Lab Book: LUCAS SARAGOSA

ESN 730023793 | lss213@exeter.ac.uk

Contents

Experiment 1: Polarisation and Birefringence

@September 29, 2025

Aims:

- Research physics around the experiment
- What is birefringence and how can we measure it
- How can we alter polarisation and investigate it

Light can be polarised, meaning that the direction the electric field is well defined. It can be linearly polarised, meaning the electric field component only oscillates (always in the perpendicular plane to movement) in one direction in that plane, eg up and down and doesn't change. The electric field / optical disturbance only oscillates in that plane, there is never a component in z if its travelling in +z/-z (1).

One question we can ask is what happens when two colinear light waves of different linear polarisation are combined? Due to superposition we know that they will combine, but we would imagine it might oscillate in the line in the plane at 45 degrees relative to the two (by adding the vectors).

Light can also be circularly polarised, meaning it doesn't oscillate in just one line in the plane of the electric field but it can oscillate across the whole plane, perhaps in an elliptical or circular manner. By issuing a phase difference between the waves this could happen.

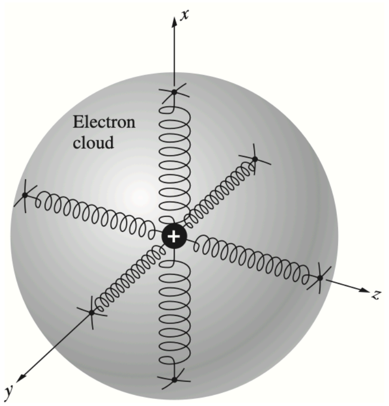

Fig. 1

A birefringent material has interesting properties. For example, many crystalline materials (periodic) are optically anisotropic, meaning that the optical properties change depending on the direction you put an EM wave through it (1, 8.4). Perhaps the atom has (modelled with spring connections) different spring constant strengths in the up-down direction from the left-right direction. Since an EM wave has that electric field component, depending on the direction of the incoming EM wave that EM wave can be refracted differently because the electrons are driven and reradiate differently. We can here say that it is *dependent on the P-state* of the EM wave, ie its polarisation (1). This spring model is modelled in Fig. 1. We can effectively say that birefringent materials have a polarisation-state dependent refractive index.

To measure birefringence, we can first pass light through the birefringent material and then crucially have a polariser after this, such that we only measure the intensity at a specific direction. We can rotate the polariser to investigate the full intensity on angle. We can then rotate the birefringent material and record a full rotation of the polariser for each birefringent material rotation.

A birefringent material has a slow and fast axis. The fast axis means light polarised in that axis will travel fastest compared to any other rotation, and opposite for slowest [1].

In the next few sessions we will investigate bifringence and polarisation and find what else could be investigated in this experiment.

@October 3, 2025 & @October 6, 2025

Aims

- Gather initial readings, take preliminary readings,
- Understand and plan what measurements we want to take,
- Get familiar with the setup and the equipment, and develop code.

I started by writing and initial code to rotate both piezoelectric motors which have the polariser and birefringent material inside them, using the ELL14 Thorlabs rotator. As well as this, the output from the Arduino connected to the Thorlabs oscilloscope reading the intensity from the photodetector is read in through a third serial USB stream.

This code pretty much worked and would rotate the polariser from 0 to 180 for each birefringent rotation 0 to 180. This is then exported to a CSV so it can be processed out of lab. The output (arbitrary units) is read from the oscilloscope which can be normalised later to see the detected light. See Appendix A.

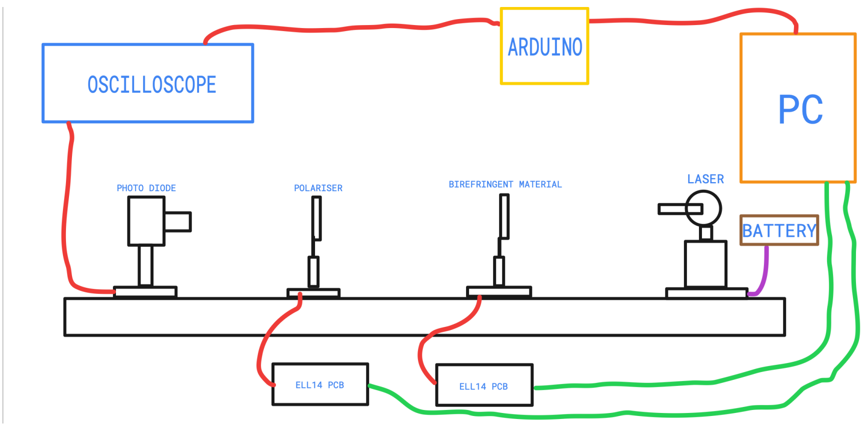

Fig 2.

Above is rough sketch of the diagram of apparatus. There are two rotators, ELL14 PCB boards controlled by an FTDI chip which ‘talks’ to the computer via standard USB serial communication. The oscilloscope outputs to an Arduino connected also by serial USB to the computer.

Note we are using 0 - 180 because 180 to 360 is a mirror or an upside down mirror.

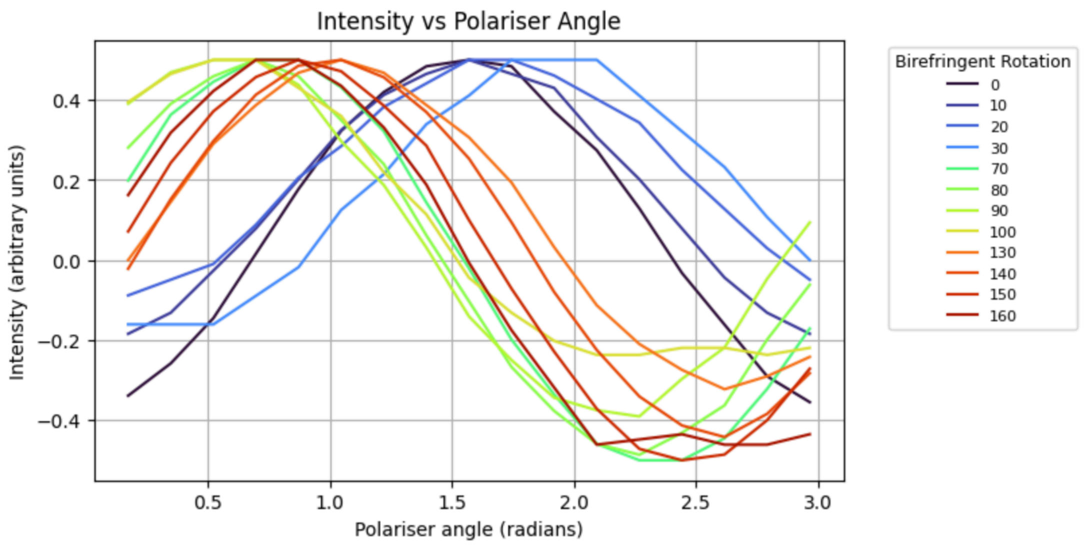

Fig 3.

After removing all erroneous readings (motor stopped due to bad code for a few of them) we got some good preliminary results. All are cosine curves, as expected and each birefringent rotation has a different phase offset, which is also as expected and relates directly to the orientation of the birefringent material (in this case a half wave plate. The code for this is in Appendix B.

Plotting this with no wave plate or birefringent material:

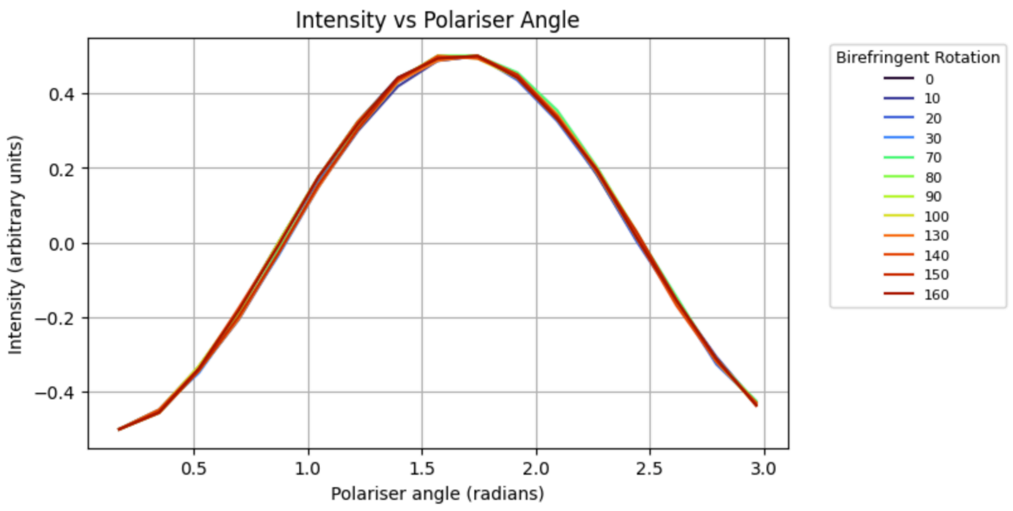

Fig 4

In Fig 4 we can see that there is no dependence of the rotation (as theres nothing there) and we just get a full cosine curve. Note here it was rotated all from 0 to 360.

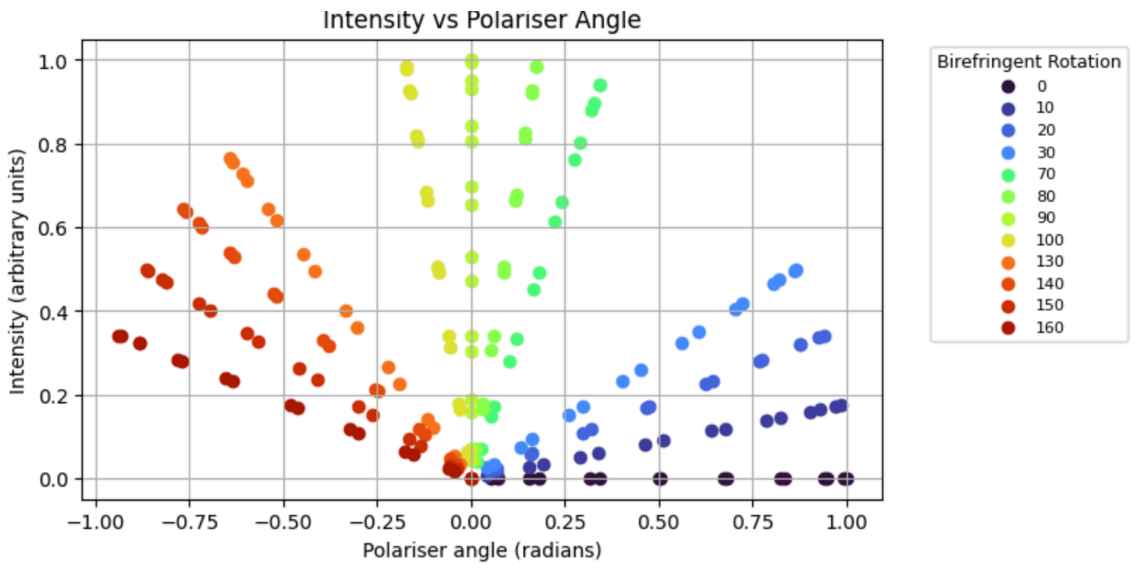

Fig 5

Plotting (in Fig 5.) the angle of birefringence as an axis on the plane and then each recorded data point shows a fan semicircle, showing we get readings correctly, with more of them being at the extremes because of the larger derivative (so more data points around them).

Summary of progress & Outlook

This session was productive and we now have a better idea of what data we want to collect formally for different materials and during data collection we will likely find out more parameters which we want to adjust (such as maybe laser wavelength, birefringent material). From here we can also start to investigate retardence and other properties.

Bibliography

[1] Hecht E., Optics 5th edition, Addison Wesley, Chapter 8, (2017)

Appendix A

Code revision 1:

```
# REVISION 1 : WRITTEN BY LUCAS SARAGOSA Friday 3rd October & On

import numpy as np
import serial
import time
import os
import threading
from tqdm import tqdm
import serial.tools.list_ports
from scipy.optimize import curve_fit
import sys
import matplotlib.pyplot as plt

flagStop = False

# SERIAL PROTOCOL FOR ELL14 AND FAMILY:
# https://www.thorlabs.com/Software/Elliptec/Communications_Protocol/ELL
x%20modules%20protocol%20manual_Issue7.pdf
# note that all ELL14 devices are using FTDI chips so FTD2XX could be used a
```

```
s well.

print("avail devices:")
for port in serial.tools.list_ports.comports():
    print(port)

# CONSTANTS ##########

POLARISER_DEV = "/dev/ttyUSB1" # change these (reset COM ports if neede
d) to 0
BIFRINGENCE_DEV = "/dev/ttyUSB4"
ARDUINO_DEV = "/dev/ttyACM0"

######################

# for multithreaded (redundant for now)
def MT_ExitPoll():
    while True:
        inp = input()
        if inp.strip() == 'q':
            flagStop = True
            break

def Degrees2Hex(deg):
    pulses = int(deg/360*143360)    # # 143360 = 360 deg
    hexPulses = hex(pulses).upper()  #  Hex characters have to be capitals
    return hexPulses[2:]

def Serial2Deg(serialString):
    pos = round((int(serialString.strip()[3:],16)/143360*360),2)
    return pos

def OpenELLDevice(device):
    dev = serial.Serial(device, baudrate=9600, stopbits=serial.STOPBITS_ONE,
timeout=1)
    dev.reset_input_buffer()
    dev.flushInput()
    dev.flushOutput()
    return dev

def OpenArduinoDevice(device):
    dev = serial.Serial(device, baudrate=9600)
    dev.reset_input_buffer()
    dev.flushInput()
    dev.flushOutput()
    return dev

# job meaning the step size the motor rotates between each move forward cm
d
def ELL_SetJog(degrees, device):
    jogStepSize = str(Degrees2Hex(degrees))
    if len(jogStepSize)<4:
        jogStepSize = jogStepSize.zfill(4)
    # 1. serial instruction to set step size
    writeString = '1sj0000'+ str(jogStepSize)
    device.write((writeString).encode('utf-8'))
    time.sleep(0.1)
    if(device.in_waiting > 0):
        serialString = device.readline().decode('ascii')
        print(serialString) # any messages back
    # 2. check its set
    device.write(("1gj" + "\n").encode('utf-8'))
    time.sleep(0.1)
    if(device.in_waiting > 0):
        serialString = device.readline().decode('ascii')
        print('Set jog step: ' + str(round((int(serialString.strip()[3:],16)/143360*36
0),2)) + ' deg\n')

# reset the ell14 to its home position (0 deg)
def ELL_ResetHome(device):
    device.write(('1ho' + '\n').encode('utf-8'))
    time.sleep(0.8) # wait 800ms for motor
```

```
# rotate a single job step.
def ELL_RotateSingle(device, verbose = True):
    device.write(('1fw' + '\n').encode('utf-8'))
    time.sleep(0.05) # 50 ms delay for motor
    if(device.in_waiting > 0 and verbose):
        serialString = device.readline().decode('ascii')
        pos0 = round(Serial2Deg(serialString))
        if pos0 > 143360:
            pos0 = 0
        print('Current position of linear polariser = ' + str(pos0) + ' deg' + '\n')

# rotate device and generate angles to power recieved readings.
def ELL_Rotate(device, ard_device, startDegree, endDegree, degreeStep, bire,
verbose = True):
    global flagStop
    ELL_SetJog(degreeStep, device)
    ELL_SetJog(degreeStep, bire)
    ELL_ResetHome(bire)
    angles = np.arange(startDegree, endDegree, degreeStep)
    allPowers = []
    for idxBire in tqdm(angles):
        ELL_ResetHome(device)
        powerReadings = [] # from ard
        for idx in tqdm(angles):

            if flagStop:
                exit(0)

            ard_device.write('pol'.encode()) # poll data
            pol = ard_device.readline().decode('ascii')
            powerReadings.append(float(pol.strip()))

            # rotate command

            ELL_RotateSingle(device, verbose)
        ELL_RotateSingle(bire, verbose)
        allPowers.append(powerReadings)

    return angles, allPowers

# plot the data from a rotation sequence. pass in the model function to plot ag
ainst (must take in theta, p1, p2 and p3)
def MakePlot(ModelProc, angles, powers, saveDir):
    plt.close('all')
    fig = plt.figure(figsize = (12,6))
    ax = fig.add_subplot(121)
    plt.plot(angles,powers[0],'ro')
    plt.xlabel('Polariser Angle (deg)')
    popt, pcov = curve_fit(ModelProc, angles, powers[0], bounds = ([-5,-5,0],
[5,5,np.pi]))
    Emax, Emin, alpha = popt
    fittingAngles = np.arange(np.nanmin(angles), np.nanmax(angles),1)
    plt.plot(fittingAngles,model_f(fittingAngles, Emax, Emin, alpha),'--b')
    plt.savefig(saveDir+'/DataPlot.png')
    data_out = np.column_stack([angles] + powers)
    np.savetxt(saveDir+'/Data.csv', data_out, delimiter=',', header='angles' + ','.j
oin([f'y_{i}' for i in range(len(powers))]))

# modelling function to overlay onto data and test theoretical equations
def model_f(theta,p1,p2,p3):
  degrees = np.pi/180
  return (p1*np.cos(theta*degrees-p3))**2 + (p2*np.sin(theta*degrees-p3))**
2

################################################################
############################################################

# perform experiment data retrieval
def AsyncRun():
    polariser = OpenELLDevice(POLARISER_DEV)
    bire = OpenELLDevice(BIFRINGENCE_DEV)
    arduino = OpenArduinoDevice(ARDUINO_DEV)

    angs, pows = ELL_Rotate(polariser, arduino, 0, 180, 10, bire, False)
    MakePlot(model_f, angs, pows, "/home/phy3138/Documents/Data_pab/")

    #ELL_Rotate(polariser, arduino, 0, 360, 10)



# kickoff
#bgthread = threading.Thread(target=AsyncRun)
#bgthread.start()
##MT_ExitPoll()

AsyncRun()
```

Appendix B

```
import numpy as np
import pandas as pd
import time
import math
import matplotlib.pyplot as plt

df = pd.read_csv('../dat/GREEN/green_noplate.csv', comment='#')
#df = pd.read_csv('../dat/GREEN/808nm_calib.csv', comment='#')
# err data angles
errs = [170, 120, 60, 50, 110, 40]
plot_as_circle = True

angle_deg = df.iloc[:, 0].values
angle_rad = np.deg2rad(angle_deg)  # 0 to pi

intensities = df.iloc[:, 1:].values
intensities_rad = intensities.T
```

```
fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
cmap = plt.cm.get_cmap('turbo', len(intensities_rad))

ax.set(
    xlabel='Polariser angle (radians)',
    ylabel='Intensity (arbitrary units)',
    title='Intensity vs Polariser Angle'
)

i = 0
for y_intensity in intensities_rad:
    if not np.any(np.isclose(i*10, errs, atol=0.5)):
        angl = math.pi * (1.0/180.0) * float(i) * 10.0
        inten = (y_intensity / np.nanmax(y_intensity))
        if plot_as_circle:
            xs = []
            ys = []
            for ii in inten:
                xs.append(math.cos(angl) * ii)
                ys.append(math.sin(angl) * ii)
            pass
            ax.scatter(xs, ys, label = str(i*10), color=cmap(i))
        else:
            ax.plot(angle_rad, inten, label = str(i*10), color=cmap(i))
    i = i + 1

ax.grid(True)
ax.legend(
    title="Birefringent Rotation",
    bbox_to_anchor=(1.05, 1), loc='upper left',
    fontsize=8, title_fontsize=9
)
plt.show()
```