
 ↪nominal_values(I_rot_ell_60), "b.")
ay.plot(unp.nominal_values(phi) * np.pi/180, unp.nominal_values(I_rot_ell_50),␣
 ↪"k.", label="rotes Licht, ell. pol. 50$\,\mu m$")
ay.plot(unp.nominal_values(phi) * np.pi/180 - np.pi , unp.
 ↪nominal_values(I_rot_ell_50), "k.")
ay.plot(unp.nominal_values(phi) * np.pi/180, unp.nominal_values(I_rot_zir), "y.
 ↪", label="rotes Licht, zir. pol")
ay.plot(unp.nominal_values(phi) * np.pi/180 - np.pi , unp.
 ↪nominal_values(I_rot_zir), "y.")

# making stuff look nice
```

```
fig.suptitle('Spannung des Fotosensors über der Orientierung des zweiten␣
 ↪Polarisationsfilters', size=18)
ax.set_xlabel('Winkel des Polarisationsfilters [°]')
ax.set_ylabel('Spannung am fotosensor [V]')
ay.set_xlabel('Winkel des Polarisationsfilters [°]')
ax.set_title('Kartesische Koordinaten')
ay.set_title('Polarkoordinaten')
ax.grid(True)
ay.grid(True)
ax.legend()
ay.legend()

plt.show()
```



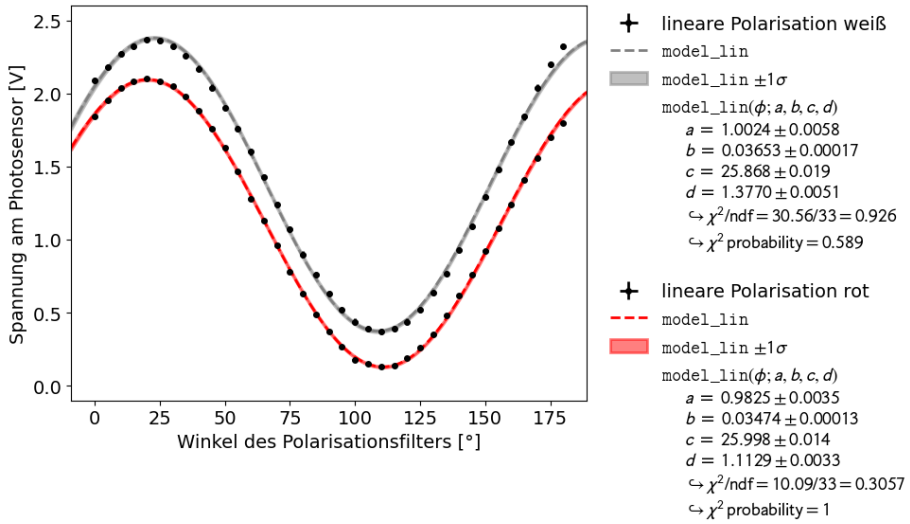
Hier wurde ein Strahlengang realisiert, bei dem das einfallende Licht auf zwei Polarisationsfilter gelenkt wird und dann auf einen Fotosensor trifft, dessen erzeugte Spannugn dann ausgelesen wird. Um eine messbare Spannung bei monochromatischem Licht zu erhalten wird zusätzlich hinter dem letzten Polfilter mit einer Linse auf den Sensor gebündelt.

Hierbei werden vier verschiedenen Szenarien einzeln betrachtet: - Es befinden sich keine zusätzlichen Geräte auf der Schiene, was nach dem ersten Filter zu einer linearen Polariation führt, die unverändert auf den zweiten Filter trifft. (Grün) - Vor der Lampe befindet sich ein Monochromator, der das Licht Rot färbt, welches dann ebenfalls linear polarisiert wird. (Rot) - Zwischen den beiden Filtern befindet sich eine doppelbrechende Glimmerplatte mit $60\mu m$ Dicke, die für eine elliptische Polarisation sorgt (Blau) - Zwischen den Filtern befindet sich eine $50\mu m$ dicke Glimmerplatte, die ebenfalls für elliptische Polarisation sorgt (Schwarz) - Eine $\frac{\lambda}{4}$-Platte sorgt für eine fast zirkulare Polarisation (Gelb)

In der oben abgebildeten Grafik sind die Spannungen am Fotosensor bei verschiedenen Winkel des Polarisationsfilters gezeigt. DIe verschiedenen Szenarien sind hierbei in den entsprechenden Farben aus der Auflistung zu sehen. Im rechten Plot wurden hierbei Polarkoordinaten verwendet und die Daten zweimal hintereinander gesetzt um eine geschlossene Darstellung bis 360° zu erhalten.

Genauere Informationen zur Datennahme und den verschiedenen Szenarien folgen in weiteren Beschreibungen.

[5]:
```
# making fits for the different colours and polarisations
# definition of the labels '
axis_labels = ('Winkel des Polarisationsfilters [°]','Spannung am Photosensor␣
 ↪[V]')

# linear polarisation
# definition of a model
def model_lin(phi,a,b=0.04,c=25,d=1.25):
    return a * np.sin(b*phi + c) + d

# defining the data
lw_data = kafe2.XYContainer(unp.nominal_values(phi), unp.
 ↪nominal_values(I_weiß_lin))
lw_data.label = 'lineare Polarisation weiß'
lw_data.axis_labels = axis_labels

lr_data = kafe2.XYContainer(unp.nominal_values(phi), unp.
 ↪nominal_values(I_rot_lin))
lr_data.label = 'lineare Polarisation rot'
lr_data.axis_labels = axis_labels

# doing the fits
lin_white = kafe2.XYFit(lw_data, model_lin)
lin_white.add_error('x', unp.std_devs(phi))
lin_white.add_error('y', unp.std_devs(I_weiß_lin))
lin_white.do_fit()

lin_rot = kafe2.XYFit(lr_data, model_lin)
lin_rot.add_error('x', unp.std_devs(phi))
lin_rot.add_error('y', unp.std_devs(I_rot_lin))
lin_rot.do_fit()

# plotting the fits
plot = kafe2.Plot([lin_white, lin_rot])
plot.customize('model_line','color',[(0,'grey'),(1,'red')])
plot.customize('data','color',[(0,'black'),(1,'black')])
plot.customize('model_error_band','color',[(0,'grey'),(1,'red')])
plot.y_range = (-0.1,2.6)
plot.plot()
plot.show()
```



Bei den oben gezeigten Daten handelt es sich um die ersten beiden der vier Szenarien. In Grau ist die Intensität der linearen Polarisation von weißem Licht und in Rot die Polarisation von Rotem Licht dargestellt.

Es fällt auf, dass die Intensität des Roten Lichtes in ihrem Minimum näher bei Null liegt, als die Intensität des weißen Lichtes. Dieser Effekt resultiert aus der Wirkung des Filters, der verschiedene Wellenlängen bei verschiedenen Winkeln polarisiert. Das weiße Licht ist hierbei eine Mischung verschiedener Wellenlängen, die in leicht verschiedenen Winkeln polarisiert werden. Es existieren also Komponenten, die bei der vollständigen Absorptione der einen Farbe immernoch teilweise transmittiert werden.

Außerdem wäre die Amplitude der Roten Kurve deutlich geringer, da weniger Gesamtintensität am Fotosensor ankommt. Jedoch wurde eine Blende geöffnet, damit mehr Licht von der Lampe ausgesendet wird und die Intensität ungefähr gleich bleibt.

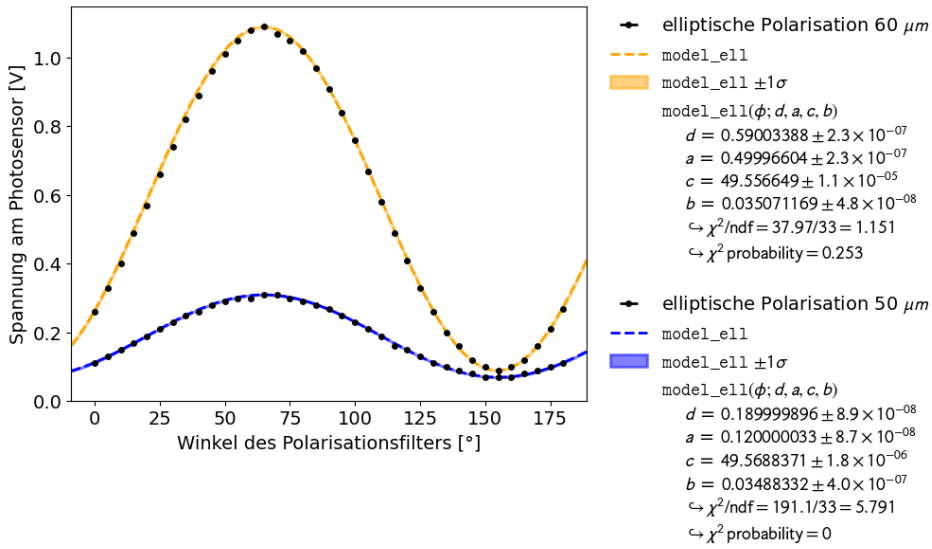
Die Fits liegen in einem akzeptablen bereich mit $\chi^2 < 1$ und es resultieren die Funktionen der Spannung für weißes und monochromatisches Licht:

$$U_w(\phi) = (1.0024 \pm 0.0058) \cdot sin((0.003653 \pm 0.00017) \cdot \phi + (25.868 \pm 0.019)) + (1.3770 \pm 0.0051)$$

$$U_r(\phi) = (0.9825 \pm 0.0035) \cdot sin((0.003474 \pm 0.00013) \cdot \phi + (25.998 \pm 0.014)) + (1.1129 \pm 0.0033)$$

[6]:
```
# making fits for the different colours and polarisations
# definition of the labels '
axis_labels = ('Winkel des Polarisationsfilters [°]','Spannung am Photosensor␣
 ↪[V]')
```

```
# elliptic polarisation
# definition of a model
def model_ell(phi,d,a,c=50,b=0.04):
    return a * np.sin(b*phi + c) + d

# defining the data
ell60_data = kafe2.XYContainer(unp.nominal_values(phi), unp.
 ↪nominal_values(I_rot_ell_60))
ell60_data.label = 'elliptische Polarisation 60 $\mu m$'
ell60_data.axis_labels = axis_labels

ell50_data = kafe2.XYContainer(unp.nominal_values(phi), unp.
 ↪nominal_values(I_rot_ell_50))
ell50_data.label = 'elliptische Polarisation 50 $\mu m$'
ell50_data.axis_labels = axis_labels

# doing the fits
ell60 = kafe2.XYFit(ell60_data, model_ell)
ell60.add_error('x', unp.std_devs(phi))
ell60.add_error('y', unp.std_devs(ell60_data))
ell60.do_fit()

ell50 = kafe2.XYFit(ell50_data, model_ell)
ell50.add_error('x', unp.std_devs(phi))
ell50.add_error('y', unp.std_devs(ell50_data))
ell50.do_fit()

# plotting the fits
plot = kafe2.Plot([ell60, ell50])
plot.customize('model_line','color',[(0,'orange'),(1,'blue')])
plot.customize('data','color',[(0,'black'),(1,'black')])
plot.customize('model_error_band','color',[(0,'orange'),(1,'blue')])
plot.y_range = (0,1.15)
plot.plot()
plot.show()
```



In diesem Versuchteil werden nach Szenario drei und vier doppelbrechende Glimmerplatten verschiedener Dicken zwischen den Polfiltern eingebracht.

Zu Beginn müssen die Glimmerplatten so ausgerichtet sein, dass beide optischen Achsen gleich stark beleuchtet werden. Es wird jeweils ein Anteil, der parallel zu einer der optischen Achsen steht schneller bzw. langsamer, wodurch eine Phasenverschiebung und so eine elliptische Polarisation entsteht.

Die beiden gemachten Fits sind noch zu akzeptieren, wobei $\chi^2 = 5.79$ der Kurve für $50\mu m$ eher zu hoch ausfällt.

Wir erhalten für die beiden Dicken jeweils:

$60\mu m$:

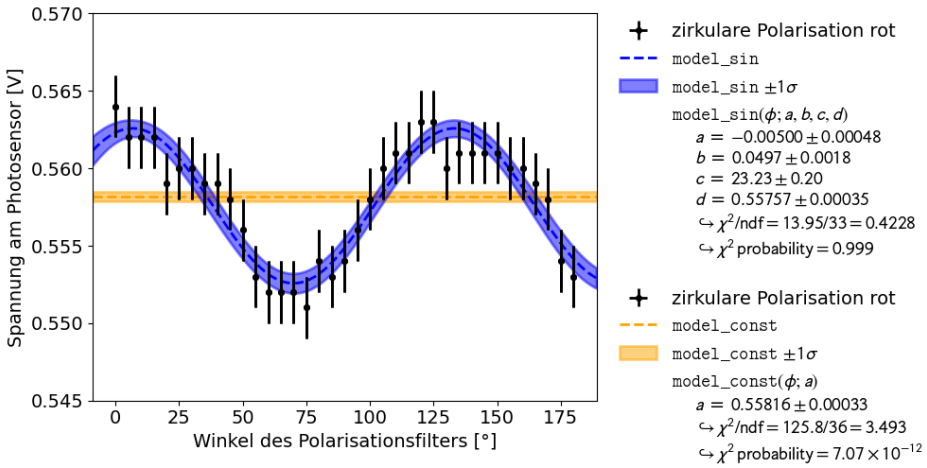
$$U(\phi) = (0.5000 \pm 0) \cdot sin((0.0351 \pm 0) \cdot \phi + (49.56 \pm 0)) + (0.5900 \pm 0)$$

$50\mu m$:

$$U(\phi) = (0.1200 \pm 0) \cdot sin((0.03488 \pm 0) \cdot \phi + (49.56 \pm 0)) + (0.1900 \pm 0)$$

[7]:
```
# making fits for the different colours and polarisations
# definition of the labels '
axis_labels = ('Winkel des Polarisationsfilters [°]','Spannung am Photosensor␣
 ↪[V]')

# circular polarisation
# definition of a model
def model_sin(phi,a,b=0.04,c=25,d=1.25):
    return a * np.sin(b*phi + c) + d

def model_const(phi,a):
```

```
    return a

# defining the data
zir_data = kafe2.XYContainer(unp.nominal_values(phi), unp.
 ↪nominal_values(I_rot_zir))
zir_data.label = 'zirkulare Polarisation rot'
zir_data.axis_labels = axis_labels

# doing the fits
zir_sin = kafe2.XYFit(zir_data, model_sin)
zir_sin.add_error('x', unp.std_devs(phi))
zir_sin.add_error('y', unp.std_devs(I_rot_zir))
zir_sin.do_fit()

zir_const = kafe2.XYFit(zir_data, model_const)
zir_const.add_error('x', unp.std_devs(phi))
zir_const.add_error('y', unp.std_devs(I_rot_zir))
zir_const.do_fit()

# plotting the fits
plot = kafe2.Plot([zir_sin, zir_const])
plot.customize('model_line','color',[(0,'blue'),(1,'orange')])
plot.customize('data','color',[(0,'black'),(1,'black')])
plot.customize('model_error_band','color',[(0,'blue'),(1,'orange')])
plot.y_range = (0.545,0.57)
plot.plot()
plot.show()
```



Im fünften Szenario wird nun eine $\frac{\lambda}{4}$-Platte in den Strahlengang eingebracht, die ebenfalls so ausgerichtet wird, dass beide optischen Achsen gleichermaßenn beleuchtet werden.

Durch die Anpassung der Modelle $U(\phi) = C$ und $U(\phi) = a \cdot sin(b\phi + c) + d$ zeigt sich in $\chi^2 = 3.49$ des orangen Fits, dass eine Konstante die vorliegende Situation nur in Näherung beschreiben könnte. Außerdem ist das Sinusförmige Verhalten in der Betrachtung auf einer geeigneten Größenskala bereits klar zu erkennen. Bestätigung wird durch den blauen Fit mit $\chi^2 = 0.423$ erhalten.

Diese Abweichung von der eigentlich zu erwartenden Konstanten ist zu erklären durch Fehler der Platte selbst oder durch eine ungenaue Ausrichtung, die wieder eine elliptische Polarisation zur Folge hat.

Es resultieren die folgenden Modelle, von denen zweiteres besser zur Beschreibung der Messdaten geeignet ist.

$$U(\phi) = 0.55816 \pm 0.00033$$
$$U(\phi) = (-0.00500 \pm 0.00048) \cdot sin((0.0497 \pm 0.0018) \cdot \phi + (23.23 \pm 0.20)) + (0.55757 \pm 0.00035)$$

3.2.2 Aufgabe 2.2: Differenz der Brechungsindizes der beobachteten Strahlen

Bestimmen Sie die Differenz $\Delta n = (n_\beta - n_\gamma)$ der beobachteten Strahlen. Verwenden Sie hierzu die gemessenen Intensitätsverteilungen für den Fall elliptisch polarisierten Lichts.

Für diese Aufgabe haben wir die Intensität von elliptisch polarisiertem rotem Licht nach Durchgang durch Glimmerplättchen der Dicke 50 m und 60 m gemessen. Da die Glimmerplättchen zwei optische Achsen besitzen, sammeln die Anteile der Strahlen, die Aufgrund ihrer Polarisation unterschiedliche optische Achsen "sehen", Phasendifferenzen gegeneinander auf. Diese Differenz $\Delta n = (n_\beta - n_\gamma)$ lässt sich über die in der Vorbereitung hergeleitete Formel berechnen:

$$\Delta n = \frac{\lambda_0}{2\pi d} arctan(\sqrt{\frac{U_{min}}{U_{max}}})$$

Die maximalen und minimalen Intensitäten (entprechen den gemessenen Spannungen) lassen sich aus den Fits der 2.1 bestimmen.
Das wird im Folgenden getan, indem zunächst die Werte der gefitteten Parameter aus den relevanten Fits oben genommen werden und dann die maximalen und minimalen Intensitäten der mit diesen gefitteten Werten für einige x-Werte modellierten Funktion gefunden werden.

[8]:
```
# Hole die Werte aus den Fits
d60, a60, c60, b60 = ell60.parameter_values[0], ell60.parameter_values[1],␣
 ↪ell60.parameter_values[2], ell60.parameter_values[3]
d50, a50, c50, b50 = ell50.parameter_values[0], ell50.parameter_values[1],␣
 ↪ell50.parameter_values[2], ell50.parameter_values[3]
# Setze Werte in die Sinus Funktion ein
func60 = a60*np.sin(b60*np.linspace(0,180, 100)+c60)+d60
func50 = a50*np.sin(b50*np.linspace(0,180, 100)+c50)+d50

# Finde die Peaks
peaks_60max = sc.signal.argrelextrema(func60, np.greater)
peaks_60min = sc.signal.argrelextrema(func60, np.less)
peaks_50max = sc.signal.argrelextrema(func50, np.greater)
```

```
peaks_50min = sc.signal.argrelextrema(func50, np.less)

print(f"Ell. Pol., 60um, Maximum: I = {func60[peaks_60max][0]:.2f} V")
print(f"Ell. Pol., 60um, Minimum: I = {func60[peaks_60min][0]:.2f} V")
print(f"Ell. Pol., 50um, Maximum: I = {func50[peaks_60max][0]:.2f} V")
print(f"Ell. Pol., 50um, Minimum: I = {func50[peaks_50min][0]:.2f} V")
```

```
Ell. Pol., 60um, Maximum: I = 1.09 V
Ell. Pol., 60um, Minimum: I = 0.09 V
Ell. Pol., 50um, Maximum: I = 0.31 V
Ell. Pol., 50um, Minimum: I = 0.07 V
```

Für die Dicke des Glimmerplättchens von $d_1 = 60$ m ist
- $U_{min} = (0.09 \pm 0.01)$ V
- $U_{max} = (1.09 \pm 0.01)$ V

Für die Dicke des Glimmerplättchens von $d_2 = 50$ m ist
- $U_{min} = (0.07 \pm 0.01)$ V
- $U_{max} = (0.31 \pm 0.01)$ V

Die Wellenlänge des Lichts, nachdem es durch den Interferenzfarbfilter gelaufen ist, beträgt $\lambda_0 = 635$ nm.
Im Prinzip hätte man die maximalen und minimalen Intensitäten auch direkt aus dem Array der Messung ablesen können, hierdurch können wir uns aber sicher sein, dass die Werte tatsächlich die richtigen sind.

[9]:
```
# Minimale und maximale Intensitäten in Volt, bestimmt aus der 2.1 per Fit
I_min_60 = ufloat(0.09 , 0.01)
I_max_60 = ufloat(1.09 , 0.01)
I_min_50 = ufloat(0.07 , 0.01)
I_max_50 = ufloat(0.31 , 0.01)
lbda = 635 * 10**(-9) # Meter
d1 = 60 * 10**(-6) # Meter
d2 = 50 * 10**(-6) # Meter

# Berechne delta n für beide Dicken
delta_n_60 = lbda * unp.arctan( unp.sqrt( I_min_60 / I_max_60 ) ) / ( 2 * np.pi␣
 ↪* d1 )
delta_n_50 = lbda * unp.arctan( unp.sqrt( I_min_50 / I_max_50 ) ) / ( 2 * np.pi␣
 ↪* d2 )
delta_n_arr = unp.uarray([ unp.nominal_values(delta_n_60) , unp.
 ↪nominal_values(delta_n_50) ] , [ unp.std_devs(delta_n_60) , unp.
 ↪std_devs(delta_n_60) ])
# Gewichteter Mittelwert der beiden Werte
delta_n = ufloat( weighted_mean_gauss( delta_n_arr ) , std_weighted_mean_gauss(␣
 ↪delta_n_arr ) )

print(f"delta n mit Dicke 60um = {delta_n_60}")
```

```
print(f"delta n mit Dicke 50um = {delta_n_50}")
print(f"Gewichteter Mittelwert delta n = {delta_n:.5f}")
```

```
delta n mit Dicke 60um = 0.000471+/-0.000025
delta n mit Dicke 50um = 0.00090+/-0.00006
Gewichteter Mittelwert delta n = 0.00068+/-0.00030
```

Wir erhalten für die Differenz von "schnellem" und "langsamem" Brechungsindex folgenden gewichteten Mittelwert: $\Delta n = (6.8 \pm 3.0) \cdot 10^{-4}$. Beim Vergleich der einzelnen Werte fällt jedoch auf, dass der jeweils andere nicht im eigenen Unsicherheitsbereich liegt. Für das 60 m dicke Glimmerplättchen erhalten wir nämlich $\Delta n = (4.71 \pm 0.25) \cdot 10^{-4}$, während das 50 m dicke Glimmerplättchen $\Delta n = (9.0 \pm 0.6) \cdot 10^{-4}$ liefert. Der Unterschied kann gut daran liegen, dass die Glimmerplättchen nicht exakt ausgerichtet werden können, sondern die Ausrichtung Fehlerbehaftet ist. Dadurch hat das Licht nicht einen Weg von 60 m bzw. 50 m durch das Plättchen, sondern etwas mehr, da es dann eben schräg im Laufweg des Lichts steht. Da die Unterschiede der Brechungsindize aber offensichtlich sehr gering sind, macht sich dieser Fehler sehr stark bemerkbar, daher dieser große Unterschied.

3.3 Aufgabe 3: Beobachtungen mit polarisiertem Licht

Hinweise zu Aufgabe 3 finden in der Datei Hinweise-Versuchsdurchfuehrung.md.

- Beobachten Sie einige Beispiele, wo Doppelbrechung im Alltag auftaucht und z.T. auch technisch angewandt wird.
- Bearbeiten Sie hierzu die folgenden Aufgaben.

3.3.1 Aufgabe 3.1: Doppelbrechung am Klebefilm

- An handelsüblichen Klebefilmen tritt Doppelbrechung auf, die Sie mit linear polarisiertem Licht sichtbar machen können.
- Untersuchen Sie die am Versuch ausliegenden Klebefilme und beschreiben Sie, was Sie beobachten.
- Stellen Sie eigene Klebefilmkonstruktionen her, um sich mit dem beobachteten Phänomen vertraut zu machen.
- Fügen Sie Ihrem Protokoll entsprechende Aufnahmen zu.

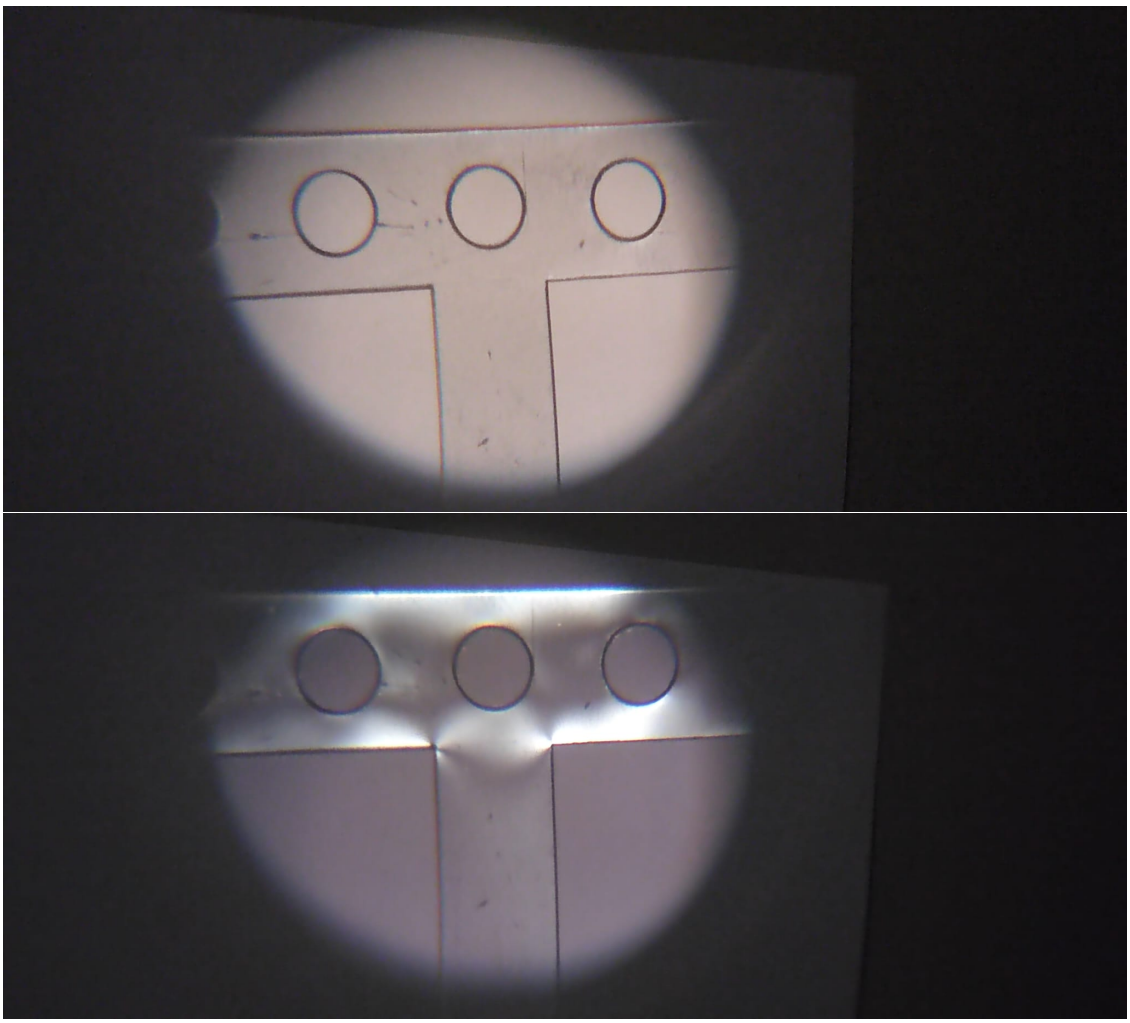
In den gezeigten Grafiken sind verschiedenen Bilder aus Klebefilm zu sehen, die mit linear polarisiertem Licht bestrahlt werden. Dieses wird dann durch einen zweiten Polarisationsfilter geleitet und kann mithilfe einer Linse an der Wand abgebildet werden.

Der Doppelbrechende Klebefilm verdreht nun die Polarisation des einfallendes Lichtes erneut, sodass

hinter dem zweiten Polfilter je nach Dicke und Orientierung des Bandes und Winkel in dem der Dieser steht verschiedene Farben an der Wand zu erkennen sind. So haben beispielsweise die Schnurrhare der Katze verschiedene Farben, je nachdem ob sie allein oder in Überlagerung mit dem Kopf der Katze stehen.

3.3.2 Aufgabe 3.2: Doppelbrechung unter mechanischer Spannung

- Eine technische Anwendung der Doppelbrechung besteht in der Sichtbarmachung von Stellen an Materialien, die besonderer mechanischer Spannung unterliegen.
- Am Versuch liegen einige Modelle aus. Versetzen Sie diese mechanischer Spannung und beobachten Sie die Transmission linear polarisierten Lichts.
- Beschreiben Sie, was Sie beobachten.
- Fügen Sie Ihrem Protokoll entsprechende Aufnahmen zu.



Ersetzt man den doppelbrechenden Klebefilm durch einen Plexiglaskörper und stellt beide Polfilter in 90° zueinander auf, so zeigt sich, dass eine Belastung des Pexiglases zu einem Aufleuchten bestimmter Bereiche führt.

Dieses Aufleuchten resultiert daraus, dass die Polarisation des transmittierten Lichtes gedreht wird,

wenn es durch das unter Stress stehende Plexiglas läuft. Der zweite Polfilter kann dieses nun gedrehte Licht nichtmehr vollständig absorbieren und es ist ein transmittierter Anteil zu erkennen.
