COMBINED CODEBASE

Automation

Generated: 2026-01-21 22:01:14
Total files: 11

TABLE OF CONTENTS

FILE 1/11: 55617113_QYA.pyw

```
#!/usr/bin/env python3

#imports
import os
# import platform
import sys
import time
from pathlib import Path

import scipy.optimize
from scipy.stats import chisquare
from sklearn.metrics import r2_score

sys.path.append(os.path.join(os.path.dirname(__file__), "lib"))

from ctypes import *
from math import log10 as log
from statistics import mean
import json

import matplotlib
import numpy as np
import pandas as pd
# import rseriesopc as rs
import scipy
from matplotlib import pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.backends.backend_qt5agg import \
    NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure

matplotlib.use("Qt5Agg")
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from lib.windows import QYA_f_win as form_window
# import flux_dict

# import globals
# import qy_window
# from lib.avaspec import *

# from tqdm import tqdm
# print(__file__.split("\\")[-1].split(".")[0])
# head, tail = os.path.split(__file__)
# print(head, tail)

# time.sleep(2)

cwd = os.getcwd()

# if os.path.isfile(cwd+"/avaspecx64.dll"):
#     # print("You are in the right directory!")
#     os.add_dll_directory(cwd)
#     pass
# else:
#     print("You are not in the directory with avaspecx64.dll")
#     raise FileNotFoundError

# time.sleep(1)


# lib = cdll.LoadLibrary("C:\Program Files\IVI Foundation\VISA\Win64\Bin\TLUP_64.dll")
```

```
class MainWindow(QMainWindow, form_window.Ui_MainWindow):
    params_list = ["analysisEndTime",
                   "analysisStartTime",
                   "analysisNumPoints",
                   "analysiswlLineEdit",
                   "manCorr",
                   "zeroLineEdit",
                   "abswlLineEdit",
                   "fluxLineEdit",
                   "fluxwlLineEdit",
                   "concLineEdit",
                   "extincLineEdit",
                   "extincwlLineEdit",
                   "pathLength",
                   "qyLineEdit",
                   "volLineEdit",
                   "kLine",
                   "simTimeLineEdit"
                   ]
    qya_params = []
    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.setWindowTitle("Quantum Yield Analysis Program")
        self.setWindowIcon(QIcon('lib/sun.png'))
#       self.OpenCommBtn.clicked.connect(self.on_OpenCommBtn_clicked)
#       do not use explicit connect together with the on_ notation, or you will get
#       two signals instead of one!
        self.fit_fig = MplCanvas()
        toolbar = NavigationToolbar(self.fit_fig, self)
        self.fitFigLayout.addWidget(self.fit_fig)
        self.fitFigLayout.addWidget(toolbar)
        self.spec_fig = MplCanvas()
        toolbar = NavigationToolbar(self.spec_fig, self)
        self.specFigLayout.addWidget(self.spec_fig)
        self.specFigLayout.addWidget(toolbar)
        self.res_fig = MplCanvas()
        toolbar = NavigationToolbar(self.res_fig, self)
        self.resFigLayout.addWidget(self.res_fig)
        self.resFigLayout.addWidget(toolbar)
        self.flux_dict = flux_dict.all_flux

        if os.path.isfile("lib/qya_params.json"):
            with open('lib/qya_params.json', 'r') as file:
                self.qya_params = json.load(file)
            for i in range(len(self.params_list)):
                if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QLineEdit":
                    self.__dict__[f"{self.params_list[i]}"].setText(self.qya_params[i])
                if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QSpinBox":
                    self.__dict__[f"{self.params_list[i]}"].setValue(int(self.qya_params[i]))
        else:
            print("No init params were found.")

    def closeEvent(self, event):

        quit_msg = "Are you sure you want to exit the program?"
        reply = QMessageBox.question(self, 'Warning',
                        quit_msg, QMessageBox.Yes, QMessageBox.No)

        if reply == QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    ###############################################################################################...
    # GENERAL UI
```

```
@pyqtSlot()
def on_saveParams_clicked(self):
    self.qya_params=[]
    for i in range(len(self.params_list)):
        if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QLineEdit":
            self.qya_params.append(self.__dict__[f"{self.params_list[i]}"].text())
        if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QSpinBox":
            self.qya_params.append(str(self.__dict__[f"{self.params_list[i]}"].value()))
    with open('lib/qya_params.json', 'w') as file:
        json.dump(self.qya_params, file)

    return


 #Print time and message

def print_to_message_box(self, text):
        t = time.localtime()
        current_time = time.strftime("%H:%M:%S", t)
        self.logTextEdit.appendPlainText(f"{current_time}  {text}")
        return

@pyqtSlot()
def on_runCalcBtn_clicked(self):
    ### Get calculation parameters from the fields and check if they are viable
    self.check_file()
    if self.good_file == False:
        self.print_to_message_box("Analysis cancelled due to file.")
        return

    self.check_params()
    params = self.get_params()
    print(params)
    ### Run QY calculation
    qys = self.run_QY_calc(params)
    if self.fluxRadio.isChecked():
        self.print_to_message_box(f"Quantum yield at {params.fluxwl} nm: {100*qys[0]:.3f} %")
        self.print_to_message_box(f"The error of the fit is: {np.sqrt(np.diag(self.pcov)).real*1...
    if self.qyRadio.isChecked():
        self.print_to_message_box(f"Photon flux at {params.fluxwl} nm: {qys[0]:.3e}")
        self.print_to_message_box(f"The error of the fit is: {np.sqrt(np.diag(self.pcov)).real*1...
    return qys

@pyqtSlot()
def on_runSimBtn_clicked(self):
    # Make the simulation prediction and show the plots.

    return

def check_file(self):
    self.good_file = True
    try:
        check = pd.read_csv(self.fileName, sep=' ', header = None,skiprows=2)

    except:
        print("File could not be read.")
        self.good_file = False
        return
    if np.min(check.iloc[1,1:]) < 0:
        qm = QMessageBox
        response = qm.question(self,'', "Some values in the spectrum are below 0. Do you want to...
        if response == qm.Yes:
            self.good_file = True
        elif response == qm.No:
            self.good_file = False
    return
```

```
def check_params(self):
    self.good_params = True

    if self.good_params == True:
        # self.print_to_message_box("Parameters are looking good. Proceeding calculation.")
        pass
    return

@pyqtSlot()
def on_loadLEDBtn_clicked(self):
    try:
        dlg = QDialog(self)
        dlg.resize(250,300)
        dlg.setFixedSize(250,300)
        dlg.setWindowTitle("Load LED data:")
        layout = QVBoxLayout()
        cb1 = QComboBox()
        cb1.addItems(["280","308","340","365","405","430","455"])
        cb1.setCurrentIndex(1)
        cb2 = QComboBox()
        cb2.addItems([f"{x}" for x in range(100,1201,100)])
        cb2.setCurrentIndex(1)
        cb3 = QComboBox()
        cb3.addItems(["Standard 10 mm", "No cuvette", "Standard MeCN", "80 ul flowcell", "80 ul ...
        layout.addWidget(QLabel("Choose wavelength:"))
        layout.addWidget(cb1)
        layout.addWidget(QLabel("Choose power:"))
        layout.addWidget(cb2)
        layout.addWidget(QLabel("Choose Cuvette:"))
        layout.addWidget(cb3)
        b3 = QPushButton("Preview photon flux")
        layout.addWidget(b3)
        preview = QLabel("Preview: N/A")
        layout.addWidget(preview)
        # preview_text = self.get_flux_from_params(cb1.currentText(),cb2.currentText(),cb3.curre...
        preview_text = f"Preview: {self.flux_dict[0]['308']['100']}"
        # preview_text = f"Preview: {self.flux_dict['um45mm3'][{str(cb1.currentText())}][str(cb2...
        b3.clicked.connect(lambda: preview.setText(str(self.flux_dict[cb3.currentIndex()][cb1.cu...
        b1 = QPushButton("Load",dlg)
        b1.clicked.connect(dlg.accept)
        b2 = QPushButton("Cancel",dlg)
        b2.clicked.connect(dlg.reject)
        btn_layout = QHBoxLayout()
        btn_layout.addWidget(b1)
        btn_layout.addWidget(b2)
        layout.addLayout(btn_layout)
        dlg.setLayout(layout)
        dlg.setWindowTitle("Load photon flux")
        dlg.setWindowModality(Qt.ApplicationModal)
        path_lengths = [1,1,1,1,1,1,1,1,1,1,1]
        dlg.accepted.connect(lambda: self.set_flux(f"{float(self.flux_dict[cb3.currentIndex()][c...
        dlg.exec_()
    except:
        self.print_to_message_box("Something went wrong with loading the photon flux.")
    return

def set_flux(self,flux,wl,v = None,pathlength = None):
    self.fluxLineEdit.setText(str(flux))
    self.fluxwlLineEdit.setText(str(wl))
    self.abswlLineEdit.setText(str(wl))
    if v == None:
        pass
    else:
        self.volLineEdit.setText(v)
    if pathlength == None:
        pass
    else:
```

```
            self.pathLength.setText(str(pathlength))
        return

    def get_params(self):
        params = pd.Series(dtype=object)
        params.qy = float(self.qyLineEdit.text())
        params.extinc = float(self.extincLineEdit.text())
        params.extincwl = float(self.extincwlLineEdit.text())
        params.conc = float(self.concLineEdit.text())
        params.flux = float(self.fluxLineEdit.text())
        params.fluxwl = float(self.fluxwlLineEdit.text())
        params.analysiswl = float(self.analysiswlLineEdit.text())
        params.abswl = float(self.abswlLineEdit.text())
        params.volume = float(self.volLineEdit.text())
        params.zeropoint = float(self.zeroLineEdit.text())
        return params

    def run_QY_calc(self,params):
        try:
            calc = QY_analysis()
            if self.qyRadio.isChecked():
                calc.qy = params.qy
            else:
                calc.qy = None
            if self.extincRadio.isChecked():
                calc.extinc = params.extinc
                calc.extincwl = params.extincwl
            if self.concRadio.isChecked():
                calc.start_conc = params.conc
            if self.fluxRadio.isChecked():
                calc.flux = params.flux
            else:
                calc.flux = None
            calc.file_name = self.fileName
            calc.plot = False
            calc.zero_wl = params.zeropoint
            calc.V = params.volume
            calc.LED_current = 300
            calc.LED_wl = params.abswl
            calc.qy_wl = params.analysiswl
            calc.num_points = self.analysisNumPoints.value()
            calc.start_point = int(self.analysisStartTime.value())
            calc.end_point = int(self.analysisEndTime.value())
            calc.path_length = float(self.pathLength.text())
            if self.extincRadio.isChecked():
                qys,t_list = calc.calculate_QY(calc_conc = True)
            else:
                qys,t_list = calc.calculate_QY()
            self.pcov = calc.pcov
            if self.qyRadio.isChecked():
                y_list = calc.nbd_conc_imag(t_list,calc.qy,qys[0],calc.c_1.real,calc.c_1.imag)
                self.Plot_new_spectrum([t_list,calc.data_wl],[y_list,calc.data_i],self.fit_fig,doubl...
                # self.Plot_new_spectrum([t_list,calc.data_wl],[calc.nbd_conc_imag(t_list,calc.qy,qy...
            elif self.fluxRadio.isChecked():
                y_list = calc.nbd_conc_imag(t_list,qys[0],calc.flux,calc.c_1.real,calc.c_1.imag)
                self.Plot_new_spectrum([t_list,calc.data_wl],[y_list,calc.data_i],self.fit_fig,doubl...
                # self.Plot_new_spectrum([t_list,calc.data_wl],[calc.nbd_conc_imag(t_list,qys[0],cal...
            # self.print_to_message_box([calc.data_wl.to_list(),calc.data_i])
            for i in range(len(calc.first_abs)):
                if i == 0:
                    self.Plot_new_spectrum(calc.first_wl,calc.first_abs[i],self.spec_fig,start_x=150...
                else:
                    self.Plot_new_spectrum(calc.first_wl,calc.first_abs[i],self.spec_fig,start_x=150...
            self.x_list = calc.data_wl.to_list()
            self.y_list = calc.data_i

            if self.prodAbsCheck.isChecked():
```

```
qy_prod_guess = 0.5
eps_prod = float(self.epsProdLine.text())
up_scale = 1.02
down_scale = 0.98
if self.qyRadio.isChecked():
    y,x,z = self.prod_abs_correction(calc.qy,calc.start_conc,calc.b,calc.V,calc.N_A,...
    init_qy = calc.qy
if self.fluxRadio.isChecked():
    y,x,z = self.prod_abs_correction(qys[0],calc.start_conc,calc.b,calc.V,calc.N_A,c...
    init_qy = qys[0]
# self.Plot_new_spectrum(self.x_list,z,self.fit_fig,clear=False,label1="Fit with pro...
# print(calc.data_i)
loose_conv = False
tight_conv = False
opt_threshold = 1e-3
qy_grad_up = 1
qy_grad_prod_up = 1
qy_grad_down = 1
qy_grad_prod_down = 1
current_qy = init_qy
current_qy_prod = qy_prod_guess
for attempt in range(100):
    if len(z) == len(calc.data_i):
        corr_matrix = np.corrcoef(calc.data_i,z)
        corr = corr_matrix[0,1]
        loose_r2 = corr**2
        tight_r2 = r2_score(calc.data_i,z)
        if self.qyRadio.isChecked():
            y_up,x_up,z_up = self.prod_abs_correction(calc.qy*up_scale,calc.start_co...
            y_down,x_down,z_down = self.prod_abs_correction(calc.qy*down_scale,calc....
            y_up,x_up,z_up_prod = self.prod_abs_correction(calc.qy,calc.start_conc,c...
            y_down,x_down,z_down_prod = self.prod_abs_correction(calc.qy,calc.start_...
        if self.fluxRadio.isChecked():
            y_up,x_up,z_up = self.prod_abs_correction(current_qy*up_scale,calc.start...
            y_down,x_down,z_down = self.prod_abs_correction(current_qy*down_scale,ca...
            y_up,x_up,z_up_prod = self.prod_abs_correction(current_qy,calc.start_con...
            y_down,x_down,z_down_prod = self.prod_abs_correction(current_qy,calc.sta...

        if loose_r2 > 0.99500:
            loose_conv = True
            loose_qy = current_qy
            loose_qy_prod = current_qy_prod
        if tight_r2 > 0.99500:
            tight_conv = True
            tight_qy = current_qy
            tight_qy_prod = current_qy_prod
        if tight_r2 > 0.9990:
            tight_qy = current_qy
            tight_qy_prod = current_qy_prod
            break
        else:
            tight_conv = False
        if abs(min([qy_grad_up,qy_grad_down])) < opt_threshold:
            qy_opt_complete = True
            opt_qy = current_qy
        else:
            qy_opt_complete = False
        if abs(min([qy_grad_prod_up,qy_grad_prod_down])) < opt_threshold:

            qy_prod_opt_complete = True
            # opt_k = current_k
        else:
            qy_prod_opt_complete = False
        if tight_conv == False or qy_prod_opt_complete == False or qy_opt_complete =...
            ###GET GRADIENTS FOR EACH QY
            # qy_grad =      ((r2_score(calc.data_i,z_up))-     (r2_score(calc.data_...
            # qy_grad_prod = ((r2_score(calc.data_i,z_up_prod))-(r2_score(calc.data_...
```

```
                qy_grad_up =        (np.sum(np.square(np.subtract(calc.data_i,z_up)))   ...
                qy_grad_down =      -(np.sum(np.square(np.subtract(calc.data_i,z_down))))...
                qy_grad_prod_up =   (np.sum(np.square(np.subtract(calc.data_i,z_up_prod)...
                qy_grad_prod_down = -(np.sum(np.square(np.subtract(calc.data_i,z_down_pr...

                print(f"{attempt} grad1: {min([qy_grad_up,qy_grad_down]):.5f}   !  qy1: ...
                if attempt > 100:
                    factor = attempt/100
                else:
                    factor = 1
                if qy_grad_up <= qy_grad_down:
                    current_qy = current_qy*up_scale
                else:
                    current_qy = current_qy*down_scale
                if qy_grad_prod_up <= qy_grad_prod_down:
                    current_qy_prod = current_qy_prod*up_scale
                else:
                    current_qy_prod = current_qy_prod*down_scale


                # new_qy = current_qy * (1+qy_grad/factor)
                # if new_qy > current_qy * up_scale:
                #     current_qy = current_qy * up_scale
                # elif new_qy < current_qy * down_scale:
                #     current_qy = current_qy * down_scale
                # else:
                #     current_qy = new_qy

                # new_qy_prod = current_qy_prod * (1+qy_grad_prod/factor)
                # if new_qy_prod > current_qy_prod * up_scale:
                #     current_qy_prod = current_qy_prod * up_scale
                # elif new_qy_prod < current_qy_prod * down_scale:
                #     current_qy_prod = current_qy_prod * down_scale
                # else:
                #     current_qy_prod = new_qy_prod
                # current_qy = current_qy * (1+qy_grad/factor)
                # current_qy_prod = current_qy_prod * (1+qy_grad_prod/factor)

                y,x,z = self.prod_abs_correction(current_qy,calc.start_conc,calc.b,calc....
            else:
                print("Converged.")
                break
        else:
            print(len(z),len(calc.data_i))
            print("failed")
    self.Plot_new_spectrum(self.x_list,z,self.fit_fig,clear=False,label1="Fit with prod ...
    self.Plot_new_spectrum(self.x_list,np.array(calc.data_i)-np.array(z),self.res_fig,cl...
    if loose_conv:
        self.print_to_message_box(f"Loose convergence was obtained with quantum yields: ...
    if tight_conv:
        self.print_to_message_box(f"Tight convergence was obtained with quantum yields: ...
    else:
        self.print_to_message_box(f"Optimisation did not converge. Last result was: {100...
        self.print_to_message_box(f"b was {calc.b}")
else:
    pass

if self.kCheck.isChecked() and not self.prodAbsCheck.isChecked():  ## Function for fitti...
    k = float(self.kLine.text())
    up_scale = 1.02
    down_scale = 0.98
    if self.qyRadio.isChecked():
        y,x,z = self.after_correction(calc.qy,k,calc.start_conc,calc.b,calc.V,calc.N_A,q...
    elif self.fluxRadio.isChecked():
        y,x,z = self.after_correction(qys[0],k,calc.start_conc,calc.b,calc.V,calc.N_A,ca...
    qy_grad = 1
    k_grad = 1
```

```
opt_threshold = 2e-2
loose_conv = False
tight_conv = False
qy_opt_complete = False
k_opt_complete = False
init_qy = qys[0]
current_qy = init_qy
current_k = k
for attempt in range(100):
    if len(z)==len(calc.data_i):
        corr_matrix = np.corrcoef(calc.data_i, z)
        corr = corr_matrix[0,1]
        loose_r2 = corr**2
        tight_r2 = r2_score(calc.data_i,z)
        if self.qyRadio.isChecked():
            y_up,x_up,z_up = self.after_correction(calc.qy*up_scale,k,calc.start_con...
            y_down,x_down,z_down = self.after_correction(calc.qy*down_scale,k,calc.s...

        if self.fluxRadio.isChecked():
            y_up,x_up,z_up = self.after_correction(current_qy*up_scale,current_k,cal...
            y_down,x_down,z_down = self.after_correction(current_qy*down_scale,curre...
            y_up,x_up,k_up = self.after_correction(current_qy,current_k*up_scale,cal...
            y_down,x_down,k_down = self.after_correction(current_qy,current_k*down_s...
        # print(f"tight r2 value is: {tight_r2:.5f}")

        if loose_r2 > 0.995000:
            loose_conv = True
            loose_qy = current_qy
            loose_k = current_k
        if tight_r2 > 0.995000:
            tight_conv = True
            tight_qy = current_qy
            tight_k = current_k
        if tight_r2 > 0.9999:
            tight_qy = current_qy
            break
        else:
            tight_conc = False
        if abs(qy_grad) < opt_threshold:
            qy_opt_complete = True
            opt_qy = current_qy
        else:
            qy_opt_complete = False
        if abs(k_grad) < opt_threshold:
            k_opt_complete = True
            opt_k = current_k
        else:
            k_opt_complete = False
        if qy_opt_complete == False or k_opt_complete == False or tight_conv == Fals...
            qy_grad =      -(np.sum(np.square(np.subtract(calc.data_i,z_up)))      -...
            if k != 0:
                k_grad =       -(np.sum(np.square(np.subtract(calc.data_i,k_up)))   ...
            else:
                k_grad = 0
            print(f"{attempt}: qy_grad: {qy_grad:.6f} !   qy: {current_qy:.5f} !    ...
            factor = 1
            if attempt > 50:
                factor = attempt/50
            new_qy = current_qy * (1+qy_grad/factor)
            if new_qy > current_qy * up_scale:
                current_qy = current_qy * up_scale
            elif new_qy < current_qy * down_scale:
                current_qy = current_qy * down_scale
            else:
                current_qy = new_qy
            new_k = current_k * (1+k_grad/factor)
            if new_k > current_k * up_scale:
```

```
                            current_k = current_k * up_scale
                        elif new_k < current_k * down_scale:
                            current_k = current_k * down_scale
                        else:
                            current_k = new_k
                        # current_k = k ## This causes the optimisation to only use the fixed va...
                        y,x,z = self.after_correction(current_qy,current_k,calc.start_conc,calc....
                    else:
                        print(f"opt finished at step{attempt}")
                        break

                    """
                    if loose_r2 > 0.99000:
                        loose_conv = True
                        loose_qy = current_qy
                    if tight_r2 > 0.99000:
                        tight_conv = True
                        tight_qy = current_qy
                        break
                    if tight_conv == False:
                        #Correct the qy for a better fit
                        factor = 2
                        corr_value = ((mean(z)/mean(calc.data_i)))
                        if corr_value < 1:
                            corr_corr = 1-factor*(1-corr_value)/(attempt+1)
                        elif corr_value >= 1:
                            corr_corr = 1+factor*(corr_value-1)/(attempt+1)
                        print(f"corr_value is {corr_value}")
                        current_qy = current_qy*corr_value*corr_corr
                        y,x,z = self.after_correction(current_qy,k,calc.start_conc,calc.b,calc.V...
                    """
                else:
                    print("Error...")
                    print(len(z))
                    print(len(calc.data_i))
                    break

            self.print_to_message_box(f"Optimized qy is: {current_qy*100:.4f} %")
            self.print_to_message_box(f"Optimized k value is: {current_k:.5f} 1/s")
            # if tight_qy:
            #     self.print_to_message_box(f"Tight QY is {tight_qy*100:.3f} %")
            if self.fluxRadio.isChecked():
                self.Plot_new_spectrum(x,np.array(y)*calc.b,self.fit_fig,clear=False,label1="Cor...
        else:
            pass
        if not self.kCheck.isChecked() and not self.prodAbsCheck.isChecked():
            #DO SIMPLE FUNCTION OPTIMISATION...

            pass
        # self.Plot_new_spectrum(calc.data_wl,np.array(calc.data_i)-calc.nbd_conc_imag(calc.data...
        # print([x.real for x in np.array(calc.data_i)-calc.nbd_conc_imag(calc.data_wl,qys[0],ca...
        # p,cov = scipy.optimize.curve_fit(lambda: calc.nbd_conc_imag(x,calc.qy,qys[0],calc.c_1....
    except:
        self.print_to_message_box("Calculation failed. Check your parameters.")
        return [1,1]
    return qys

def led_intgrl_correction(self):

    return

def prod_abs_correction(self,qy,start_conc,b,V,N_A,I,t_list,b_prod=0,qy_prod=0,k=0,b_ex = 1):
    # b = float(self.epsReacLine.text())
    b_analysis_reac = float(self.epsReacAnal.text())
    b_analysis_prod = float(self.epsProdAnal.text())

    last_time = t_list.to_list()[-1]
```

```
    time_step = 1
    time_range = np.linspace(0,last_time,int(last_time/time_step))
    c = start_conc
    c_list = [c]
    c_list_final = []
    c_list_final_prod  = []
    c_prod_list = [0]
    if self.kCheck.isChecked():
        k = float(self.kLine.text())
    else:
        k = 0

    for y in range(len(time_range)):
        b_trans = b_ex
        c_trans = c_list[-1]
        b_cis = b_prod
        c_cis = c-c_list[-1]
        q_0 = I
        # q_trans = (c_trans * b_trans)/(c_trans * b_trans + c_cis * b_cis) * q_0 * (1-10**(-c_t...
        # q_cis = (c_cis * b_cis)/(c_trans * b_trans + c_cis * b_cis) * q_0 * (1-10**(-c_trans *...

        c_grad = (((1/V)*(q_0/N_A)*((1-10**(-(c_trans*b_trans+c_cis*b_cis)))/(c_trans*b_trans+c_...
        # c_grad = (-qy*I*(1-10**(-b*c_list[-1])))/(V*N_A) + (qy_prod*I*(1-10**(-b*(c-c_list[-1])...
        c_list.append(c_list[-1]+c_grad)
        c_prod_list.append(c_prod_list[-1]-c_grad)

        # if y%10 == 0:
        #     c_list_final.append(c_list[-1])
    # print(c_trans,b_trans,c_cis,b_cis)
    if abs(c_list[-1]+c_prod_list[-1]-c) > 1e-7:
        print(f"Difference in conc is {c-(c_list[-1]+c_prod_list[-1]):.2e}")
    c_list.pop(-1)
    c_prod_list.pop(-1)
    for i in t_list:
        c_list_final.append(c_list[np.argmin([abs(o-i) for o in time_range])])
        c_list_final_prod.append(c_prod_list[np.argmin([abs(o-i) for o in time_range])])



    return c_list,time_range,np.add(np.array(c_list_final)*b,np.array(c_list_final_prod)*b_analy...

def after_correction(self,qy,k,start_conc,b,V,N_A,I,t_list,b_ex = 1):
    last_time = t_list.to_list()[-1]
    time_step = 0.2
    time_range = np.linspace(0,last_time,int(last_time/time_step))
    # time_seconds = int(self.simTimeLineEdit.text())

    # time_steps = int(time_seconds/time_step)
    # time_range = [x*time_step for x in range(time_steps)]
    c = start_conc
    c_list = [c]
    c_list_final = []
    for y in range(len(time_range)):
        c_grad = (-qy*I*(1-10**(-b_ex*c_list[-1])))/(V*N_A) + k*(c-c_list[-1]))*time_step
        c_list.append(c_list[-1]+c_grad)
        # if y%10 == 0:
        #     c_list_final.append(c_list[-1])
    c_list.pop(-1)
    for i in t_list:
        c_list_final.append(c_list[np.argmin([abs(o-i) for o in time_range])])

    return c_list,time_range,np.array(c_list_final)*b

def Plot_new_spectrum_old(self, x,y,func, draw = True,start_x = 0,end_x = -1):
    func.axes.clear()
    if len(x)>1:
        func.axes.plot(x[0], y[0],label = "Fit")
```

```
        func.axes.plot(x[1], y[1],label = "Data")
    else:
        func.axes.plot(x, y,label = "Fit")

    func.axes.set_xlabel("Irradiation time (s)")
    func.axes.set_ylabel(f"Absorption  @ {self.analysiswlLineEdit.text()} nm")
    func.axes.legend()
    # self.fit_fig.axes.tight_layout()
    if draw == True:
        func.draw()
    else:
        pass

def Plot_new_spectrum(self, x,y,func, draw = True,start_x = 0,end_x = -1,dots = False,double=Fal...
    if clear:
        func.axes.clear()
    else:
        pass
    if typ == "spec":
        func.axes.plot(x[start_x:end_x], y[start_x:end_x],label = label1,color="black",linewidth...
        func.axes.set_xlabel("Wavelength (nm)")
        func.axes.set_ylabel(f"Absorbance")
        func.draw()
        return
    if double:
        print("All instances were lists")

        if dots:
            func.axes.plot(x[1], y[1],label = label2)
            func.axes.scatter(x[0], y[0], label = label1)
        else:
            # func.axes.plot(x[0][start_x:end_x], y[0][start_x:end_x],label = label1)
            func.axes.plot(x[0], y[0],label = label1,color="red")
            func.axes.scatter(x[1], y[1],label = label2,color="black",marker="x")

    else:
        # func.axes.plot(x[start_x:end_x], y[start_x:end_x],label = "Data")
        if dots == True:
            func.axes.plot(x, y,label = label1)
            func.axes.scatter(x, y)
        else:
            func.axes.plot(x[start_x:end_x], y[start_x:end_x],label = label1)

    func.axes.set_xlabel("Irradiation time (s)")
    func.axes.set_ylabel(f"Absorption  @ {self.analysiswlLineEdit.text()} nm")
    func.axes.legend()

    # self.fit_fig.axes.tight_layout()
    if draw == True:
        func.draw()
    else:
        pass

@pyqtSlot()
def on_getFileBtn_clicked(self):
    self.fileName = self.openFileNameDialog("Choose file containing UV-VIS data from the QY expe...
    self.chosenLabel.setText("File chosen: "+ self.fileName.split("/")[-1])
    self.chosenLabel.setToolTip("File chosen: "+ self.fileName)
    self.chosenLabel.setToolTipDuration(30000)
    return

def openFileNameDialog(self,windowTitle):


    options = QFileDialog.Options()
    # options |= QFileDialog.DontUseNativeDialog
    fileName, _ = QFileDialog.getOpenFileName(self,windowTitle, "","All Files (*);;Python Files ...
```

```
        if fileName:
            print(fileName)
        return fileName

    @pyqtSlot()
    def on_runSimBtn_clicked(self):
        ### RUN LINEAR PART FIRST?
        # conc = start_conc - (qy*I*t)/(V*N_A)
        ### THIS IS NON LINEAR PART
        sim = QY_analysis()
        self.sim_time = int(self.simTimeLineEdit.text())
        t_list = np.arange(0,self.sim_time*2)*0.5
        self.b = float(self.extincLineEdit.text())
        sim.b = self.b
        sim.b_ex = self.b
        sim.num_points = self.analysisNumPoints.value()
        s = float(self.concLineEdit.text())
        n=1
        print(self.b*s)
        self.c_1 = (np.log(1 - (np.power(10,(self.b * s)) + 0j )) + 2j * np.pi * n)/(self.b * np.log...
        flux = float(self.fluxLineEdit.text())
        qy = float(self.qyLineEdit.text())
        plt.plot(t_list,sim.nbd_conc_imag(t_list,qy,flux,self.c_1.real,self.c_1.imag)/self.b)
        plt.ylim(bottom=0)
        plt.show()
        return

    @pyqtSlot()
    def on_btnExport_clicked(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self,"QFileDialog.getSaveFileName()","","All Files...
        if fileName:
            self.print_to_message_box(f"Saving to: {fileName}")
            self.save_path = fileName
        else:
            self.print_to_message_box("No filename was selected and therefore the experiment has not...
            return
        try:
            df = pd.DataFrame()
            df["time"] = self.x_list
            df["abs"] = self.y_list
            df.to_csv(f"{fileName}.csv",index=False)
        except:
            self.print_to_message_box("No data could be exported. Perform analysis first.")
        return

class QY_analysis:
    def __init__(self):
        print("Please setup all reaction parameters for the calculation. The parameters can be found...
        #Below is the standard data for the KMP1
        self.file_name = ""
        self.LED_wl = 365 #Wavelength of the LED for irradiation
        self.qy_wl = 340 #Wavelength for the absorption peak of NBD for analysis
        self.start_conc = 9.5426e-06 #Starting concenctration of experiments performed on 21Mar2023
        self.qy = None #0.61 is the QY calculated for KMP1
        self.flux = None #Standard flux set to None, for calibration of Flux
        self.V = 80e-6 #The volume of the flow cell
        self.LED_current = 300 #The current of the LED
        self.N_A = 6.022e+23 #Avogadros number, it's required so don't question it.
        self.corr = 0
        self.zero_wl = 400
        self.num_points = 15
        self.start_point = 0
        self.end_point = -1
        self.path_length = 1
        return
```

```
    def calculate_QY(self,calc_conc = False, start_x = 0, end_x = -1):
        #Some constants

        #First check if the necessary parameters are set
        self.get_current_params()
#         cont = input("These are the current parameters. Are you sure you want to continue? y/n: ")
        cont = "y"
        if cont.lower() != "y":
            print("Cancelling calculation.")
            return
        data = pd.read_csv(self.file_name, sep=' ', header = None,skiprows=2)
        self.zero_col = np.argmin([abs(x-self.zero_wl) for x in data.iloc[0,:]])
        print(self.zero_col)
        if self.corr != 0:
            corr = self.corr
        else:
            corr = -np.mean(data.iloc[1,self.zero_col-5:self.zero_col+5])
        # print(np.min(data.iloc[1,1:]))
        ex_col = data.iloc[0,:].round(0).to_list().index(self.LED_wl)
        wl_col = data.iloc[0,:].round(0).to_list().index(self.qy_wl)
        data_wl = data.iloc[1:,0][self.start_point:self.end_point]
        data_wl = data_wl - data.iloc[self.start_point+1,0] #correction so it always starts at 0
        self.data_wl = data_wl
        data_i = [num + corr for num in data.iloc[1:,wl_col]][self.start_point:self.end_point]
        data_i = (data.iloc[1:,wl_col]-data.iloc[1:,self.zero_col]).to_list()[self.start_point:self....
        self.data_i = data_i
        data_ex = [num + corr for num in data.iloc[1:,ex_col]][self.start_point:self.end_point]
        data_ex = (data.iloc[1:,ex_col]-data.iloc[1:,self.zero_col]).to_list()[self.start_point:self...
        data_t0 = data.iloc[1,1:][self.start_point:self.end_point]
        if calc_conc:
            extinc_col = data.iloc[0,:].round(0).to_list().index(self.extincwl)
            self.start_conc = (data.iloc[1,extinc_col] + corr)/(self.extinc*self.path_length)
            print(f"Calculated starting concentration is: {self.start_conc}")
        else:
            pass
        self.first_abs = []
        self.first_wl = data.iloc[0,2:].to_list()
        for spec_point in np.linspace(self.start_point,self.end_point,10,dtype=int):
            print(spec_point)
            self.first_abs.append(np.array(data.iloc[spec_point,2:].to_list())-data.iloc[:,self.zero...
        self.b = data_i[0] /(self.start_conc*self.path_length)
        self.b_ex = data_ex[0]/(self.start_conc*self.path_length)
        remove_qc = []
        data_c = []
        for dot in range(len(data_i)):
            diff = (1-(data_i[dot]/data_i[0]))*data_i[-1]
            remove_qc.append(diff)
            data_c.append(data_i[dot]-diff)
        data_c = data_i
        self.data_ex = data_ex
        num_points = self.num_points
        if self.flux == None:
            # for i in range(15):
                # params = self.run_qy_calc(data_wl[:i],data_c[:i],self.start_conc)
            params = self.run_qy_calc(data_wl[:num_points],data_c[:num_points],self.start_conc)
        if self.qy == None:
            # for i in range(15):
            #     params = self.run_flux_calc(data_wl[:i],data_c[:i],self.start_conc)
            params = self.run_flux_calc(data_wl[:num_points],data_c[:num_points],self.start_conc)
        x=np.linspace(0,data_wl[self.end_point],400)
        return params , x

    def help(self):
        print(f"The required parameters are listed below. To see the current parameters run .get_cur...
        return

    def get_current_params(self):
```

```
        print(f"Current parameters are:\nData file: {self.file_name}\nIrradiation wavelength: {self....
              f"Measurement wavelength: {self.qy_wl} nm\nStarting concenctration: {self.start_conc} ...
              f"Calculated Quantum Yield: {self.qy}\nCalculated Photon Flux: {self.flux} 1/s\n"+
              f"Flow cell volume: {self.V} l\nLED current: {self.LED_current}")
        return

    def nbd_conc_imag(self,t,qy,I,cr,ci):
        a = (qy*I)/(self.V*self.N_A)
        return (0.434294*np.log(1-(2.71828**(2.30259*self.b_ex*complex(cr,ci)-2.30259*a*self.b_ex*t)...

    def helper_qy(self,x,qy):
        return self.nbd_conc_imag(x,qy,self.flux,self.c_1.real,self.c_1.imag)

    def helper_flux(self,x,I):
        return self.nbd_conc_imag(x,self.qy,I,self.c_1.real,self.c_1.imag)

    def run_qy_calc(self,x_vals,y_vals,start_conc):
        old_x_vals = x_vals
        old_y_vals = y_vals
#         x_vals = old_x_vals[:10]
#         y_vals = old_y_vals[:10]
        if len(x_vals)>1:
            self.b = y_vals[0] /(start_conc*self.path_length)
            s = self.start_conc
            n = 1
            self.c_1 = (np.log(1 - (10**(self.b_ex * s) + 0j )) + 2j * np.pi * n)/(self.b_ex * np.lo...
    #         popt,pcov = scipy.optimize.curve_fit(nbd_conc_imag,x_vals,y_vals,bounds=([0,1e12,1.1*c...
            try:
                popt,pcov = scipy.optimize.curve_fit(self.helper_flux,x_vals,y_vals,p0=[10e14],bound...
                self.pcov = pcov
            except:
                print("Curve fit did not converge. Check your parameters.")

            print(f"PHOTON FLUX PREDICTED TO BE: {popt[0]/self.LED_current:.2e} 1/s @ BASED ON {len(...
            return popt
        return "Not enough datapoints..."

    def run_flux_calc(self,x_vals,y_vals,start_conc):
        old_x_vals = x_vals
        old_y_vals = y_vals
#         x_vals = old_x_vals[:10]
#         y_vals = old_y_vals[:10]
        if len(x_vals)>1:
            self.b = y_vals[0] /start_conc
            s = self.start_conc
            n = 1
            self.c_1 = (np.log(1 - (10**(self.b_ex * s) + 0j )) + 2j * np.pi * n)/(self.b_ex * np.lo...
    #         popt,pcov = scipy.optimize.curve_fit(nbd_conc_imag,x_vals,y_vals,bounds=([0,1e12,1.1*c...
            try:
                popt,pcov = scipy.optimize.curve_fit(self.helper_qy,x_vals,y_vals,p0=[0.5],bounds=([...
                self.pcov = pcov
            except:
                print("Curve fit did not converge. Check your parameters.")

            print(f"QUANTUM YIELD PREDICTED TO BE: {popt[0]:.2e} % @ BASED ON {len(x_vals)} DATA POI...
            return popt
        return "Not enough datapoints..."

    def run_qy_sim(self):
        ### Get data ready for the plot
        return



class Worker(QObject):
    finished = pyqtSignal()
    func = None
```

```
    def run(self):
        self.func()
        self.finished.emit()
        return

class MplCanvas(FigureCanvasQTAgg):
    def __init__(self, parent=None, width=12, height=12, dpi=100):
        fig = plt.figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)


def main():
    app = QApplication(sys.argv)
    app.lastWindowClosed.connect(app.quit)
    app.setApplicationName("QY Calculation Program")
    form = MainWindow()
    form.show()
    app.exec_()

if __name__ == "__main__":
    main()
```

FILE 2/11: 55617116_QYEX.pyw

```
■#!/usr/bin/env python3

#imports
import os
import sys
import time
import pandas as pd
import numpy as np
import rseriesopc as rs
from matplotlib import pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from ctypes import *
from math import log10 as log
import serial
import serial.tools.list_ports
import json
import requests

from lib.windows import QYEX_f2_win as form_window

import globals
from lib.avaspec import *

cwd = os.getcwd()

if os.path.isfile(cwd+"/avaspecx64.dll"):
    # print("You are in the right directory!")
    os.add_dll_directory(cwd)
    pass
else:
    print("You are not in the directory with avaspecx64.dll")
    raise FileNotFoundError

# time.sleep(1)


lib = cdll.LoadLibrary("C:\Program Files\IVI Foundation\VISA\Win64\Bin\TLUP_64.dll")

class MainWindow(QMainWindow, form_window.Ui_MainWindow):
    newdata = pyqtSignal()
    cancel = pyqtSignal()
    start_TBC = pyqtSignal()
    cancel_qy = pyqtSignal()
    stop_disp = pyqtSignal()
    abs_spectrum = pyqtSignal()
    stop_update = pyqtSignal()
    cancelled = False
    first = True
    use_light = False
    stop_dispersion = False
    TBC_started = False
    Spectrum_figure = plt.figure(dpi = 100)


    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.setWindowTitle("Quantum Yield EXperiment Program")
        self.setWindowIcon(QIcon('lib/sun.png'))
```

```
self.showMaximized()
self.IntTimeEdt.setText("{0:.3f}".format(2))
self.NumAvgEdt.setText("{0:d}".format(4000))
self.NumMeasEdt.setText("{0:d}".format(0))
self.StartMeasBtn.setEnabled(False)
self.t0 = time.time()
OnlyInt = QIntValidator()
OnlyInt.setRange(1, 1000)
self.NumAvgEdt.setValidator(OnlyInt)
self.NumMeasEdt.setValidator(OnlyInt)
self.newdata.connect(self.handle_newdata)
self.cancel.connect(self.cancel_meas)
self.cancel_qy.connect(self.set_qy_cancel)
self.start_TBC.connect(self.set_start_TBC)
self.stop_update.connect(self.stop_update_func)
self.get_abs_spectrum = False
self.actionHelpOxy.triggered.connect(self.show_help_oxy)
self.log_name = "default_log.txt"
self.log_path = "logs/default_log_path.txt"
self.time_off_end = 0
self.time_off_total = 0
self.qy_cancelled=False
self.LEDlist.activated.connect(self.update_power)

self.Spectrum_figure = MplCanvas()
self.monitorLayout.addWidget(self.Spectrum_figure)
plt.xlabel("Wavelength [nm]")
plt.ylabel("Intensity")
plt.tight_layout()
plt.close()

self.time_figure = MplCanvas()
self.timeLayout.addWidget(self.time_figure)
plt.xlabel("Wavelength [nm]")
plt.ylabel("Intensity")
plt.tight_layout()
plt.close()

self.abs_figure = MplCanvas()
self.absLayout.addWidget(self.abs_figure)
plt.xlabel("Wavelength [nm]")
plt.ylabel("Intensity")
plt.tight_layout()
plt.close()

self.on_Set_wavelength_btn_clicked()

#connect function for continuous temperature update
self.tempCheck.toggled.connect(self.tempCheck_func)

#load preset samples in case of crash
if os.path.isfile(cwd+"/Valve_Presets.json"):
    with open('Valve_Presets.json', 'r') as file:
        self.Valve_Presets = json.load(file)
    for i in range(1,17):
        self.__dict__['Sample_name_' + str(i)].setText(self.Valve_Presets[i-1])


# read com ports and connect to SF10 pump
self.Pump = False
self.qpod = False

self.on_setPowerBtn_clicked()

# sf10 pump auto connect
self.on_pumpConnect_clicked()
```

```
    #knauer sampler valve auto connect
    self.on_samplerConnect_clicked()

    #qpod1 auto connect
    self.on_qpodConnect_clicked()

    #shelly auto connect
    self.Shelly_IP.setText('192.168.137.85')
    self.on_shellyConnect_clicked()




##########################################################################################################...

# GENERAL UI

def print_to_message_box(self, text):#Print time and message
        t = time.localtime()
        current_time = time.strftime("%H:%M:%S", t)
        self.logTextEdit.append(f"{current_time}  {text}")
        return

def closeEvent(self, event):

    quit_msg = "Are you sure you want to exit the program?"
    reply = QMessageBox.question(self, 'Warning',
                     quit_msg, QMessageBox.Yes, QMessageBox.No)

    if reply == QMessageBox.Yes:
        event.accept()
    else:
        event.ignore()

##########################################################################################################...

#### PART CONTAINING TEMPERATURE CONTROL

@pyqtSlot()
def on_qpodConnect_clicked(self):
    for pinfo in serial.tools.list_ports.comports():
        if pinfo.serial_number == 'D30ALF5KA' and pinfo.vid == 1027: #QPOD 1 is D30ALF5KA. QPOD ...
            self.qpod = serial.Serial(pinfo.device,19200,timeout=3)
            cmd = str("[F1 ID ?]")
            self.qpod.write(cmd.encode())
            print("message sent")
            response = self.qpod.read_until(expected=b"]",size=20).decode("utf-8")
            print(response)
            time.sleep(0.1)
            self.qpod.write(b"[F1 CT -]")
            time.sleep(0.1)
            self.qpod.write(b"[F1 HT -]")
            self.tempOff.setEnabled(True)
            self.tempOn.setEnabled(True)
            self.qpodDisconnect.setEnabled(True)
            self.qpodConnect.setEnabled(False)
    return

@pyqtSlot()
def on_qpodDisconnect_clicked(self):
    try:
        self.qpod.close()
    except:
        self.print_to_message_box("qpod could not disconnect. perhaps it was not connected prope...
    self.qpodDisconnect.setEnabled(False)
    self.tempOff.setEnabled(False)
```

```
        self.tempOn.setEnabled(False)
        self.qpodConnect.setEnabled(True)
        return

    @pyqtSlot()
    def on_tempOn_clicked(self):
        if self.qpod:
            cmd = "[F1 TC +]"
            self.qpod.write(cmd.encode())
            cmd = "[F1 TC ?]"
            self.qpod.write(cmd.encode())
            response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
            if response == "[F1 TC +]":
                self.print_to_message_box("Temperature control turned ON.")
            else:
                self.print_to_message_box("Temperature control could not be turned on. Try again.")
        else:
            self.print_to_message_box("Could not send message. Qpod not connected.")
        return

    @pyqtSlot()
    def on_tempOff_clicked(self):
        if self.qpod:
            cmd = "[F1 TC -]"
            self.qpod.write(cmd.encode())
            cmd = "[F1 TC ?]"
            self.qpod.write(cmd.encode())
            response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
            if response == "[F1 TC -]":
                self.print_to_message_box("Temperature control turned OFF.")
            else:
                self.print_to_message_box("Temperature control could not be turned off. Try again.")
        else:
            self.print_to_message_box("Could not send message. Qpod not connected.")
        return

    @pyqtSlot()
    def on_tempSet_clicked(self):
        dlg = QDialog(self)
        dlg.resize(200,100)
        dlg.setFixedSize(200,100)
        dlg.setWindowTitle("Set target temperature: ")
        layout = QVBoxLayout()
        layout.addWidget(QLabel("Write target temperature: "))
        temp_spin = QSpinBox()
        temp_spin.setMinimum(0)
        temp_spin.setMaximum(100)
        temp_spin.setValue(25)
        temp_spin.setFocus()
        layout.addWidget(temp_spin)
        b1 = QPushButton("Set",dlg)
        b1.clicked.connect(dlg.accept)
        b2 = QPushButton("Cancel",dlg)
        b2.clicked.connect(dlg.reject)
        btn_layout = QHBoxLayout()
        btn_layout.addWidget(b1)
        btn_layout.addWidget(b2)
        layout.addLayout(btn_layout)
        dlg.setLayout(layout)
        dlg.setWindowModality(Qt.ApplicationModal)
        dlg.accepted.connect(lambda: self.set_temp(temp_spin.value()))
        dlg.open()
        temp_spin.setFocus()
        temp_spin.selectAll()

        return
```

```
def set_temp(self,TT):
    if self.qpod:
        cmd = f"[F1 TT S {TT}]"
        self.qpod.write(cmd.encode())
        cmd = "[F1 TT ?]"
        self.qpod.write(cmd.encode())
        response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
        print(response)
        expect = f"[F1 TT {TT:.2f}]"
        if response.strip() == expect:
            self.print_to_message_box(f"Target temperature set to {TT}.")
            self.temp_label.setText(f"T (C°): {TT}")
        else:
            self.print_to_message_box("Target temperature could not be set. Try again.")
    else:
        self.print_to_message_box("Could not send message. Qpod not connected.")

    return

def temp_update(self):
    while self.update_temp == True:
            cmd = "[F1 CT ?]"
            self.qpod.write(cmd.encode())
            response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
            print(response)
            temp = response.split(" ")[-1].replace("]","")
            self.tempCheck.setText(f"Current temp: {temp}")
            time.sleep(5)
    return

def tempCheck_func(self): #this is connected to the toggle of the temperature update checkbox
    if self.qpod:
        if self.tempCheck.isChecked():
            self.update_temp = True
            self.thread_temp = QThread() # this created an additional computing thread for proce...
            self.worker_temp = Worker() # this is a worker that will tell when the job is done
            self.worker_temp.func = self.temp_update #here the job of the worker is defined. it ...
            self.worker_temp.moveToThread(self.thread_temp) #the workers job is moved from the f...
            self.thread_temp.started.connect(self.worker_temp.run) # when the thread is started,...
            self.worker_temp.finished.connect(self.thread_temp.quit) # when the worker is finish...
            self.worker_temp.finished.connect(self.worker_temp.deleteLater)
            self.thread_temp.finished.connect(self.thread_temp.deleteLater)
            self.thread_temp.start() #here the thread is actually started
            print("update started")
        elif self.tempCheck.isChecked() == False:
            self.tempCheck.setText("Current temp: ??")
            self.stop_update.emit()
            print("update stopped")# end temp update
            pass
    else:
        self.print_to_message_box("Qpod not connected.  ")
    return

def stop_update_func(self):
    self.update_temp = False

#### MULTI SAMPLE CONTROL
#### PART CONTAINING CONTROL OF KNAUER SELECTION VALVE AND SF10 PUMP

@pyqtSlot()
def on_samplerConnect_clicked(self):
    for pinfo in serial.tools.list_ports.comports():
        print (pinfo.name, pinfo.serial_number, pinfo.vid)
        if pinfo.serial_number == 'KNE01QKMA' and pinfo.vid == 1027: #Knauer valve
            self.valve = serial.Serial(pinfo.device)
            serialcmd = str(f'POSITION:{5}\r')
            self.valve.write(serialcmd.encode())
```

```
                # serialcmd = str(f'POSITION?\r')
                # self.valve.write(serialcmd.encode())
                # print(self.valve.readline().strip().decode("latin-1"))
            self.valve_connected = True
    return

@pyqtSlot()
def on_Valve_Select_Button_clicked(self):
    # if self.valve_position == int(self.ValveList.currentText()):
    #     return
    self.valve_position = int(self.ValveList.currentText())
    serialcmd = str(f'POSITION:{self.valve_position}\r')
    self.valve.write(serialcmd.encode())
    self.valve.read_until(expected='\r'.encode('UTF-8'))
    time.sleep(0.5)
    serialcmd = str(f'POSITION?\r')
    self.valve.write(serialcmd.encode())
    line = self.valve.read_until(expected='\r'.encode('UTF-8')).decode("latin-1")
    self.valve_position = int(line.split('POSITION:')[1].split('\r')[0])

    self.print_to_message_box(f"Valve position set to {self.valve_position}")

@pyqtSlot()
def on_pumpConnect_clicked(self):
    for pinfo in serial.tools.list_ports.comports():
        print (pinfo.name, pinfo.serial_number, pinfo.vid)
        if pinfo.serial_number == '6' and pinfo.vid == 1027: #SF10 pump
            self.pump_solvent = serial.Serial(pinfo.device)
            serialcmd = str(f'GV\r')
            self.pump_solvent.write(serialcmd.encode())
            print(self.pump_solvent.readline().strip().decode("latin-1"))
            serialcmd = str(f'REMOTEEN vap9 1\r')
            self.pump_solvent.write(serialcmd.encode())
            print(self.pump_solvent.readline().strip().decode("latin-1"))
            serialcmd = str(f'MODE DOSE\r')
            self.pump_solvent.write(serialcmd.encode())
            serialcmd = str(f'START\r')
            self.pump_solvent.write(serialcmd.encode())
            serialcmd = str(f'STOP\r')
            self.pump_solvent.write(serialcmd.encode())
            print(self.pump_solvent.readline().strip().decode("latin-1"))
            print("Pump connected")
            self.Pump_button.setEnabled(True)
            self.Pump = True
    return

@pyqtSlot()
def on_Load_Button_clicked(self):
    rate = self.Pump_flowrate.value() # flow rate in ml/min
    vol = self.Load_Volume.value() #in ml
    self.print_to_message_box(f"Loading {vol} ml at {rate} ml/min, valve {self.valve_position}")
    serialcmd = str(f'MODE DOSE\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'SETFLOW {rate}\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'SETDOSE {vol}\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'SETREG 3.0\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'START\r')
    self.pump_solvent.write(serialcmd.encode())
    time.sleep(60*vol/rate) #in seconds
    serialcmd = str(f'STOP\r')
    self.pump_solvent.write(serialcmd.encode())
    return

@pyqtSlot()
```

```
def on_Clean_Button_clicked(self):
    if self.Knauer:
        self.valve_position = 16
        serialcmd = str(f'POSITION:{self.valve_position}\r')
        self.valve.write(serialcmd.encode())
    if self.SF10:
        rate = 1 # flow rate in ml/min
        vol = self.Clean_Volume.value() #in ml
        self.print_to_message_box(f"Cleaning {vol} ml at {rate} ml/min, valve {self.valve_positi...
        serialcmd = str(f'MODE DOSE\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETFLOW {rate}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETDOSE {vol}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETREG 3.0\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'START\r')
        self.pump_solvent.write(serialcmd.encode())
        time.sleep(60*vol/rate) #in seconds
        serialcmd = str(f'STOP\r')
        self.pump_solvent.write(serialcmd.encode())
    else:
        self.print_to_message_box("No pump connected")

@pyqtSlot()
def on_Save_Valve_Presets_Button_clicked(self):
    for i in range(1,17):
        self.Valve_Presets[i-1] = self.__dict__['Sample_name_' + str(i)].text()
    with open('Valve_Presets.json', 'w') as file:
        json.dump(self.Valve_Presets, file)

@pyqtSlot()
def on_Clear_Valve_Presets_Button_clicked(self):
    for i in range(1,17):
        self.__dict__['Sample_name_' + str(i)].clear()


#### SHELLY POWER PLUG CONTROL
@pyqtSlot()
def on_shellyConnect_clicked(self):
    # self.Shelly_on.setEnabled(True)
    url = f'http://{self.Shelly_IP.text()}/rpc/Switch.GetStatus?id=0'
    print(url)
    try:
        # Make a GET request to the Shelly device's status URL
        response = requests.get(url,timeout=3)
        # Check for successful response
        if response.status_code == 200:
            print(response.json()) # Parse the JSON response and check for output
            if response.json().get('output'):
                self.Shelly_off.setEnabled(True)
            else:
                self.Shelly_on.setEnabled(True)
    except Exception as e:
        print(f"An error occurred: {e}")
    return

@pyqtSlot()
def on_Shelly_on_clicked(self):
    try:
        response = requests.get(f'http://{self.Shelly_IP.text()}/relay/0?turn=on',timeout=2) #tu...
        if response.json().get('ison') == True:
            self.Shelly_on.setEnabled(False)
            self.Shelly_off.setEnabled(True)
            self.plugLabel.setText("Plug status: ON")
    except:
```

```
            self.print_to_message_box("Shelly smart plug disconnected. Check if mobile hotspot is on...

    @pyqtSlot()
    def on_Shelly_off_clicked(self,check=True):
        if check:
            off_msg = "Are you sure you want to turn off the power supply for the lamp? You will not...
            reply = QMessageBox.question(self, 'Warning', off_msg, QMessageBox.Yes, QMessageBox.No)
            if reply == QMessageBox.No:
                return
        else:
            pass
        try:
            #turn off shelly device
            response = requests.get(f'http://{self.Shelly_IP.text()}/relay/0?turn=off',timeout=2)
            if response.json().get('ison') == False:
                self.Shelly_on.setEnabled(True)
                self.Shelly_off.setEnabled(False)
                self.plugLabel.setText("Plug status: OFF")
        except:
            self.print_to_message_box("Shelly smart plug disconnected. Check if mobile hotspot is on...


    #### PART CONTAINING PUMP CONTROL
    @pyqtSlot()
    def on_Pump_button_clicked(self):
        rate = 1000 # flow rate in ul/min
        vol = self.Pump_vol_Box.value() #in ul
        self.print_to_message_box(f"Pushing {vol} ul at {rate} ul/min")
        serialcmd = str(f'MODE DOSE\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETFLOW {rate/1000}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETDOSE {vol/1000}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETREG 3.0\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'START\r')
        self.pump_solvent.write(serialcmd.encode())
        time.sleep(60*vol/rate) #in seconds
        serialcmd = str(f'STOP\r')
        self.pump_solvent.write(serialcmd.encode())
        return

#### PART CONTAINING LED SWITCH AUTOMATION
    @pyqtSlot()
    def update_power(self):
        time.sleep(0.1)
        try:
            upHandle = c_long(int(self.LEDlist.currentText().split(".")[0]))
            currentSetpoint = c_double(0)
            ret = lib.TLUP_getLedOutputState(upHandle)
            # print(ret)
            if ret == 1:
                lib.TLUP_getLedCurrentSetpoint(upHandle,0, byref(currentSetpoint))
                self.LEDcurrentpower.setText(f"Current power: {currentSetpoint.value}")
        except:
            self.print_to_message_box("No LED chosen.")
        return

    @pyqtSlot()
    def on_connectLED_clicked(self):
        self.print_to_message_box("Trying to connect to LEDs.")
        deviceCount = c_uint32()
        lib.TLUP_findRsrc(0,byref(deviceCount))
        if deviceCount.value > 0:
            self.print_to_message_box("Number of upSeries devices found: " + str(deviceCount.value))
        else:
```

```
            self.print_to_message_box("No upSeries devices found.")
            return
        # print()
        modelName = create_string_buffer(256)
        serialNumber = create_string_buffer(256)
        lib.TLUP_getRsrcInfo(0, 0, modelName, serialNumber, 0, 0)
        # print("Connecting to this device:")
        self.print_to_message_box(f"Model name: {modelName.value.decode()}, Serial number: {serialNu...
        # print()
        self.upNames = [0 for i in range(deviceCount.value)]
        self.upHandles = [0 for i in range(deviceCount.value)]

        for i in range(deviceCount.value):
            # print(i)
            #Initializing the first connected upSeries device.
            self.upNames[i] = create_string_buffer(256)
            lib.TLUP_getRsrcName(0, i, self.upNames[i])
            self.upHandles[i]=c_int(0)
            res=lib.TLUP_init(self.upNames[i].value, 0, 0, byref(self.upHandles[i]))
            # print(f"uphandle is: {self.upHandles[i] , self.upHandles[i].value} for {self.upNames[i...
        wls = []

        for i in self.upHandles:
            currentSetpoint = c_double()
            LEDName = create_string_buffer(256)
            LEDSerialNumber = create_string_buffer(256)
            LEDCurrentLimit = c_double()
            LEDForwardVoltage = c_double()
            LEDWavelength = c_double(0)
            lib.TLUP_getLedInfo(i, LEDName, LEDSerialNumber, byref(LEDCurrentLimit),
                        byref(LEDForwardVoltage), byref(LEDWavelength))
            wls.append(f"{i.value}. {LEDWavelength.value}")
        for l in range(self.LEDlist.count()):
            self.LEDlist.removeItem(0)
        self.LEDlist.addItem("Select LED...")
        self.LEDlist.addItems([f"{x} nm" for x in wls])
        self.connectLED.setEnabled(False)
        self.disconnectLED.setEnabled(True)
        return

    @pyqtSlot()
    def on_disconnectLED_clicked(self):
        try:
            for handle in self.upHandles:
                lib.TLUP_close(handle)
            self.connectLED.setEnabled(True)
            self.disconnectLED.setEnabled(False)
        except:
            self.print_to_message_box("No LEDs connected.")
        return

    @pyqtSlot()
    def on_setLED_clicked(self):
        try:
            upHandle = c_long(int(self.LEDlist.currentText().split(".")[0]))

            print(lib.TLUP_setLedUseNonThorlabsLed(upHandle, 1))
            if self.current_power < 20:
                self.current_power = 20
            if self.current_power > 1200:
                self.current_power = 1200

            currentSetpoint = c_double(float(self.current_power*0.001)) #in mA
            lib.TLUP_setLedCurrentSetpoint(upHandle,currentSetpoint)
            time.sleep(0.5)
            print(lib.TLUP_switchLedOutput(upHandle,1))
            print("Switch LED on.")
```

```
        self.LED_stat_text.setText(f"LED status: ON @ {float(self.LEDpower.value())}")
    except:
        print("LED not available.")
    return

@pyqtSlot()
def on_LEDoff_clicked(self): #turns LED OFF
    try:
        upHandle = c_long(int(self.LEDlist.currentText().split(".")[0]))
        # print(lib.TLUP_setLedUseNonThorlabsLed(upHandle, 1))
        lib.TLUP_switchLedOutput(upHandle,0)
        self.LED_stat_text.setText("LED status: OFF")
        # print("Switch LED off.")
    except:
        print("You didnt choose an LED!!!")
    return


#########################################################################################################...

# SPECTROMETER CONTROL BELOW

@pyqtSlot()
def on_log_file_btn_clicked(self):
    text, ok = QInputDialog().getText(self, "QInputDialog().getText()",
                                      "Log to:", QLineEdit.Normal,
                                      "Filename")
    if ok and text:
        self.log_name = text
        self.log_name_label.setText("Saving to: "+text)
    #something about open dialog
    return

@pyqtSlot()
def on_OpenCommBtn_clicked(self):
    try:
        ret = AVS_Init(0)
        # QMessageBox.information(self,"Info","AVS_Init returned:  {0:d}".format(ret))
        ret = AVS_GetNrOfDevices()
        # QMessageBox.information(self,"Info","AVS_GetNrOfDevices returned:  {0:d}".format(ret))
        req = 0
        mylist = AvsIdentityType * 1
        ret = AVS_GetList(75, req, mylist)
        serienummer = str(ret[1].SerialNumber.decode("utf-8"))
        if serienummer:
            QMessageBox.information(self,"Info","Found spectrometer with Serialnumber: " + serie...
        else:
            self.print_to_message_box("Could not find spectrometer. Try again.")
            return
        globals.dev_handle = AVS_Activate(ret[1])
        # QMessageBox.information(self,"Info","AVS_Activate returned:  {0:d}".format(globals.dev...
        devcon = DeviceConfigType
        reqsize = 0
        ret = AVS_GetParameter(globals.dev_handle, 63484, reqsize, devcon)
        globals.pixels = ret[1].m_Detector_m_NrPixels
        # print(f'length of globals pixels in: {globals.pixels}')
        ret = AVS_GetLambda(globals.dev_handle,globals.wavelength)
        x = 0
        self.wavelength = np.array(ret[:globals.pixels])
        np_round_to_tenths = np.around(self.wavelength, 1)
        globals.wavelength = list(np_round_to_tenths)

        self.measconfig = MeasConfigType
        self.measconfig.m_StartPixel = 0
        self.measconfig.m_StopPixel = globals.pixels - 1
        self.measconfig.m_IntegrationTime = 0
        self.measconfig.m_IntegrationDelay = 0
```

```
            self.measconfig.m_NrAverages = 0
            self.measconfig.m_CorDynDark_m_Enable = 1  # nesting of types does NOT work!!
            self.measconfig.m_CorDynDark_m_ForgetPercentage = 100
            self.measconfig.m_Smoothing_m_SmoothPix = 2
            self.measconfig.m_Smoothing_m_SmoothModel = 0
            self.measconfig.m_SaturationDetection = 0
            self.measconfig.m_Trigger_m_Mode = 0
            self.measconfig.m_Trigger_m_Source = 0
            self.measconfig.m_Trigger_m_SourceType = 0
            self.measconfig.m_Control_m_StrobeControl = 0
            self.measconfig.m_Control_m_LaserDelay = 0
            self.measconfig.m_Control_m_LaserWidth = 0
            self.measconfig.m_Control_m_LaserWaveLength = 0.0
            self.measconfig.m_Control_m_StoreToRam = 0

            if globals.pixels != 0:
                self.StartMeasBtn.setEnabled(False)
                self.StopMeasBtn.setEnabled(True)
                self.OpenCommBtn.setEnabled(False)
                self.CloseCommBtn.setEnabled(True)
                self.getLightBtn.setEnabled(True)
                self.loadRefBtn.setEnabled(True)
        except:
            self.print_to_message_box("No device found.")

        return

    @pyqtSlot()
    def on_CloseCommBtn_clicked(self):
        callbackclass.callback(self, 0, 0)
        self.StartMeasBtn.setEnabled(False)
        self.StopMeasBtn.setEnabled(False)
        self.OpenCommBtn.setEnabled(True)
        self.CloseCommBtn.setEnabled(False)
        return

        FPGAver = bytes(VERSION_LEN)
        FWver = bytes(VERSION_LEN)
        DLLver = bytes(VERSION_LEN)
        ret = AVS_GetVersionInfo(globals.dev_handle, FPGAver, FWver, DLLver)
        FPGAver = ret[0]
        FWver = ret[1]
        DLLver = ret[2]
        QMessageBox.information(self,"Info","FPGA version: {FPGA} \nFirmware version: {FW} \nDLL ver...
                                .format(FPGA=FPGAver.value.decode('utf-8'),
                                        FW=FWver.value.decode('utf-8'),
                                        DLL=DLLver.value.decode('utf-8')))
        return

    @pyqtSlot()
    def on_StartMeasBtn_clicked_old(self):
        self.StartMeasBtn.setEnabled(False)
        ret = AVS_UseHighResAdc(globals.dev_handle, True)
        self.measconfig.m_IntegrationTime = float(self.IntTimeEdt.text()) #set exposure
        self.measconfig.m_NrAverages = int(self.NumAvgEdt.text()) #set number of averages
        ret = AVS_PrepareMeasure(globals.dev_handle, self.measconfig)
        timestamp = 0
        nummeas = int(self.NumMeasEdt.text())
        self.cancelled = False

        scans = 0

        if nummeas == 0:
            nummeas=100000
        while (scans < nummeas):
            if (self.cancelled == False):
```

```
            ret = AVS_Measure(globals.dev_handle, 0, 1)
            dataready = False
            while (dataready == False):
                dataready = (AVS_PollScan(globals.dev_handle) == True)
                time.sleep(0.001)
            if dataready == True:
                ret = AVS_GetScopeData(globals.dev_handle, timestamp, globals.spectraldata)
                timestamp = ret[0]
                x = 0
                for x in range(globals.pixels): # 0 through 2047
                    globals.spectraldata[x] = int(ret[1][x])
                globals.spectraldata = [globals.spectraldata[x] for x in range(globals.pixels)]

                scans = scans + 1
                self.time_spectrum = time.time()
                self.newdata.emit()
                time.sleep(0.3)

        else:
            break
    self.StartMeasBtn.setEnabled(True)
    return

@pyqtSlot()
def on_StartMeasBtn_clicked(self): #activates when the button "Start Measurement" is clicked
    self.first = False ### THIS MUST BE FALSE OTHERWISE THE START MEASUREMENT BUTTON WILL OVERWR...
    try:
        if self.thread.isRunning():
            print("Shutting down running thread.")
            self.thread.terminate()
            time.sleep(1)
        else:
            print("No thread was running.")
    except:
        print("Didn't find thread.")
    self.thread = QThread() # this created an additional computing thread for processes, so the ...
    self.worker = Worker() # this is a worker that will tell when the job is done
    self.worker.func = self.on_StartMeasBtn_clicked_old #here the job of the worker is defined. ...
    self.worker.moveToThread(self.thread) #the workers job is moved from the frontend to the thr...
    self.thread.started.connect(self.worker.run) # when the thread is started, the worker runs
    self.worker.finished.connect(self.thread.quit) # when the worker is finished, then the threa...
    self.worker.finished.connect(self.worker.deleteLater)
    self.thread.finished.connect(self.thread.deleteLater)
    self.thread.start() #here the thread is actually started
    return

@pyqtSlot()
def on_StopMeasBtn_clicked(self):
    self.cancel.emit()
    time.sleep(1)

    return

@pyqtSlot()
def cancel_meas(self):
    self.cancelled = True
    return

@pyqtSlot()
def handle_newdata(self):
    # self.label_5.setText(
    #     f"Estimated time required: {(float(self.IntTimeEdt.text())*int(self.NumAvgEdt.text())*...
    try:
        self.time_off_total += self.time_off_end
        self.new_time =  self.time_spectrum - self.t0 - self.time_off_total

        self.Spectrum_figure.axes.cla()
```

```
            self.Spectrum_figure.axes.plot(globals.wavelength,globals.spectraldata)
            self.Spectrum_figure.axes.set_xlabel("Wavelength [nm]")
            self.Spectrum_figure.axes.set_ylabel("Intensity")
            self.Spectrum_figure.draw()

            self.save_data(save_pdf=True)
        except:
            print("new data was not handled")
        return

    @pyqtSlot()
    def save_data(self,save_pdf = False):
        try:
            if self.use_light == True:
                try:
                    globals.spectral_minus_light = [log(globals.light_spectrum[x] / globals.spectral...
                except:
                    globals.spectral_minus_light = np.zeros(globals.pixels)
                # globals.spectral_minus_light = [log(globals.light_spectrum[x] / globals.spectralda...

                # globals.spectral_minus_light = [0.0 - globals.spectraldata[x] + globals.light_spec...
                if self.first:
                    with open(self.log_path,'w') as f:
                        f.write(f"Temperature: {self.Temperature_box.value()} C, Exposure time: {flo...
                        f.write('light '  + 'spectrum ' + ' '.join([str(globals.light_spectrum[x]) f...
                        f.write('0 '  + '0 ' + ' '.join([str(globals.wavelength[x]) for x in range(g...
                        f.write('0 ' +str(self.LEDpower.value())+' ' + ' '.join([str(globals.spectra...
                else:
                    with open(self.log_path,'a') as f:
                        f.write(str(self.new_time)+' '+ f"{self.TBC_started}"+ ' ' + ' '.join([str(g...
                if save_pdf == True:
                    self.abs_figure.axes.cla()
                    self.abs_figure.axes.plot(globals.wavelength[:2048], [globals.spectral_minus_lig...
                    self.abs_figure.axes.set_xlim(left=self.Min_wl_box.value(), right=self.Max_wl_bo...
                    self.abs_figure.axes.set_xlabel("Wavelength [nm]")
                    self.abs_figure.axes.set_ylabel("Absorbance [OD]")
                    self.abs_figure.draw()

                    # plot new dynamics
                    self.Plot_new_dynamics()
                else:
                    print("you didnt choose reference light spectrum")
                self.first = False
            except:
                print("data was not saved")
            return

################################################## PLOT Values real time. From TTAUC script #########...

    @pyqtSlot()
    def on_Set_wavelength_btn_clicked(self): #set values to plot and background value to subtract
        self.wavelengths_int = [int(eval(i)) for i in self.Wavelength_line.text().split(",")]
        self.background = int(self.Background_line.text())
        return

    @pyqtSlot()
    def Plot_new_dynamics(self): #also removes absorption values and can be used to calculate thresh...
        try:
            pddata = pd.read_csv(self.log_path, sep=' ', header=2) #2 for two lines of metadata and ...
            meta = 5 #how many places reserved for metadata in the columns
            wavelengths_pd = np.array(list(pddata)[meta:])
            wavelength_axis = np.around(wavelengths_pd.astype(float),1)
            bg_index = (np.abs(wavelength_axis - self.background)).argmin() + meta
            Data_points = len(pddata.iloc[:,0])
            Dynamics = np.zeros(Data_points)
            Time = np.zeros(Data_points)
            self.time_figure.axes.clear()
```

```
            for wl in range(len(self.wavelengths_int)): # pl vs time plot
                wl_index = (np.abs(wavelength_axis - self.wavelengths_int[wl])).argmin() + meta
                for i in range(Data_points):
                    Time[i] = np.array(pddata.iloc[i,0])
                    Dynamics[i] = pddata.iloc[i,wl_index] - pddata.iloc[i,bg_index]
                Time_axis = np.around(Time.astype(float),2)
                # self.time_figure.axes.cla()
                self.time_figure.axes.plot(Time_axis,Dynamics,label=wavelength_axis[wl_index-meta])

            self.time_figure.axes.legend()
            self.time_figure.axes.set_xlabel("Time [s]")
            self.time_figure.axes.set_ylabel("Absorbance")
            self.time_figure.draw()
        except:
            print("no dynamics plotted")
            return
        return

##########################################################################################################...

    @pyqtSlot()
    def on_setPowerBtn_clicked(self): #sets the power that the LED will use when turned on
        try:
            self.current_power = float(self.LEDpower.value())
            self.LEDcurrentpower.setText(f"Current power: {self.current_power}")
        except:
            self.print_to_message_box("Could not set power.")
        return

    @pyqtSlot()
    def on_showRefBtn_clicked(self): #shows the reference spectum in the raw spectrum plot
        self.Spectrum_figure.axes.cla()
        self.Spectrum_figure.axes.plot(globals.wavelength[0:2048],globals.light_spectrum[0:2048],lab...
        self.Spectrum_figure.axes.set_xlabel("Wavelength [nm]")
        self.Spectrum_figure.axes.set_ylabel("Intensity")
        self.Spectrum_figure.figure.tight_layout()
        self.Spectrum_figure.draw()
        return

    @pyqtSlot()
    def on_saveRefBtn_clicked(self): #saves the reference spectrum
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self,"QFileDialog.getSaveFileName()","","Text File...
        if fileName:
            self.print_to_message_box(f"Saving reference spectrum to: {fileName}")
        else:
            self.print_to_message_box("No filename was selected and therefore the reference spectrum...
            return
        with open(fileName,"w") as f:
            f.write(' '.join([str(x) for x in globals.wavelength[0:2048]])+'\n'+' '.join(str(x) for ...

        return

    @pyqtSlot()
    def on_loadRefBtn_clicked(self): #loads a reference spectrum previously saved
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getOpenFileName(self,"QFileDialog.getSaveFileName()","","Text File...

        if fileName:
            self.print_to_message_box(f"Loading reference spectrum from: {fileName}")
        else:
            self.print_to_message_box("No filename was selected and therefore the reference spectrum...
            return
        try:
            pddata_ref = pd.read_csv(fileName, sep=' ', header=0)
            globals.light_spectrum = np.array(pddata_ref.iloc[0,:])
            self.use_light = True
```

```
    except:
        self.print_to_message_box("The reference spectrum could not be loaded...")
        return
    self.showRefBtn.setEnabled(True)
    self.saveRefBtn.setEnabled(True)
    self.StartMeasBtn.setEnabled(True)
    return

@pyqtSlot()
def on_getLightBtn_clicked(self):
    self.StartMeasBtn.setEnabled(False)
    try:
        self.on_lampOnBtn_clicked()
        time.sleep(0.5)
    except:
        print("Lamp is not connected...")
    ret = AVS_UseHighResAdc(globals.dev_handle, True)
    measconfig = MeasConfigType
    measconfig.m_StartPixel = 0
    measconfig.m_StopPixel = globals.pixels - 1
    measconfig.m_IntegrationTime = float(self.IntTimeEdt.text())
    measconfig.m_IntegrationDelay = 0
    measconfig.m_NrAverages = int(1000/float(self.IntTimeEdt.text()))
    measconfig.m_CorDynDark_m_Enable = 1  # nesting of types does NOT work!!
    measconfig.m_CorDynDark_m_ForgetPercentage = 0
    measconfig.m_Smoothing_m_SmoothPix = 0
    measconfig.m_Smoothing_m_SmoothModel = 0
    measconfig.m_SaturationDetection = 0
    measconfig.m_Trigger_m_Mode = 0
    measconfig.m_Trigger_m_Source = 0
    measconfig.m_Trigger_m_SourceType = 0
    measconfig.m_Control_m_StrobeControl = 0
    measconfig.m_Control_m_LaserDelay = 0
    measconfig.m_Control_m_LaserWidth = 0
    measconfig.m_Control_m_LaserWaveLength = 0.0
    measconfig.m_Control_m_StoreToRam = 0
    ret = AVS_PrepareMeasure(globals.dev_handle, measconfig)
    nummeas = int(self.NumMeasEdt.text())
    self.cancelled = False
    timestamp = 0
    ret = AVS_Measure(globals.dev_handle, 0, 1)
    dataready = False
    while (dataready == False):
        dataready = (AVS_PollScan(globals.dev_handle) == True)
        time.sleep(0.001)
    if dataready == True:
        ret = AVS_GetScopeData(globals.dev_handle, timestamp, globals.spectraldata )
        wl = AVS_GetLambda(globals.dev_handle, globals.wavelength)
        timestamp = ret[0]
        x = 0
        while (x < globals.pixels): # 0 through 2047
            globals.spectraldata[x] = ret[1][x]
            x += 1

    light = [globals.spectraldata[x] for x in range(globals.pixels)]
    globals.light_spectrum = light

    self.use_light = True
    try:
        time.sleep(0.5)
        self.on_lampOffBtn_clicked()
    except:
        print("Lamp is still not connected...")
    self.StartMeasBtn.setEnabled(True)
    self.saveRefBtn.setEnabled(True)
    self.showRefBtn.setEnabled(True)
```

```
    self.on_showRefBtn_clicked() #show ref spectrum

    return

@pyqtSlot()
def on_laserOnBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 7, 1) #OPENS CHANNEL 1 WHICH CORRESPONDS TO DO8
        self.channel1_text.setText("Laser status: On")
    except:
        self.channel1_text.setText("ERROR")
    return

@pyqtSlot()
def on_laserOffBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 7, 0) #CLOSES CHANNEL 1 WHICH CORRESPONDS TO DO8
        self.channel1_text.setText("Laser status: Off")
    except:
        self.channel1_text.setText("ERROR")
    return

@pyqtSlot()
def on_lampOnBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 3, 1) #OPENS CHANNEL 2 WHICH CORRESPONDS TO DO5
        self.channel2_text.setText("Lamp status: On")
        # self.print_to_message_box("Lamp shutter opened.")
    except:
        self.channel2_text.setText("ERROR")
    return

@pyqtSlot()
def on_lampOffBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 3, 0) #CLOSES CHANNEL 2 WHICH CORRESPONDS TO DO5
        self.channel2_text.setText("Lamp status: Off")
        # self.print_to_message_box("Lamp shutter closed.")
    except:
        self.channel2_text.setText("ERROR")
    return

@pyqtSlot()
def runQY(self):
    self.startQYBtn.setEnabled(False)
    self.qyRunning.setText("Experiment running...")
    if self.NumMeasEdt.text() != "1":
        self.NumMeasEdt.setText("1")
    self.t0 = time.time()
    self.time_off_total = 0
    self.time_off_start = 0
    self.time_off_end = 0
    self.time_spectrum = 0
    meas_interval = int(self.intervalLine.text())
    TBC_interval = int(self.TBC_interval_box.value())
    blank_measurements = self.Blank_meas_num.value()
    self.qy_cancelled = False
    self.TBC_started = False
    if self.qyPointsSpinBox.value() > 0:
        num_points = self.qyPointsSpinBox.value()
    else:
        num_points = 100000
    for i in range(num_points):
        if self.qy_cancelled == True: #Cancel if Stop button was pressed
            self.qyRunning.setText("Experiment not running.")
            self.on_lampOffBtn_clicked()
            self.on_LEDoff_clicked()
```

```
            self.startQYBtn.setEnabled(True)
            self.print_to_message_box("QY experiment stopped (manual).")
            return

        elif self.TBC_started == True: #Begin backconversion if button was pressed
            self.time_off_start = time.time()
            self.on_lampOnBtn_clicked()
            time.sleep(0.1)
            self.on_StartMeasBtn_clicked_old(scans=1)
            time.sleep(0.2)
            self.on_lampOffBtn_clicked()
            time.sleep(0.1)
            # self.time_off_end = time.time()-self.time_off_start

            self.time_off_end = 0 # This has to be 0 to obtain the proper time of the back conve...
            try:
                TBC_interval_new = int(self.TBC_interval_box.value())
            except:
                TBC_interval_new = TBC_interval
                pass
            for b in range(int(TBC_interval_new)):
                time.sleep(1)
                if self.qy_cancelled == True:
                    break

        else: #continue with isomerisation measurement
            self.time_off_start = time.time()
            time.sleep(0.1)
            self.on_lampOnBtn_clicked()
            time.sleep(0.1)
            self.on_StartMeasBtn_clicked_old(scans=1)
            time.sleep(0.1)
            self.on_lampOffBtn_clicked()
            time.sleep(0.1)
            if i > blank_measurements: #dont turn on LED for initial set of measurements
                self.on_setLED_clicked()
            self.time_off_end = time.time()-self.time_off_start
            print(f"Time off is: {self.time_off_end}")
            try:
                meas_interval_new = int(self.intervalLine.text())
            except:
                meas_interval_new = meas_interval
                pass
            lamp_warm_time = self.lampWarmSpin.value()
            if self.lampToggleCheck.isChecked():
                self.on_Shelly_off_clicked(check=False)
                print("Shelly off")
            for b in range(int(meas_interval_new)):
                time.sleep(1)
                if self.qy_cancelled == True:
                    break
                if b == int(meas_interval_new-lamp_warm_time):
                    if self.lampToggleCheck.isChecked():
                        self.on_Shelly_on_clicked()
                        print("Shelly on")
            self.on_LEDoff_clicked()




self.on_lampOffBtn_clicked()
self.on_LEDoff_clicked()
self.qyRunning.setText("Experiment not running.")
self.startQYBtn.setEnabled(True)
self.print_to_message_box("QY experiment finished succesfully.")
return
```

```
@pyqtSlot()
def on_TBC_start_clicked(self):
    self.start_TBC.emit()
    self.print_to_message_box("Thermal back-conversion experiment started")
    return

@pyqtSlot()
def set_start_TBC(self):
    self.TBC_started = True
    return

@pyqtSlot()
def set_qy_cancel(self):
    self.qy_cancelled = True
    return

@pyqtSlot()
def on_stopQYBtn_clicked(self):
    self.cancel_qy.emit()
    self.print_to_message_box("QY experiment end requested. Shutting down.")
    return

@pyqtSlot()
def on_startQYBtn_clicked(self):

    #This is where you select the log file name
    options = QFileDialog.Options()
    fileName, _ = QFileDialog.getSaveFileName(self,"QFileDialog.getSaveFileName()","","All Files...
    if fileName:
        self.print_to_message_box(f"Saving to: {fileName}")
        self.log_path = fileName
    else:
        self.print_to_message_box("No filename was selected and therefore the experiment has not...
        self.startQYBtn.setEnabled(True)
        return

    self.print_to_message_box("QY experiment started.")
    self.first = True
    meas_interval = self.intervalLine.text()
    self.get_abs_spectrum = True
    # self.ratio_t0 = time.time()
    self.t0 = time.time()

    try:
        if self.thread_qy.isRunning():
            print("Shutting down running thread.")
            self.thread_qy.terminate()
            time.sleep(1)
        else:
            print("No thread was running.")
    except:
        print("Didn't find thread.")
    self.thread_qy = QThread() # this created an additional computing thread for processes, so t...
    self.worker_qy = Worker() # this is a worker that will tell when the job is done
    self.worker_qy.func = self.runQY #here the job of the worker is defined. it should only be o...
    self.worker_qy.moveToThread(self.thread_qy) #the workers job is moved from the frontend to t...
    self.thread_qy.started.connect(self.worker_qy.run) # when the thread is started, the worker ...
    self.worker_qy.finished.connect(self.thread_qy.quit) # when the worker is finished, the the ...
    self.worker_qy.finished.connect(self.worker_qy.deleteLater)
    self.thread_qy.finished.connect(self.thread_qy.deleteLater)
    self.thread_qy.start() #here the thread is actually started
    return

##########################################################################################################...
### ALL BELOW IS VAPOURTEC
@pyqtSlot()
def on_OpenCommBtn_vap_clicked(self):
```

```
        if self.checkA.isChecked() == False and self.checkB.isChecked() == False and self.checkC.isC...
            print("Remember to pick which pumps you want to connect to")
            return
        print("This button should connect to the Vapourtec")
        self.client = rs.RSeriesClient('opc.tcp://localhost:43344')
        self.conState = self.client.connect()
        # print(conState)
        self.rseries = self.client.getRSeries()
        self.manualControl = self.rseries.getManualControl()
        # self.reactor = self.manualControl.getR4I().getReactors()['3']
        print(f"Pumps available are {self.manualControl.getR2Secondary().getPumps()}")
        # print(f"Pump selected is {self.pump_name.currentText()}")
        if self.checkA.isChecked():
            self.pumpA = self.manualControl.getR2Primary()._getPump(f"A")
        if self.checkB.isChecked():
            self.pumpB = self.manualControl.getR2Primary()._getPump(f"B")
        if self.checkC.isChecked():
            self.pumpC = self.manualControl.getR2Secondary()._getPump(f"A")
        if self.checkD.isChecked():
            self.pumpD = self.manualControl.getR2Secondary()._getPump(f"B")
        # self.pump = self.manualControl.getR2Secondary()._getPump(f"{self.pump_name.currentText()}"...
        # self.temperature = self.reactor.getTemperature()
        if self.pumpD.getValveSRState():
            self.switchSRBtn_D.setText(f"Switch SR. Current: R")
        else:
            self.switchSRBtn_D.setText("Switch SR. Current: S")
        time.sleep(1)
        # self.pump.setValveSRState(True)
        self.StartFlowBtn_D.setEnabled(True)
        self.StopFlowBtn_D.setEnabled(True)
        self.StartFlowBtn_C.setEnabled(True)
        self.StopFlowBtn_C.setEnabled(True)
        self.CloseCommBtn_vap.setEnabled(True)
        self.switchSRBtn_D.setEnabled(True)
        self.switchSRBtn_C.setEnabled(True)
        self.getFlow.setEnabled(True)
        return

    @pyqtSlot()
    def on_CloseCommBtn_vap_clicked(self):
        print("This should close the connection to Vapourtec")
        print('stopping manual control')
        self.manualControl.stopAll()
        print('turn off pump and reactor')
        self.pumpD.setFlowRate(0)
        # self.temperature.setTemperature(25) #This is turned off for testing

        if self.conState:
            self.client.disconnect()
        return

    @pyqtSlot()
    def show_help_oxy(self):
        QMessageBox.information(self, "Info", "Setup for Oxygen level experiment.\
             \nSet Integration time to 500 ms\nSet Number of averages to 5\nSet Number of measuremen...
             \nThe Pump chosen should be B, becuase pump D is apparently called B on the Secondary R...
             \nFor now you can change the flow rate manually by selecting the flow rate and clicking...
        return

class MplCanvas(FigureCanvasQTAgg):

    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)

class Worker(QObject):
```

```
    finished = pyqtSignal()
    func = None
    def run(self):
        self.func()
        self.finished.emit()
        return

def main():
    app = QApplication(sys.argv)
    app.lastWindowClosed.connect(app.quit)
    app.setApplicationName("Quantum Yield Program")
    form = MainWindow()
    form.show()
    app.exec_()

if __name__ == "__main__":
    main()
```

FILE 3/11: 55617119_readme.md

This folder contains the data analysis tools and automation control scripts for the paper: An automa...

FILE 4/11: data/55617149_readme.md

Data files are available at https://doi.org/10.6084/m9.figshare.28254350

FILE 5/11: lib/55617122_readme.md

```
lib contents
```

FILE 6/11: lib/forms/55617134_QYA.pyw

```
#!/usr/bin/env python3

#imports
import os
# import platform
import sys
import time
from pathlib import Path

import scipy.optimize
from scipy.stats import chisquare
from sklearn.metrics import r2_score

sys.path.append(os.path.join(os.path.dirname(__file__), "lib"))

from ctypes import *
from math import log10 as log
from statistics import mean
import json

import matplotlib
import numpy as np
import pandas as pd
# import rseriesopc as rs
import scipy
from matplotlib import pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.backends.backend_qt5agg import \
    NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure

matplotlib.use("Qt5Agg")
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from lib.windows import QYA_f_win as form_window
import flux_dict

# import globals
# import qy_window
from lib.avaspec import *

# from tqdm import tqdm
# print(__file__.split("\\")[-1].split(".")[0])
# head, tail = os.path.split(__file__)
# print(head, tail)

# time.sleep(2)

cwd = os.getcwd()

# if os.path.isfile(cwd+"/avaspecx64.dll"):
#     # print("You are in the right directory!")
#     os.add_dll_directory(cwd)
#     pass
# else:
#     print("You are not in the directory with avaspecx64.dll")
#     raise FileNotFoundError

# time.sleep(1)


# lib = cdll.LoadLibrary("C:\Program Files\IVI Foundation\VISA\Win64\Bin\TLUP_64.dll")
```

```
class MainWindow(QMainWindow, form_window.Ui_MainWindow):
    params_list = ["analysisEndTime",
                   "analysisStartTime",
                   "analysisNumPoints",
                   "analysiswlLineEdit",
                   "manCorr",
                   "zeroLineEdit",
                   "abswlLineEdit",
                   "fluxLineEdit",
                   "fluxwlLineEdit",
                   "concLineEdit",
                   "extincLineEdit",
                   "extincwlLineEdit",
                   "pathLength",
                   "qyLineEdit",
                   "volLineEdit",
                   "kLine",
                   "simTimeLineEdit"
                   ]
    qya_params = []
    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.setWindowTitle("Quantum Yield Analysis Program")
        self.setWindowIcon(QIcon('lib/sun.png'))
#       self.OpenCommBtn.clicked.connect(self.on_OpenCommBtn_clicked)
#       do not use explicit connect together with the on_ notation, or you will get
#       two signals instead of one!
        self.fit_fig = MplCanvas()
        toolbar = NavigationToolbar(self.fit_fig, self)
        self.fitFigLayout.addWidget(self.fit_fig)
        self.fitFigLayout.addWidget(toolbar)
        self.spec_fig = MplCanvas()
        toolbar = NavigationToolbar(self.spec_fig, self)
        self.specFigLayout.addWidget(self.spec_fig)
        self.specFigLayout.addWidget(toolbar)
        self.res_fig = MplCanvas()
        toolbar = NavigationToolbar(self.res_fig, self)
        self.resFigLayout.addWidget(self.res_fig)
        self.resFigLayout.addWidget(toolbar)
        self.flux_dict = flux_dict.all_flux

        if os.path.isfile("lib/qya_params.json"):
            with open('lib/qya_params.json', 'r') as file:
                self.qya_params = json.load(file)
            for i in range(len(self.params_list)):
                if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QLineEdit":
                    self.__dict__[f"{self.params_list[i]}"].setText(self.qya_params[i])
                if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QSpinBox":
                    self.__dict__[f"{self.params_list[i]}"].setValue(int(self.qya_params[i]))
        else:
            print("No init params were found.")

    def closeEvent(self, event):

        quit_msg = "Are you sure you want to exit the program?"
        reply = QMessageBox.question(self, 'Warning',
                        quit_msg, QMessageBox.Yes, QMessageBox.No)

        if reply == QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

    ###############################################################################################...
    # GENERAL UI
```

```
@pyqtSlot()
def on_saveParams_clicked(self):
    self.qya_params=[]
    for i in range(len(self.params_list)):
        if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QLineEdit":
            self.qya_params.append(self.__dict__[f"{self.params_list[i]}"].text())
        if self.__dict__[f"{self.params_list[i]}"].__class__.__name__ == "QSpinBox":
            self.qya_params.append(str(self.__dict__[f"{self.params_list[i]}"].value()))
    with open('lib/qya_params.json', 'w') as file:
        json.dump(self.qya_params, file)

    return


 #Print time and message

def print_to_message_box(self, text):
        t = time.localtime()
        current_time = time.strftime("%H:%M:%S", t)
        self.logTextEdit.appendPlainText(f"{current_time}  {text}")
        return

@pyqtSlot()
def on_runCalcBtn_clicked(self):
    ### Get calculation parameters from the fields and check if they are viable
    self.check_file()
    if self.good_file == False:
        self.print_to_message_box("Analysis cancelled due to file.")
        return

    self.check_params()
    params = self.get_params()
    print(params)
    ### Run QY calculation
    qys = self.run_QY_calc(params)
    if self.fluxRadio.isChecked():
        self.print_to_message_box(f"Quantum yield at {params.fluxwl} nm: {100*qys[0]:.3f} %")
        self.print_to_message_box(f"The error of the fit is: {np.sqrt(np.diag(self.pcov)).real*1...
    if self.qyRadio.isChecked():
        self.print_to_message_box(f"Photon flux at {params.fluxwl} nm: {qys[0]:.3e}")
        self.print_to_message_box(f"The error of the fit is: {np.sqrt(np.diag(self.pcov)).real*1...
    return qys

@pyqtSlot()
def on_runSimBtn_clicked(self):
    # Make the simulation prediction and show the plots.

    return

def check_file(self):
    self.good_file = True
    try:
        check = pd.read_csv(self.fileName, sep=' ', header = None,skiprows=2)

    except:
        print("File could not be read.")
        self.good_file = False
        return
    if np.min(check.iloc[1,1:]) < 0:
        qm = QMessageBox
        response = qm.question(self,'', "Some values in the spectrum are below 0. Do you want to...
        if response == qm.Yes:
            self.good_file = True
        elif response == qm.No:
            self.good_file = False
    return
```

```
def check_params(self):
    self.good_params = True

    if self.good_params == True:
        # self.print_to_message_box("Parameters are looking good. Proceeding calculation.")
        pass
    return

@pyqtSlot()
def on_loadLEDBtn_clicked(self):
    try:
        dlg = QDialog(self)
        dlg.resize(250,300)
        dlg.setFixedSize(250,300)
        dlg.setWindowTitle("Load LED data:")
        layout = QVBoxLayout()
        cb1 = QComboBox()
        cb1.addItems(["280","308","340","365","405","430","455"])
        cb1.setCurrentIndex(1)
        cb2 = QComboBox()
        cb2.addItems([f"{x}" for x in range(100,1201,100)])
        cb2.setCurrentIndex(1)
        cb3 = QComboBox()
        cb3.addItems(["Standard 10 mm", "No cuvette", "Standard MeCN", "80 ul flowcell", "80 ul ...
        layout.addWidget(QLabel("Choose wavelength:"))
        layout.addWidget(cb1)
        layout.addWidget(QLabel("Choose power:"))
        layout.addWidget(cb2)
        layout.addWidget(QLabel("Choose Cuvette:"))
        layout.addWidget(cb3)
        b3 = QPushButton("Preview photon flux")
        layout.addWidget(b3)
        preview = QLabel("Preview: N/A")
        layout.addWidget(preview)
        # preview_text = self.get_flux_from_params(cb1.currentText(),cb2.currentText(),cb3.curre...
        preview_text = f"Preview: {self.flux_dict[0]['308']['100']}"
        # preview_text = f"Preview: {self.flux_dict['um45mm3'][{str(cb1.currentText())}][str(cb2...
        b3.clicked.connect(lambda: preview.setText(str(self.flux_dict[cb3.currentIndex()][cb1.cu...
        b1 = QPushButton("Load",dlg)
        b1.clicked.connect(dlg.accept)
        b2 = QPushButton("Cancel",dlg)
        b2.clicked.connect(dlg.reject)
        btn_layout = QHBoxLayout()
        btn_layout.addWidget(b1)
        btn_layout.addWidget(b2)
        layout.addLayout(btn_layout)
        dlg.setLayout(layout)
        dlg.setWindowTitle("Load photon flux")
        dlg.setWindowModality(Qt.ApplicationModal)
        path_lengths = [1,1,1,1,1,1,1,1,1,1,1]
        dlg.accepted.connect(lambda: self.set_flux(f"{float(self.flux_dict[cb3.currentIndex()][c...
        dlg.exec_()
    except:
        self.print_to_message_box("Something went wrong with loading the photon flux.")
    return

def set_flux(self,flux,wl,v = None,pathlength = None):
    self.fluxLineEdit.setText(str(flux))
    self.fluxwlLineEdit.setText(str(wl))
    self.abswlLineEdit.setText(str(wl))
    if v == None:
        pass
    else:
        self.volLineEdit.setText(v)
    if pathlength == None:
        pass
    else:
```

```
            self.pathLength.setText(str(pathlength))
        return

    def get_params(self):
        params = pd.Series(dtype=object)
        params.qy = float(self.qyLineEdit.text())
        params.extinc = float(self.extincLineEdit.text())
        params.extincwl = float(self.extincwlLineEdit.text())
        params.conc = float(self.concLineEdit.text())
        params.flux = float(self.fluxLineEdit.text())
        params.fluxwl = float(self.fluxwlLineEdit.text())
        params.analysiswl = float(self.analysiswlLineEdit.text())
        params.abswl = float(self.abswlLineEdit.text())
        params.volume = float(self.volLineEdit.text())
        params.zeropoint = float(self.zeroLineEdit.text())
        return params

    def run_QY_calc(self,params):
        try:
            calc = QY_analysis()
            if self.qyRadio.isChecked():
                calc.qy = params.qy
            else:
                calc.qy = None
            if self.extincRadio.isChecked():
                calc.extinc = params.extinc
                calc.extincwl = params.extincwl
            if self.concRadio.isChecked():
                calc.start_conc = params.conc
            if self.fluxRadio.isChecked():
                calc.flux = params.flux
            else:
                calc.flux = None
            calc.file_name = self.fileName
            calc.plot = False
            calc.zero_wl = params.zeropoint
            calc.V = params.volume
            calc.LED_current = 300
            calc.LED_wl = params.abswl
            calc.qy_wl = params.analysiswl
            calc.num_points = self.analysisNumPoints.value()
            calc.start_point = int(self.analysisStartTime.value())
            calc.end_point = int(self.analysisEndTime.value())
            calc.path_length = float(self.pathLength.text())
            if self.extincRadio.isChecked():
                qys,t_list = calc.calculate_QY(calc_conc = True)
            else:
                qys,t_list = calc.calculate_QY()
            self.pcov = calc.pcov
            if self.qyRadio.isChecked():
                y_list = calc.nbd_conc_imag(t_list,calc.qy,qys[0],calc.c_1.real,calc.c_1.imag)
                self.Plot_new_spectrum([t_list,calc.data_wl],[y_list,calc.data_i],self.fit_fig,doubl...
                # self.Plot_new_spectrum([t_list,calc.data_wl],[calc.nbd_conc_imag(t_list,calc.qy,qy...
            elif self.fluxRadio.isChecked():
                y_list = calc.nbd_conc_imag(t_list,qys[0],calc.flux,calc.c_1.real,calc.c_1.imag)
                self.Plot_new_spectrum([t_list,calc.data_wl],[y_list,calc.data_i],self.fit_fig,doubl...
                # self.Plot_new_spectrum([t_list,calc.data_wl],[calc.nbd_conc_imag(t_list,qys[0],cal...
            # self.print_to_message_box([calc.data_wl.to_list(),calc.data_i])
            for i in range(len(calc.first_abs)):
                if i == 0:
                    self.Plot_new_spectrum(calc.first_wl,calc.first_abs[i],self.spec_fig,start_x=150...
                else:
                    self.Plot_new_spectrum(calc.first_wl,calc.first_abs[i],self.spec_fig,start_x=150...
            self.x_list = calc.data_wl.to_list()
            self.y_list = calc.data_i

            if self.prodAbsCheck.isChecked():
```

```
qy_prod_guess = 0.5
eps_prod = float(self.epsProdLine.text())
up_scale = 1.02
down_scale = 0.98
if self.qyRadio.isChecked():
    y,x,z = self.prod_abs_correction(calc.qy,calc.start_conc,calc.b,calc.V,calc.N_A,...
    init_qy = calc.qy
if self.fluxRadio.isChecked():
    y,x,z = self.prod_abs_correction(qys[0],calc.start_conc,calc.b,calc.V,calc.N_A,c...
    init_qy = qys[0]
# self.Plot_new_spectrum(self.x_list,z,self.fit_fig,clear=False,label1="Fit with pro...
# print(calc.data_i)
loose_conv = False
tight_conv = False
opt_threshold = 1e-3
qy_grad_up = 1
qy_grad_prod_up = 1
qy_grad_down = 1
qy_grad_prod_down = 1
current_qy = init_qy
current_qy_prod = qy_prod_guess
for attempt in range(100):
    if len(z) == len(calc.data_i):
        corr_matrix = np.corrcoef(calc.data_i,z)
        corr = corr_matrix[0,1]
        loose_r2 = corr**2
        tight_r2 = r2_score(calc.data_i,z)
        if self.qyRadio.isChecked():
            y_up,x_up,z_up = self.prod_abs_correction(calc.qy*up_scale,calc.start_co...
            y_down,x_down,z_down = self.prod_abs_correction(calc.qy*down_scale,calc....
            y_up,x_up,z_up_prod = self.prod_abs_correction(calc.qy,calc.start_conc,c...
            y_down,x_down,z_down_prod = self.prod_abs_correction(calc.qy,calc.start_...
        if self.fluxRadio.isChecked():
            y_up,x_up,z_up = self.prod_abs_correction(current_qy*up_scale,calc.start...
            y_down,x_down,z_down = self.prod_abs_correction(current_qy*down_scale,ca...
            y_up,x_up,z_up_prod = self.prod_abs_correction(current_qy,calc.start_con...
            y_down,x_down,z_down_prod = self.prod_abs_correction(current_qy,calc.sta...

        if loose_r2 > 0.99500:
            loose_conv = True
            loose_qy = current_qy
            loose_qy_prod = current_qy_prod
        if tight_r2 > 0.99500:
            tight_conv = True
            tight_qy = current_qy
            tight_qy_prod = current_qy_prod
        if tight_r2 > 0.9990:
            tight_qy = current_qy
            tight_qy_prod = current_qy_prod
            break
        else:
            tight_conv = False
        if abs(min([qy_grad_up,qy_grad_down])) < opt_threshold:
            qy_opt_complete = True
            opt_qy = current_qy
        else:
            qy_opt_complete = False
        if abs(min([qy_grad_prod_up,qy_grad_prod_down])) < opt_threshold:

            qy_prod_opt_complete = True
            # opt_k = current_k
        else:
            qy_prod_opt_complete = False
        if tight_conv == False or qy_prod_opt_complete == False or qy_opt_complete =...
            ###GET GRADIENTS FOR EACH QY
            # qy_grad =      ((r2_score(calc.data_i,z_up))-     (r2_score(calc.data_...
            # qy_grad_prod = ((r2_score(calc.data_i,z_up_prod))-(r2_score(calc.data_...
```

```
                qy_grad_up =        (np.sum(np.square(np.subtract(calc.data_i,z_up)))   ...
                qy_grad_down =      -(np.sum(np.square(np.subtract(calc.data_i,z_down))))...
                qy_grad_prod_up =   (np.sum(np.square(np.subtract(calc.data_i,z_up_prod)...
                qy_grad_prod_down = -(np.sum(np.square(np.subtract(calc.data_i,z_down_pr...

                print(f"{attempt} grad1: {min([qy_grad_up,qy_grad_down]):.5f}   !  qy1: ...
                if attempt > 100:
                    factor = attempt/100
                else:
                    factor = 1
                if qy_grad_up <= qy_grad_down:
                    current_qy = current_qy*up_scale
                else:
                    current_qy = current_qy*down_scale
                if qy_grad_prod_up <= qy_grad_prod_down:
                    current_qy_prod = current_qy_prod*up_scale
                else:
                    current_qy_prod = current_qy_prod*down_scale


                # new_qy = current_qy * (1+qy_grad/factor)
                # if new_qy > current_qy * up_scale:
                #     current_qy = current_qy * up_scale
                # elif new_qy < current_qy * down_scale:
                #     current_qy = current_qy * down_scale
                # else:
                #     current_qy = new_qy

                # new_qy_prod = current_qy_prod * (1+qy_grad_prod/factor)
                # if new_qy_prod > current_qy_prod * up_scale:
                #     current_qy_prod = current_qy_prod * up_scale
                # elif new_qy_prod < current_qy_prod * down_scale:
                #     current_qy_prod = current_qy_prod * down_scale
                # else:
                #     current_qy_prod = new_qy_prod
                # current_qy = current_qy * (1+qy_grad/factor)
                # current_qy_prod = current_qy_prod * (1+qy_grad_prod/factor)

                y,x,z = self.prod_abs_correction(current_qy,calc.start_conc,calc.b,calc....
            else:
                print("Converged.")
                break
        else:
            print(len(z),len(calc.data_i))
            print("failed")
    self.Plot_new_spectrum(self.x_list,z,self.fit_fig,clear=False,label1="Fit with prod ...
    self.Plot_new_spectrum(self.x_list,np.array(calc.data_i)-np.array(z),self.res_fig,cl...
    if loose_conv:
        self.print_to_message_box(f"Loose convergence was obtained with quantum yields: ...
    if tight_conv:
        self.print_to_message_box(f"Tight convergence was obtained with quantum yields: ...
    else:
        self.print_to_message_box(f"Optimisation did not converge. Last result was: {100...
        self.print_to_message_box(f"b was {calc.b}")
else:
    pass

if self.kCheck.isChecked() and not self.prodAbsCheck.isChecked():  ## Function for fitti...
    k = float(self.kLine.text())
    up_scale = 1.02
    down_scale = 0.98
    if self.qyRadio.isChecked():
        y,x,z = self.after_correction(calc.qy,k,calc.start_conc,calc.b,calc.V,calc.N_A,q...
    elif self.fluxRadio.isChecked():
        y,x,z = self.after_correction(qys[0],k,calc.start_conc,calc.b,calc.V,calc.N_A,ca...
    qy_grad = 1
    k_grad = 1
```

```
opt_threshold = 2e-2
loose_conv = False
tight_conv = False
qy_opt_complete = False
k_opt_complete = False
init_qy = qys[0]
current_qy = init_qy
current_k = k
for attempt in range(100):
    if len(z)==len(calc.data_i):
        corr_matrix = np.corrcoef(calc.data_i, z)
        corr = corr_matrix[0,1]
        loose_r2 = corr**2
        tight_r2 = r2_score(calc.data_i,z)
        if self.qyRadio.isChecked():
            y_up,x_up,z_up = self.after_correction(calc.qy*up_scale,k,calc.start_con...
            y_down,x_down,z_down = self.after_correction(calc.qy*down_scale,k,calc.s...

        if self.fluxRadio.isChecked():
            y_up,x_up,z_up = self.after_correction(current_qy*up_scale,current_k,cal...
            y_down,x_down,z_down = self.after_correction(current_qy*down_scale,curre...
            y_up,x_up,k_up = self.after_correction(current_qy,current_k*up_scale,cal...
            y_down,x_down,k_down = self.after_correction(current_qy,current_k*down_s...
        # print(f"tight r2 value is: {tight_r2:.5f}")

        if loose_r2 > 0.995000:
            loose_conv = True
            loose_qy = current_qy
            loose_k = current_k
        if tight_r2 > 0.995000:
            tight_conv = True
            tight_qy = current_qy
            tight_k = current_k
        if tight_r2 > 0.9999:
            tight_qy = current_qy
            break
        else:
            tight_conc = False
        if abs(qy_grad) < opt_threshold:
            qy_opt_complete = True
            opt_qy = current_qy
        else:
            qy_opt_complete = False
        if abs(k_grad) < opt_threshold:
            k_opt_complete = True
            opt_k = current_k
        else:
            k_opt_complete = False
        if qy_opt_complete == False or k_opt_complete == False or tight_conv == Fals...
            qy_grad =      -(np.sum(np.square(np.subtract(calc.data_i,z_up)))      -...
            if k != 0:
                k_grad =       -(np.sum(np.square(np.subtract(calc.data_i,k_up)))   ...
            else:
                k_grad = 0
            print(f"{attempt}: qy_grad: {qy_grad:.6f} !   qy: {current_qy:.5f} !    ...
            factor = 1
            if attempt > 50:
                factor = attempt/50
            new_qy = current_qy * (1+qy_grad/factor)
            if new_qy > current_qy * up_scale:
                current_qy = current_qy * up_scale
            elif new_qy < current_qy * down_scale:
                current_qy = current_qy * down_scale
            else:
                current_qy = new_qy
            new_k = current_k * (1+k_grad/factor)
            if new_k > current_k * up_scale:
```

```
                            current_k = current_k * up_scale
                        elif new_k < current_k * down_scale:
                            current_k = current_k * down_scale
                        else:
                            current_k = new_k
                        # current_k = k ## This causes the optimisation to only use the fixed va...
                        y,x,z = self.after_correction(current_qy,current_k,calc.start_conc,calc....
                    else:
                        print(f"opt finished at step{attempt}")
                        break

                    """
                    if loose_r2 > 0.99000:
                        loose_conv = True
                        loose_qy = current_qy
                    if tight_r2 > 0.99000:
                        tight_conv = True
                        tight_qy = current_qy
                        break
                    if tight_conv == False:
                        #Correct the qy for a better fit
                        factor = 2
                        corr_value = ((mean(z)/mean(calc.data_i)))
                        if corr_value < 1:
                            corr_corr = 1-factor*(1-corr_value)/(attempt+1)
                        elif corr_value >= 1:
                            corr_corr = 1+factor*(corr_value-1)/(attempt+1)
                        print(f"corr_value is {corr_value}")
                        current_qy = current_qy*corr_value*corr_corr
                        y,x,z = self.after_correction(current_qy,k,calc.start_conc,calc.b,calc.V...
                    """
                else:
                    print("Error...")
                    print(len(z))
                    print(len(calc.data_i))
                    break

            self.print_to_message_box(f"Optimized qy is: {current_qy*100:.4f} %")
            self.print_to_message_box(f"Optimized k value is: {current_k:.5f} 1/s")
            # if tight_qy:
            #     self.print_to_message_box(f"Tight QY is {tight_qy*100:.3f} %")
            if self.fluxRadio.isChecked():
                self.Plot_new_spectrum(x,np.array(y)*calc.b,self.fit_fig,clear=False,label1="Cor...
        else:
            pass
        if not self.kCheck.isChecked() and not self.prodAbsCheck.isChecked():
            #DO SIMPLE FUNCTION OPTIMISATION...

            pass
        # self.Plot_new_spectrum(calc.data_wl,np.array(calc.data_i)-calc.nbd_conc_imag(calc.data...
        # print([x.real for x in np.array(calc.data_i)-calc.nbd_conc_imag(calc.data_wl,qys[0],ca...
        # p,cov = scipy.optimize.curve_fit(lambda: calc.nbd_conc_imag(x,calc.qy,qys[0],calc.c_1....
    except:
        self.print_to_message_box("Calculation failed. Check your parameters.")
        return [1,1]
    return qys

def led_intgrl_correction(self):

    return

def prod_abs_correction(self,qy,start_conc,b,V,N_A,I,t_list,b_prod=0,qy_prod=0,k=0,b_ex = 1):
    # b = float(self.epsReacLine.text())
    b_analysis_reac = float(self.epsReacAnal.text())
    b_analysis_prod = float(self.epsProdAnal.text())

    last_time = t_list.to_list()[-1]
```

```
    time_step = 1
    time_range = np.linspace(0,last_time,int(last_time/time_step))
    c = start_conc
    c_list = [c]
    c_list_final = []
    c_list_final_prod  = []
    c_prod_list = [0]
    if self.kCheck.isChecked():
        k = float(self.kLine.text())
    else:
        k = 0

    for y in range(len(time_range)):
        b_trans = b_ex
        c_trans = c_list[-1]
        b_cis = b_prod
        c_cis = c-c_list[-1]
        q_0 = I
        # q_trans = (c_trans * b_trans)/(c_trans * b_trans + c_cis * b_cis) * q_0 * (1-10**(-c_t...
        # q_cis = (c_cis * b_cis)/(c_trans * b_trans + c_cis * b_cis) * q_0 * (1-10**(-c_trans *...

        c_grad = (((1/V)*(q_0/N_A)*((1-10**(-(c_trans*b_trans+c_cis*b_cis)))/(c_trans*b_trans+c_...
        # c_grad = (-qy*I*(1-10**(-b*c_list[-1]))/(V*N_A) + (qy_prod*I*(1-10**(-b*(c-c_list[-1])...
        c_list.append(c_list[-1]+c_grad)
        c_prod_list.append(c_prod_list[-1]-c_grad)

        # if y%10 == 0:
        #     c_list_final.append(c_list[-1])
    # print(c_trans,b_trans,c_cis,b_cis)
    if abs(c_list[-1]+c_prod_list[-1]-c) > 1e-7:
        print(f"Difference in conc is {c-(c_list[-1]+c_prod_list[-1]):.2e}")
    c_list.pop(-1)
    c_prod_list.pop(-1)
    for i in t_list:
        c_list_final.append(c_list[np.argmin([abs(o-i) for o in time_range])])
        c_list_final_prod.append(c_prod_list[np.argmin([abs(o-i) for o in time_range])])



    return c_list,time_range,np.add(np.array(c_list_final)*b,np.array(c_list_final_prod)*b_analy...

def after_correction(self,qy,k,start_conc,b,V,N_A,I,t_list,b_ex = 1):
    last_time = t_list.to_list()[-1]
    time_step = 0.2
    time_range = np.linspace(0,last_time,int(last_time/time_step))
    # time_seconds = int(self.simTimeLineEdit.text())

    # time_steps = int(time_seconds/time_step)
    # time_range = [x*time_step for x in range(time_steps)]
    c = start_conc
    c_list = [c]
    c_list_final = []
    for y in range(len(time_range)):
        c_grad = (-qy*I*(1-10**(-b_ex*c_list[-1]))/(V*N_A) + k*(c-c_list[-1]))*time_step
        c_list.append(c_list[-1]+c_grad)
        # if y%10 == 0:
        #     c_list_final.append(c_list[-1])
    c_list.pop(-1)
    for i in t_list:
        c_list_final.append(c_list[np.argmin([abs(o-i) for o in time_range])])

    return c_list,time_range,np.array(c_list_final)*b

def Plot_new_spectrum_old(self, x,y,func, draw = True,start_x = 0,end_x = -1):
    func.axes.clear()
    if len(x)>1:
        func.axes.plot(x[0], y[0],label = "Fit")
```

```
        func.axes.plot(x[1], y[1],label = "Data")
    else:
        func.axes.plot(x, y,label = "Fit")

    func.axes.set_xlabel("Irradiation time (s)")
    func.axes.set_ylabel(f"Absorption  @ {self.analysiswlLineEdit.text()} nm")
    func.axes.legend()
    # self.fit_fig.axes.tight_layout()
    if draw == True:
        func.draw()
    else:
        pass

def Plot_new_spectrum(self, x,y,func, draw = True,start_x = 0,end_x = -1,dots = False,double=Fal...
    if clear:
        func.axes.clear()
    else:
        pass
    if typ == "spec":
        func.axes.plot(x[start_x:end_x], y[start_x:end_x],label = label1,color="black",linewidth...
        func.axes.set_xlabel("Wavelength (nm)")
        func.axes.set_ylabel(f"Absorbance")
        func.draw()
        return
    if double:
        print("All instances were lists")

        if dots:
            func.axes.plot(x[1], y[1],label = label2)
            func.axes.scatter(x[0], y[0], label = label1)
        else:
            # func.axes.plot(x[0][start_x:end_x], y[0][start_x:end_x],label = label1)
            func.axes.plot(x[0], y[0],label = label1,color="red")
            func.axes.scatter(x[1], y[1],label = label2,color="black",marker="x")

    else:
        # func.axes.plot(x[start_x:end_x], y[start_x:end_x],label = "Data")
        if dots == True:
            func.axes.plot(x, y,label = label1)
            func.axes.scatter(x, y)
        else:
            func.axes.plot(x[start_x:end_x], y[start_x:end_x],label = label1)

    func.axes.set_xlabel("Irradiation time (s)")
    func.axes.set_ylabel(f"Absorption  @ {self.analysiswlLineEdit.text()} nm")
    func.axes.legend()

    # self.fit_fig.axes.tight_layout()
    if draw == True:
        func.draw()
    else:
        pass

@pyqtSlot()
def on_getFileBtn_clicked(self):
    self.fileName = self.openFileNameDialog("Choose file containing UV-VIS data from the QY expe...
    self.chosenLabel.setText("File chosen: "+ self.fileName.split("/")[-1])
    self.chosenLabel.setToolTip("File chosen: "+ self.fileName)
    self.chosenLabel.setToolTipDuration(30000)
    return

def openFileNameDialog(self,windowTitle):


    options = QFileDialog.Options()
    # options |= QFileDialog.DontUseNativeDialog
    fileName, _ = QFileDialog.getOpenFileName(self,windowTitle, "","All Files (*);;Python Files ...
```

```
        if fileName:
            print(fileName)
        return fileName

    @pyqtSlot()
    def on_runSimBtn_clicked(self):
        ### RUN LINEAR PART FIRST?
        # conc = start_conc - (qy*I*t)/(V*N_A)
        ### THIS IS NON LINEAR PART
        sim = QY_analysis()
        self.sim_time = int(self.simTimeLineEdit.text())
        t_list = np.arange(0,self.sim_time*2)*0.5
        self.b = float(self.extincLineEdit.text())
        sim.b = self.b
        sim.b_ex = self.b
        sim.num_points = self.analysisNumPoints.value()
        s = float(self.concLineEdit.text())
        n=1
        print(self.b*s)
        self.c_1 = (np.log(1 - (np.power(10,(self.b * s)) + 0j )) + 2j * np.pi * n)/(self.b * np.log...
        flux = float(self.fluxLineEdit.text())
        qy = float(self.qyLineEdit.text())
        plt.plot(t_list,sim.nbd_conc_imag(t_list,qy,flux,self.c_1.real,self.c_1.imag)/self.b)
        plt.ylim(bottom=0)
        plt.show()
        return

    @pyqtSlot()
    def on_btnExport_clicked(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self,"QFileDialog.getSaveFileName()","","All Files...
        if fileName:
            self.print_to_message_box(f"Saving to: {fileName}")
            self.save_path = fileName
        else:
            self.print_to_message_box("No filename was selected and therefore the experiment has not...
            return
        try:
            df = pd.DataFrame()
            df["time"] = self.x_list
            df["abs"] = self.y_list
            df.to_csv(f"{fileName}.csv",index=False)
        except:
            self.print_to_message_box("No data could be exported. Perform analysis first.")
        return

class QY_analysis:
    def __init__(self):
        print("Please setup all reaction parameters for the calculation. The parameters can be found...
        #Below is the standard data for the KMP1
        self.file_name = ""
        self.LED_wl = 365 #Wavelength of the LED for irradiation
        self.qy_wl = 340 #Wavelength for the absorption peak of NBD for analysis
        self.start_conc = 9.5426e-06 #Starting concenctration of experiments performed on 21Mar2023
        self.qy = None #0.61 is the QY calculated for KMP1
        self.flux = None #Standard flux set to None, for calibration of Flux
        self.V = 80e-6 #The volume of the flow cell
        self.LED_current = 300 #The current of the LED
        self.N_A = 6.022e+23 #Avogadros number, it's required so don't question it.
        self.corr = 0
        self.zero_wl = 400
        self.num_points = 15
        self.start_point = 0
        self.end_point = -1
        self.path_length = 1
        return
```

```
    def calculate_QY(self,calc_conc = False, start_x = 0, end_x = -1):
        #Some constants

        #First check if the necessary parameters are set
        self.get_current_params()
#         cont = input("These are the current parameters. Are you sure you want to continue? y/n: ")
        cont = "y"
        if cont.lower() != "y":
            print("Cancelling calculation.")
            return
        data = pd.read_csv(self.file_name, sep=' ', header = None,skiprows=2)
        self.zero_col = np.argmin([abs(x-self.zero_wl) for x in data.iloc[0,:]])
        print(self.zero_col)
        if self.corr != 0:
            corr = self.corr
        else:
            corr = -np.mean(data.iloc[1,self.zero_col-5:self.zero_col+5])
        # print(np.min(data.iloc[1,1:]))
        ex_col = data.iloc[0,:].round(0).to_list().index(self.LED_wl)
        wl_col = data.iloc[0,:].round(0).to_list().index(self.qy_wl)
        data_wl = data.iloc[1:,0][self.start_point:self.end_point]
        data_wl = data_wl - data.iloc[self.start_point+1,0] #correction so it always starts at 0
        self.data_wl = data_wl
        data_i = [num + corr for num in data.iloc[1:,wl_col]][self.start_point:self.end_point]
        data_i = (data.iloc[1:,wl_col]-data.iloc[1:,self.zero_col]).to_list()[self.start_point:self....
        self.data_i = data_i
        data_ex = [num + corr for num in data.iloc[1:,ex_col]][self.start_point:self.end_point]
        data_ex = (data.iloc[1:,ex_col]-data.iloc[1:,self.zero_col]).to_list()[self.start_point:self...
        data_t0 = data.iloc[1,1:][self.start_point:self.end_point]
        if calc_conc:
            extinc_col = data.iloc[0,:].round(0).to_list().index(self.extincwl)
            self.start_conc = (data.iloc[1,extinc_col] + corr)/(self.extinc*self.path_length)
            print(f"Calculated starting concentration is: {self.start_conc}")
        else:
            pass
        self.first_abs = []
        self.first_wl = data.iloc[0,2:].to_list()
        for spec_point in np.linspace(self.start_point,self.end_point,10,dtype=int):
            print(spec_point)
            self.first_abs.append(np.array(data.iloc[spec_point,2:].to_list())-data.iloc[:,self.zero...
        self.b = data_i[0] /(self.start_conc*self.path_length)
        self.b_ex = data_ex[0]/(self.start_conc*self.path_length)
        remove_qc = []
        data_c = []
        for dot in range(len(data_i)):
            diff = (1-(data_i[dot]/data_i[0]))*data_i[-1]
            remove_qc.append(diff)
            data_c.append(data_i[dot]-diff)
        data_c = data_i
        self.data_ex = data_ex
        num_points = self.num_points
        if self.flux == None:
            # for i in range(15):
                # params = self.run_qy_calc(data_wl[:i],data_c[:i],self.start_conc)
            params = self.run_qy_calc(data_wl[:num_points],data_c[:num_points],self.start_conc)
        if self.qy == None:
            # for i in range(15):
            #     params = self.run_flux_calc(data_wl[:i],data_c[:i],self.start_conc)
            params = self.run_flux_calc(data_wl[:num_points],data_c[:num_points],self.start_conc)
        x=np.linspace(0,data_wl[self.end_point],400)
        return params , x

    def help(self):
        print(f"The required parameters are listed below. To see the current parameters run .get_cur...
        return

    def get_current_params(self):
```

```
        print(f"Current parameters are:\nData file: {self.file_name}\nIrradiation wavelength: {self....
              f"Measurement wavelength: {self.qy_wl} nm\nStarting concenctration: {self.start_conc} ...
              f"Calculated Quantum Yield: {self.qy}\nCalculated Photon Flux: {self.flux} 1/s\n"+
              f"Flow cell volume: {self.V} l\nLED current: {self.LED_current}")
        return

    def nbd_conc_imag(self,t,qy,I,cr,ci):
        a = (qy*I)/(self.V*self.N_A)
        return (0.434294*np.log(1-(2.71828**(2.30259*self.b_ex*complex(cr,ci)-2.30259*a*self.b_ex*t)...

    def helper_qy(self,x,qy):
        return self.nbd_conc_imag(x,qy,self.flux,self.c_1.real,self.c_1.imag)

    def helper_flux(self,x,I):
        return self.nbd_conc_imag(x,self.qy,I,self.c_1.real,self.c_1.imag)

    def run_qy_calc(self,x_vals,y_vals,start_conc):
        old_x_vals = x_vals
        old_y_vals = y_vals
#         x_vals = old_x_vals[:10]
#         y_vals = old_y_vals[:10]
        if len(x_vals)>1:
            self.b = y_vals[0] /(start_conc*self.path_length)
            s = self.start_conc
            n = 1
            self.c_1 = (np.log(1 - (10**(self.b_ex * s) + 0j )) + 2j * np.pi * n)/(self.b_ex * np.lo...
    #         popt,pcov = scipy.optimize.curve_fit(nbd_conc_imag,x_vals,y_vals,bounds=([0,1e12,1.1*c...
            try:
                popt,pcov = scipy.optimize.curve_fit(self.helper_flux,x_vals,y_vals,p0=[10e14],bound...
                self.pcov = pcov
            except:
                print("Curve fit did not converge. Check your parameters.")

            print(f"PHOTON FLUX PREDICTED TO BE: {popt[0]/self.LED_current:.2e} 1/s @ BASED ON {len(...
            return popt
        return "Not enough datapoints..."

    def run_flux_calc(self,x_vals,y_vals,start_conc):
        old_x_vals = x_vals
        old_y_vals = y_vals
#         x_vals = old_x_vals[:10]
#         y_vals = old_y_vals[:10]
        if len(x_vals)>1:
            self.b = y_vals[0] /start_conc
            s = self.start_conc
            n = 1
            self.c_1 = (np.log(1 - (10**(self.b_ex * s) + 0j )) + 2j * np.pi * n)/(self.b_ex * np.lo...
    #         popt,pcov = scipy.optimize.curve_fit(nbd_conc_imag,x_vals,y_vals,bounds=([0,1e12,1.1*c...
            try:
                popt,pcov = scipy.optimize.curve_fit(self.helper_qy,x_vals,y_vals,p0=[0.5],bounds=([...
                self.pcov = pcov
            except:
                print("Curve fit did not converge. Check your parameters.")

            print(f"QUANTUM YIELD PREDICTED TO BE: {popt[0]:.2e} % @ BASED ON {len(x_vals)} DATA POI...
            return popt
        return "Not enough datapoints..."

    def run_qy_sim(self):
        ### Get data ready for the plot
        return



class Worker(QObject):
    finished = pyqtSignal()
    func = None
```

```
    def run(self):
        self.func()
        self.finished.emit()
        return

class MplCanvas(FigureCanvasQTAgg):
    def __init__(self, parent=None, width=12, height=12, dpi=100):
        fig = plt.figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)


def main():
    app = QApplication(sys.argv)
    app.lastWindowClosed.connect(app.quit)
    app.setApplicationName("QY Calculation Program")
    form = MainWindow()
    form.show()
    app.exec_()

if __name__ == "__main__":
    main()
```

FILE 7/11: lib/forms/55617140_QYEX.pyw

```
■#!/usr/bin/env python3

#imports
import os
import sys
import time
import pandas as pd
import numpy as np
import rseriesopc as rs
from matplotlib import pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from ctypes import *
from math import log10 as log
import serial
import serial.tools.list_ports
import json
import requests

from lib.windows import QYEX_f2_win as form_window

import globals
from lib.avaspec import *

cwd = os.getcwd()

if os.path.isfile(cwd+"/avaspecx64.dll"):
    # print("You are in the right directory!")
    os.add_dll_directory(cwd)
    pass
else:
    print("You are not in the directory with avaspecx64.dll")
    raise FileNotFoundError

# time.sleep(1)


lib = cdll.LoadLibrary("C:\Program Files\IVI Foundation\VISA\Win64\Bin\TLUP_64.dll")

class MainWindow(QMainWindow, form_window.Ui_MainWindow):
    newdata = pyqtSignal()
    cancel = pyqtSignal()
    start_TBC = pyqtSignal()
    cancel_qy = pyqtSignal()
    stop_disp = pyqtSignal()
    abs_spectrum = pyqtSignal()
    stop_update = pyqtSignal()
    cancelled = False
    first = True
    use_light = False
    stop_dispersion = False
    TBC_started = False
    Spectrum_figure = plt.figure(dpi = 100)


    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.setWindowTitle("Quantum Yield EXperiment Program")
        self.setWindowIcon(QIcon('lib/sun.png'))
```

```
self.showMaximized()
self.IntTimeEdt.setText("{0:.3f}".format(2))
self.NumAvgEdt.setText("{0:d}".format(4000))
self.NumMeasEdt.setText("{0:d}".format(0))
self.StartMeasBtn.setEnabled(False)
self.t0 = time.time()
OnlyInt = QIntValidator()
OnlyInt.setRange(1, 1000)
self.NumAvgEdt.setValidator(OnlyInt)
self.NumMeasEdt.setValidator(OnlyInt)
self.newdata.connect(self.handle_newdata)
self.cancel.connect(self.cancel_meas)
self.cancel_qy.connect(self.set_qy_cancel)
self.start_TBC.connect(self.set_start_TBC)
self.stop_update.connect(self.stop_update_func)
self.get_abs_spectrum = False
self.actionHelpOxy.triggered.connect(self.show_help_oxy)
self.log_name = "default_log.txt"
self.log_path = "logs/default_log_path.txt"
self.time_off_end = 0
self.time_off_total = 0
self.qy_cancelled=False
self.LEDlist.activated.connect(self.update_power)

self.Spectrum_figure = MplCanvas()
self.monitorLayout.addWidget(self.Spectrum_figure)
plt.xlabel("Wavelength [nm]")
plt.ylabel("Intensity")
plt.tight_layout()
plt.close()

self.time_figure = MplCanvas()
self.timeLayout.addWidget(self.time_figure)
plt.xlabel("Wavelength [nm]")
plt.ylabel("Intensity")
plt.tight_layout()
plt.close()

self.abs_figure = MplCanvas()
self.absLayout.addWidget(self.abs_figure)
plt.xlabel("Wavelength [nm]")
plt.ylabel("Intensity")
plt.tight_layout()
plt.close()

self.on_Set_wavelength_btn_clicked()

#connect function for continuous temperature update
self.tempCheck.toggled.connect(self.tempCheck_func)

#load preset samples in case of crash
if os.path.isfile(cwd+"/Valve_Presets.json"):
    with open('Valve_Presets.json', 'r') as file:
        self.Valve_Presets = json.load(file)
    for i in range(1,17):
        self.__dict__['Sample_name_' + str(i)].setText(self.Valve_Presets[i-1])


# read com ports and connect to SF10 pump
self.Pump = False
self.qpod = False

self.on_setPowerBtn_clicked()

# sf10 pump auto connect
self.on_pumpConnect_clicked()
```

```
    #knauer sampler valve auto connect
    self.on_samplerConnect_clicked()

    #qpod1 auto connect
    self.on_qpodConnect_clicked()

    #shelly auto connect
    self.Shelly_IP.setText('192.168.137.85')
    self.on_shellyConnect_clicked()




##########################################################################################################...

# GENERAL UI

def print_to_message_box(self, text):#Print time and message
        t = time.localtime()
        current_time = time.strftime("%H:%M:%S", t)
        self.logTextEdit.append(f"{current_time}  {text}")
        return

def closeEvent(self, event):

    quit_msg = "Are you sure you want to exit the program?"
    reply = QMessageBox.question(self, 'Warning',
                     quit_msg, QMessageBox.Yes, QMessageBox.No)

    if reply == QMessageBox.Yes:
        event.accept()
    else:
        event.ignore()

##########################################################################################################...

#### PART CONTAINING TEMPERATURE CONTROL

@pyqtSlot()
def on_qpodConnect_clicked(self):
    for pinfo in serial.tools.list_ports.comports():
        if pinfo.serial_number == 'D30ALF5KA' and pinfo.vid == 1027: #QPOD 1 is D30ALF5KA. QPOD ...
            self.qpod = serial.Serial(pinfo.device,19200,timeout=3)
            cmd = str("[F1 ID ?]")
            self.qpod.write(cmd.encode())
            print("message sent")
            response = self.qpod.read_until(expected=b"]",size=20).decode("utf-8")
            print(response)
            time.sleep(0.1)
            self.qpod.write(b"[F1 CT -]")
            time.sleep(0.1)
            self.qpod.write(b"[F1 HT -]")
            self.tempOff.setEnabled(True)
            self.tempOn.setEnabled(True)
            self.qpodDisconnect.setEnabled(True)
            self.qpodConnect.setEnabled(False)
    return

@pyqtSlot()
def on_qpodDisconnect_clicked(self):
    try:
        self.qpod.close()
    except:
        self.print_to_message_box("qpod could not disconnect. perhaps it was not connected prope...
    self.qpodDisconnect.setEnabled(False)
    self.tempOff.setEnabled(False)
```

```
        self.tempOn.setEnabled(False)
        self.qpodConnect.setEnabled(True)
        return

    @pyqtSlot()
    def on_tempOn_clicked(self):
        if self.qpod:
            cmd = "[F1 TC +]"
            self.qpod.write(cmd.encode())
            cmd = "[F1 TC ?]"
            self.qpod.write(cmd.encode())
            response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
            if response == "[F1 TC +]":
                self.print_to_message_box("Temperature control turned ON.")
            else:
                self.print_to_message_box("Temperature control could not be turned on. Try again.")
        else:
            self.print_to_message_box("Could not send message. Qpod not connected.")
        return

    @pyqtSlot()
    def on_tempOff_clicked(self):
        if self.qpod:
            cmd = "[F1 TC -]"
            self.qpod.write(cmd.encode())
            cmd = "[F1 TC ?]"
            self.qpod.write(cmd.encode())
            response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
            if response == "[F1 TC -]":
                self.print_to_message_box("Temperature control turned OFF.")
            else:
                self.print_to_message_box("Temperature control could not be turned off. Try again.")
        else:
            self.print_to_message_box("Could not send message. Qpod not connected.")
        return

    @pyqtSlot()
    def on_tempSet_clicked(self):
        dlg = QDialog(self)
        dlg.resize(200,100)
        dlg.setFixedSize(200,100)
        dlg.setWindowTitle("Set target temperature: ")
        layout = QVBoxLayout()
        layout.addWidget(QLabel("Write target temperature: "))
        temp_spin = QSpinBox()
        temp_spin.setMinimum(0)
        temp_spin.setMaximum(100)
        temp_spin.setValue(25)
        temp_spin.setFocus()
        layout.addWidget(temp_spin)
        b1 = QPushButton("Set",dlg)
        b1.clicked.connect(dlg.accept)
        b2 = QPushButton("Cancel",dlg)
        b2.clicked.connect(dlg.reject)
        btn_layout = QHBoxLayout()
        btn_layout.addWidget(b1)
        btn_layout.addWidget(b2)
        layout.addLayout(btn_layout)
        dlg.setLayout(layout)
        dlg.setWindowModality(Qt.ApplicationModal)
        dlg.accepted.connect(lambda: self.set_temp(temp_spin.value()))
        dlg.open()
        temp_spin.setFocus()
        temp_spin.selectAll()

        return
```

```
def set_temp(self,TT):
    if self.qpod:
        cmd = f"[F1 TT S {TT}]"
        self.qpod.write(cmd.encode())
        cmd = "[F1 TT ?]"
        self.qpod.write(cmd.encode())
        response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
        print(response)
        expect = f"[F1 TT {TT:.2f}]"
        if response.strip() == expect:
            self.print_to_message_box(f"Target temperature set to {TT}.")
            self.temp_label.setText(f"T (C°): {TT}")
        else:
            self.print_to_message_box("Target temperature could not be set. Try again.")
    else:
        self.print_to_message_box("Could not send message. Qpod not connected.")

    return

def temp_update(self):
    while self.update_temp == True:
            cmd = "[F1 CT ?]"
            self.qpod.write(cmd.encode())
            response = self.qpod.read_until(expected=b"]",size=None).decode("utf-8")
            print(response)
            temp = response.split(" ")[-1].replace("]","")
            self.tempCheck.setText(f"Current temp: {temp}")
            time.sleep(5)
    return

def tempCheck_func(self): #this is connected to the toggle of the temperature update checkbox
    if self.qpod:
        if self.tempCheck.isChecked():
            self.update_temp = True
            self.thread_temp = QThread() # this created an additional computing thread for proce...
            self.worker_temp = Worker() # this is a worker that will tell when the job is done
            self.worker_temp.func = self.temp_update #here the job of the worker is defined. it ...
            self.worker_temp.moveToThread(self.thread_temp) #the workers job is moved from the f...
            self.thread_temp.started.connect(self.worker_temp.run) # when the thread is started,...
            self.worker_temp.finished.connect(self.thread_temp.quit) # when the worker is finish...
            self.worker_temp.finished.connect(self.worker_temp.deleteLater)
            self.thread_temp.finished.connect(self.thread_temp.deleteLater)
            self.thread_temp.start() #here the thread is actually started
            print("update started")
        elif self.tempCheck.isChecked() == False:
            self.tempCheck.setText("Current temp: ??")
            self.stop_update.emit()
            print("update stopped")# end temp update
            pass
    else:
        self.print_to_message_box("Qpod not connected.  ")
    return

def stop_update_func(self):
    self.update_temp = False

#### MULTI SAMPLE CONTROL
#### PART CONTAINING CONTROL OF KNAUER SELECTION VALVE AND SF10 PUMP

@pyqtSlot()
def on_samplerConnect_clicked(self):
    for pinfo in serial.tools.list_ports.comports():
        print (pinfo.name, pinfo.serial_number, pinfo.vid)
        if pinfo.serial_number == 'KNE01QKMA' and pinfo.vid == 1027: #Knauer valve
            self.valve = serial.Serial(pinfo.device)
            serialcmd = str(f'POSITION:{5}\r')
            self.valve.write(serialcmd.encode())
```

```
                # serialcmd = str(f'POSITION?\r')
                # self.valve.write(serialcmd.encode())
                # print(self.valve.readline().strip().decode("latin-1"))
            self.valve_connected = True
    return

@pyqtSlot()
def on_Valve_Select_Button_clicked(self):
    # if self.valve_position == int(self.ValveList.currentText()):
    #     return
    self.valve_position = int(self.ValveList.currentText())
    serialcmd = str(f'POSITION:{self.valve_position}\r')
    self.valve.write(serialcmd.encode())
    self.valve.read_until(expected='\r'.encode('UTF-8'))
    time.sleep(0.5)
    serialcmd = str(f'POSITION?\r')
    self.valve.write(serialcmd.encode())
    line = self.valve.read_until(expected='\r'.encode('UTF-8')).decode("latin-1")
    self.valve_position = int(line.split('POSITION:')[1].split('\r')[0])

    self.print_to_message_box(f"Valve position set to {self.valve_position}")

@pyqtSlot()
def on_pumpConnect_clicked(self):
    for pinfo in serial.tools.list_ports.comports():
        print (pinfo.name, pinfo.serial_number, pinfo.vid)
        if pinfo.serial_number == '6' and pinfo.vid == 1027: #SF10 pump
            self.pump_solvent = serial.Serial(pinfo.device)
            serialcmd = str(f'GV\r')
            self.pump_solvent.write(serialcmd.encode())
            print(self.pump_solvent.readline().strip().decode("latin-1"))
            serialcmd = str(f'REMOTEEN vap9 1\r')
            self.pump_solvent.write(serialcmd.encode())
            print(self.pump_solvent.readline().strip().decode("latin-1"))
            serialcmd = str(f'MODE DOSE\r')
            self.pump_solvent.write(serialcmd.encode())
            serialcmd = str(f'START\r')
            self.pump_solvent.write(serialcmd.encode())
            serialcmd = str(f'STOP\r')
            self.pump_solvent.write(serialcmd.encode())
            print(self.pump_solvent.readline().strip().decode("latin-1"))
            print("Pump connected")
            self.Pump_button.setEnabled(True)
            self.Pump = True
    return

@pyqtSlot()
def on_Load_Button_clicked(self):
    rate = self.Pump_flowrate.value() # flow rate in ml/min
    vol = self.Load_Volume.value() #in ml
    self.print_to_message_box(f"Loading {vol} ml at {rate} ml/min, valve {self.valve_position}")
    serialcmd = str(f'MODE DOSE\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'SETFLOW {rate}\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'SETDOSE {vol}\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'SETREG 3.0\r')
    self.pump_solvent.write(serialcmd.encode())
    serialcmd = str(f'START\r')
    self.pump_solvent.write(serialcmd.encode())
    time.sleep(60*vol/rate) #in seconds
    serialcmd = str(f'STOP\r')
    self.pump_solvent.write(serialcmd.encode())
    return

@pyqtSlot()
```

```
def on_Clean_Button_clicked(self):
    if self.Knauer:
        self.valve_position = 16
        serialcmd = str(f'POSITION:{self.valve_position}\r')
        self.valve.write(serialcmd.encode())
    if self.SF10:
        rate = 1 # flow rate in ml/min
        vol = self.Clean_Volume.value() #in ml
        self.print_to_message_box(f"Cleaning {vol} ml at {rate} ml/min, valve {self.valve_positi...
        serialcmd = str(f'MODE DOSE\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETFLOW {rate}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETDOSE {vol}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETREG 3.0\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'START\r')
        self.pump_solvent.write(serialcmd.encode())
        time.sleep(60*vol/rate) #in seconds
        serialcmd = str(f'STOP\r')
        self.pump_solvent.write(serialcmd.encode())
    else:
        self.print_to_message_box("No pump connected")

@pyqtSlot()
def on_Save_Valve_Presets_Button_clicked(self):
    for i in range(1,17):
        self.Valve_Presets[i-1] = self.__dict__['Sample_name_' + str(i)].text()
    with open('Valve_Presets.json', 'w') as file:
        json.dump(self.Valve_Presets, file)

@pyqtSlot()
def on_Clear_Valve_Presets_Button_clicked(self):
    for i in range(1,17):
        self.__dict__['Sample_name_' + str(i)].clear()


#### SHELLY POWER PLUG CONTROL
@pyqtSlot()
def on_shellyConnect_clicked(self):
    # self.Shelly_on.setEnabled(True)
    url = f'http://{self.Shelly_IP.text()}/rpc/Switch.GetStatus?id=0'
    print(url)
    try:
        # Make a GET request to the Shelly device's status URL
        response = requests.get(url,timeout=3)
        # Check for successful response
        if response.status_code == 200:
            print(response.json()) # Parse the JSON response and check for output
            if response.json().get('output'):
                self.Shelly_off.setEnabled(True)
            else:
                self.Shelly_on.setEnabled(True)
    except Exception as e:
        print(f"An error occurred: {e}")
    return

@pyqtSlot()
def on_Shelly_on_clicked(self):
    try:
        response = requests.get(f'http://{self.Shelly_IP.text()}/relay/0?turn=on',timeout=2) #tu...
        if response.json().get('ison') == True:
            self.Shelly_on.setEnabled(False)
            self.Shelly_off.setEnabled(True)
            self.plugLabel.setText("Plug status: ON")
    except:
```

```
            self.print_to_message_box("Shelly smart plug disconnected. Check if mobile hotspot is on...

    @pyqtSlot()
    def on_Shelly_off_clicked(self,check=True):
        if check:
            off_msg = "Are you sure you want to turn off the power supply for the lamp? You will not...
            reply = QMessageBox.question(self, 'Warning', off_msg, QMessageBox.Yes, QMessageBox.No)
            if reply == QMessageBox.No:
                return
        else:
            pass
        try:
            #turn off shelly device
            response = requests.get(f'http://{self.Shelly_IP.text()}/relay/0?turn=off',timeout=2)
            if response.json().get('ison') == False:
                self.Shelly_on.setEnabled(True)
                self.Shelly_off.setEnabled(False)
                self.plugLabel.setText("Plug status: OFF")
        except:
            self.print_to_message_box("Shelly smart plug disconnected. Check if mobile hotspot is on...


    #### PART CONTAINING PUMP CONTROL
    @pyqtSlot()
    def on_Pump_button_clicked(self):
        rate = 1000 # flow rate in ul/min
        vol = self.Pump_vol_Box.value() #in ul
        self.print_to_message_box(f"Pushing {vol} ul at {rate} ul/min")
        serialcmd = str(f'MODE DOSE\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETFLOW {rate/1000}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETDOSE {vol/1000}\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'SETREG 3.0\r')
        self.pump_solvent.write(serialcmd.encode())
        serialcmd = str(f'START\r')
        self.pump_solvent.write(serialcmd.encode())
        time.sleep(60*vol/rate) #in seconds
        serialcmd = str(f'STOP\r')
        self.pump_solvent.write(serialcmd.encode())
        return

#### PART CONTAINING LED SWITCH AUTOMATION
    @pyqtSlot()
    def update_power(self):
        time.sleep(0.1)
        try:
            upHandle = c_long(int(self.LEDlist.currentText().split(".")[0]))
            currentSetpoint = c_double(0)
            ret = lib.TLUP_getLedOutputState(upHandle)
            # print(ret)
            if ret == 1:
                lib.TLUP_getLedCurrentSetpoint(upHandle,0, byref(currentSetpoint))
                self.LEDcurrentpower.setText(f"Current power: {currentSetpoint.value}")
        except:
            self.print_to_message_box("No LED chosen.")
        return

    @pyqtSlot()
    def on_connectLED_clicked(self):
        self.print_to_message_box("Trying to connect to LEDs.")
        deviceCount = c_uint32()
        lib.TLUP_findRsrc(0,byref(deviceCount))
        if deviceCount.value > 0:
            self.print_to_message_box("Number of upSeries devices found: " + str(deviceCount.value))
        else:
```

```
            self.print_to_message_box("No upSeries devices found.")
            return
        # print()
        modelName = create_string_buffer(256)
        serialNumber = create_string_buffer(256)
        lib.TLUP_getRsrcInfo(0, 0, modelName, serialNumber, 0, 0)
        # print("Connecting to this device:")
        self.print_to_message_box(f"Model name: {modelName.value.decode()}, Serial number: {serialNu...
        # print()
        self.upNames = [0 for i in range(deviceCount.value)]
        self.upHandles = [0 for i in range(deviceCount.value)]

        for i in range(deviceCount.value):
            # print(i)
            #Initializing the first connected upSeries device.
            self.upNames[i] = create_string_buffer(256)
            lib.TLUP_getRsrcName(0, i, self.upNames[i])
            self.upHandles[i]=c_int(0)
            res=lib.TLUP_init(self.upNames[i].value, 0, 0, byref(self.upHandles[i]))
            # print(f"uphandle is: {self.upHandles[i] , self.upHandles[i].value} for {self.upNames[i...
        wls = []

        for i in self.upHandles:
            currentSetpoint = c_double()
            LEDName = create_string_buffer(256)
            LEDSerialNumber = create_string_buffer(256)
            LEDCurrentLimit = c_double()
            LEDForwardVoltage = c_double()
            LEDWavelength = c_double(0)
            lib.TLUP_getLedInfo(i, LEDName, LEDSerialNumber, byref(LEDCurrentLimit),
                        byref(LEDForwardVoltage), byref(LEDWavelength))
            wls.append(f"{i.value}. {LEDWavelength.value}")
        for l in range(self.LEDlist.count()):
            self.LEDlist.removeItem(0)
        self.LEDlist.addItem("Select LED...")
        self.LEDlist.addItems([f"{x} nm" for x in wls])
        self.connectLED.setEnabled(False)
        self.disconnectLED.setEnabled(True)
        return

    @pyqtSlot()
    def on_disconnectLED_clicked(self):
        try:
            for handle in self.upHandles:
                lib.TLUP_close(handle)
            self.connectLED.setEnabled(True)
            self.disconnectLED.setEnabled(False)
        except:
            self.print_to_message_box("No LEDs connected.")
        return

    @pyqtSlot()
    def on_setLED_clicked(self):
        try:
            upHandle = c_long(int(self.LEDlist.currentText().split(".")[0]))

            print(lib.TLUP_setLedUseNonThorlabsLed(upHandle, 1))
            if self.current_power < 20:
                self.current_power = 20
            if self.current_power > 1200:
                self.current_power = 1200

            currentSetpoint = c_double(float(self.current_power*0.001)) #in mA
            lib.TLUP_setLedCurrentSetpoint(upHandle,currentSetpoint)
            time.sleep(0.5)
            print(lib.TLUP_switchLedOutput(upHandle,1))
            print("Switch LED on.")
```

```
        self.LED_stat_text.setText(f"LED status: ON @ {float(self.LEDpower.value())}")
    except:
        print("LED not available.")
    return

@pyqtSlot()
def on_LEDoff_clicked(self): #turns LED OFF
    try:
        upHandle = c_long(int(self.LEDlist.currentText().split(".")[0]))
        # print(lib.TLUP_setLedUseNonThorlabsLed(upHandle, 1))
        lib.TLUP_switchLedOutput(upHandle,0)
        self.LED_stat_text.setText("LED status: OFF")
        # print("Switch LED off.")
    except:
        print("You didnt choose an LED!!!")
    return


#########################################################################################################...

# SPECTROMETER CONTROL BELOW

@pyqtSlot()
def on_log_file_btn_clicked(self):
    text, ok = QInputDialog().getText(self, "QInputDialog().getText()",
                                      "Log to:", QLineEdit.Normal,
                                      "Filename")
    if ok and text:
        self.log_name = text
        self.log_name_label.setText("Saving to: "+text)
    #something about open dialog
    return

@pyqtSlot()
def on_OpenCommBtn_clicked(self):
    try:
        ret = AVS_Init(0)
        # QMessageBox.information(self,"Info","AVS_Init returned:  {0:d}".format(ret))
        ret = AVS_GetNrOfDevices()
        # QMessageBox.information(self,"Info","AVS_GetNrOfDevices returned:  {0:d}".format(ret))
        req = 0
        mylist = AvsIdentityType * 1
        ret = AVS_GetList(75, req, mylist)
        serienummer = str(ret[1].SerialNumber.decode("utf-8"))
        if serienummer:
            QMessageBox.information(self,"Info","Found spectrometer with Serialnumber: " + serie...
        else:
            self.print_to_message_box("Could not find spectrometer. Try again.")
            return
        globals.dev_handle = AVS_Activate(ret[1])
        # QMessageBox.information(self,"Info","AVS_Activate returned:  {0:d}".format(globals.dev...
        devcon = DeviceConfigType
        reqsize = 0
        ret = AVS_GetParameter(globals.dev_handle, 63484, reqsize, devcon)
        globals.pixels = ret[1].m_Detector_m_NrPixels
        # print(f'length of globals pixels in: {globals.pixels}')
        ret = AVS_GetLambda(globals.dev_handle,globals.wavelength)
        x = 0
        self.wavelength = np.array(ret[:globals.pixels])
        np_round_to_tenths = np.around(self.wavelength, 1)
        globals.wavelength = list(np_round_to_tenths)

        self.measconfig = MeasConfigType
        self.measconfig.m_StartPixel = 0
        self.measconfig.m_StopPixel = globals.pixels - 1
        self.measconfig.m_IntegrationTime = 0
        self.measconfig.m_IntegrationDelay = 0
```

```
        self.measconfig.m_NrAverages = 0
        self.measconfig.m_CorDynDark_m_Enable = 1  # nesting of types does NOT work!!
        self.measconfig.m_CorDynDark_m_ForgetPercentage = 100
        self.measconfig.m_Smoothing_m_SmoothPix = 2
        self.measconfig.m_Smoothing_m_SmoothModel = 0
        self.measconfig.m_SaturationDetection = 0
        self.measconfig.m_Trigger_m_Mode = 0
        self.measconfig.m_Trigger_m_Source = 0
        self.measconfig.m_Trigger_m_SourceType = 0
        self.measconfig.m_Control_m_StrobeControl = 0
        self.measconfig.m_Control_m_LaserDelay = 0
        self.measconfig.m_Control_m_LaserWidth = 0
        self.measconfig.m_Control_m_LaserWaveLength = 0.0
        self.measconfig.m_Control_m_StoreToRam = 0

        if globals.pixels != 0:
            self.StartMeasBtn.setEnabled(False)
            self.StopMeasBtn.setEnabled(True)
            self.OpenCommBtn.setEnabled(False)
            self.CloseCommBtn.setEnabled(True)
            self.getLightBtn.setEnabled(True)
            self.loadRefBtn.setEnabled(True)
    except:
        self.print_to_message_box("No device found.")

    return

@pyqtSlot()
def on_CloseCommBtn_clicked(self):
    callbackclass.callback(self, 0, 0)
    self.StartMeasBtn.setEnabled(False)
    self.StopMeasBtn.setEnabled(False)
    self.OpenCommBtn.setEnabled(True)
    self.CloseCommBtn.setEnabled(False)
    return

    FPGAver = bytes(VERSION_LEN)
    FWver = bytes(VERSION_LEN)
    DLLver = bytes(VERSION_LEN)
    ret = AVS_GetVersionInfo(globals.dev_handle, FPGAver, FWver, DLLver)
    FPGAver = ret[0]
    FWver = ret[1]
    DLLver = ret[2]
    QMessageBox.information(self,"Info","FPGA version: {FPGA} \nFirmware version: {FW} \nDLL ver...
                            .format(FPGA=FPGAver.value.decode('utf-8'),
                                    FW=FWver.value.decode('utf-8'),
                                    DLL=DLLver.value.decode('utf-8')))
    return

@pyqtSlot()
def on_StartMeasBtn_clicked_old(self):
    self.StartMeasBtn.setEnabled(False)
    ret = AVS_UseHighResAdc(globals.dev_handle, True)
    self.measconfig.m_IntegrationTime = float(self.IntTimeEdt.text()) #set exposure
    self.measconfig.m_NrAverages = int(self.NumAvgEdt.text()) #set number of averages
    ret = AVS_PrepareMeasure(globals.dev_handle, self.measconfig)
    timestamp = 0
    nummeas = int(self.NumMeasEdt.text())
    self.cancelled = False

    scans = 0

    if nummeas == 0:
        nummeas=100000
    while (scans < nummeas):
        if (self.cancelled == False):
```

```
            ret = AVS_Measure(globals.dev_handle, 0, 1)
            dataready = False
            while (dataready == False):
                dataready = (AVS_PollScan(globals.dev_handle) == True)
                time.sleep(0.001)
            if dataready == True:
                ret = AVS_GetScopeData(globals.dev_handle, timestamp, globals.spectraldata)
                timestamp = ret[0]
                x = 0
                for x in range(globals.pixels): # 0 through 2047
                    globals.spectraldata[x] = int(ret[1][x])
                globals.spectraldata = [globals.spectraldata[x] for x in range(globals.pixels)]

                scans = scans + 1
                self.time_spectrum = time.time()
                self.newdata.emit()
                time.sleep(0.3)

        else:
            break
    self.StartMeasBtn.setEnabled(True)
    return

@pyqtSlot()
def on_StartMeasBtn_clicked(self): #activates when the button "Start Measurement" is clicked
    self.first = False ### THIS MUST BE FALSE OTHERWISE THE START MEASUREMENT BUTTON WILL OVERWR...
    try:
        if self.thread.isRunning():
            print("Shutting down running thread.")
            self.thread.terminate()
            time.sleep(1)
        else:
            print("No thread was running.")
    except:
        print("Didn't find thread.")
    self.thread = QThread() # this created an additional computing thread for processes, so the ...
    self.worker = Worker() # this is a worker that will tell when the job is done
    self.worker.func = self.on_StartMeasBtn_clicked_old #here the job of the worker is defined. ...
    self.worker.moveToThread(self.thread) #the workers job is moved from the frontend to the thr...
    self.thread.started.connect(self.worker.run) # when the thread is started, the worker runs
    self.worker.finished.connect(self.thread.quit) # when the worker is finished, then the threa...
    self.worker.finished.connect(self.worker.deleteLater)
    self.thread.finished.connect(self.thread.deleteLater)
    self.thread.start() #here the thread is actually started
    return

@pyqtSlot()
def on_StopMeasBtn_clicked(self):
    self.cancel.emit()
    time.sleep(1)

    return

@pyqtSlot()
def cancel_meas(self):
    self.cancelled = True
    return

@pyqtSlot()
def handle_newdata(self):
    # self.label_5.setText(
    #     f"Estimated time required: {(float(self.IntTimeEdt.text())*int(self.NumAvgEdt.text())*...
    try:
        self.time_off_total += self.time_off_end
        self.new_time =  self.time_spectrum - self.t0 - self.time_off_total

        self.Spectrum_figure.axes.cla()
```

```
            self.Spectrum_figure.axes.plot(globals.wavelength,globals.spectraldata)
            self.Spectrum_figure.axes.set_xlabel("Wavelength [nm]")
            self.Spectrum_figure.axes.set_ylabel("Intensity")
            self.Spectrum_figure.draw()

            self.save_data(save_pdf=True)
        except:
            print("new data was not handled")
        return

    @pyqtSlot()
    def save_data(self,save_pdf = False):
        try:
            if self.use_light == True:
                try:
                    globals.spectral_minus_light = [log(globals.light_spectrum[x] / globals.spectral...
                except:
                    globals.spectral_minus_light = np.zeros(globals.pixels)
                # globals.spectral_minus_light = [log(globals.light_spectrum[x] / globals.spectralda...

                # globals.spectral_minus_light = [0.0 - globals.spectraldata[x] + globals.light_spec...
                if self.first:
                    with open(self.log_path,'w') as f:
                        f.write(f"Temperature: {self.Temperature_box.value()} C, Exposure time: {flo...
                        f.write('light '  + 'spectrum ' + ' '.join([str(globals.light_spectrum[x]) f...
                        f.write('0 '  + '0 ' + ' '.join([str(globals.wavelength[x]) for x in range(g...
                        f.write('0 ' +str(self.LEDpower.value())+' ' + ' '.join([str(globals.spectra...
                else:
                    with open(self.log_path,'a') as f:
                        f.write(str(self.new_time)+' '+ f"{self.TBC_started}"+ ' ' + ' '.join([str(g...
                if save_pdf == True:
                    self.abs_figure.axes.cla()
                    self.abs_figure.axes.plot(globals.wavelength[:2048], [globals.spectral_minus_lig...
                    self.abs_figure.axes.set_xlim(left=self.Min_wl_box.value(), right=self.Max_wl_bo...
                    self.abs_figure.axes.set_xlabel("Wavelength [nm]")
                    self.abs_figure.axes.set_ylabel("Absorbance [OD]")
                    self.abs_figure.draw()

                    # plot new dynamics
                    self.Plot_new_dynamics()
                else:
                    print("you didnt choose reference light spectrum")
                self.first = False
            except:
                print("data was not saved")
            return

################################################## PLOT Values real time. From TTAUC script #########...

    @pyqtSlot()
    def on_Set_wavelength_btn_clicked(self): #set values to plot and background value to subtract
        self.wavelengths_int = [int(eval(i)) for i in self.Wavelength_line.text().split(",")]
        self.background = int(self.Background_line.text())
        return

    @pyqtSlot()
    def Plot_new_dynamics(self): #also removes absorption values and can be used to calculate thresh...
        try:
            pddata = pd.read_csv(self.log_path, sep=' ', header=2) #2 for two lines of metadata and ...
            meta = 5 #how many places reserved for metadata in the columns
            wavelengths_pd = np.array(list(pddata)[meta:])
            wavelength_axis = np.around(wavelengths_pd.astype(float),1)
            bg_index = (np.abs(wavelength_axis - self.background)).argmin() + meta
            Data_points = len(pddata.iloc[:,0])
            Dynamics = np.zeros(Data_points)
            Time = np.zeros(Data_points)
            self.time_figure.axes.clear()
```

```
            for wl in range(len(self.wavelengths_int)): # pl vs time plot
                wl_index = (np.abs(wavelength_axis - self.wavelengths_int[wl])).argmin() + meta
                for i in range(Data_points):
                    Time[i] = np.array(pddata.iloc[i,0])
                    Dynamics[i] = pddata.iloc[i,wl_index] - pddata.iloc[i,bg_index]
                Time_axis = np.around(Time.astype(float),2)
                # self.time_figure.axes.cla()
                self.time_figure.axes.plot(Time_axis,Dynamics,label=wavelength_axis[wl_index-meta])

            self.time_figure.axes.legend()
            self.time_figure.axes.set_xlabel("Time [s]")
            self.time_figure.axes.set_ylabel("Absorbance")
            self.time_figure.draw()
        except:
            print("no dynamics plotted")
            return
        return

##########################################################################################################...

    @pyqtSlot()
    def on_setPowerBtn_clicked(self): #sets the power that the LED will use when turned on
        try:
            self.current_power = float(self.LEDpower.value())
            self.LEDcurrentpower.setText(f"Current power: {self.current_power}")
        except:
            self.print_to_message_box("Could not set power.")
        return

    @pyqtSlot()
    def on_showRefBtn_clicked(self): #shows the reference spectum in the raw spectrum plot
        self.Spectrum_figure.axes.cla()
        self.Spectrum_figure.axes.plot(globals.wavelength[0:2048],globals.light_spectrum[0:2048],lab...
        self.Spectrum_figure.axes.set_xlabel("Wavelength [nm]")
        self.Spectrum_figure.axes.set_ylabel("Intensity")
        self.Spectrum_figure.figure.tight_layout()
        self.Spectrum_figure.draw()
        return

    @pyqtSlot()
    def on_saveRefBtn_clicked(self): #saves the reference spectrum
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self,"QFileDialog.getSaveFileName()","","Text File...
        if fileName:
            self.print_to_message_box(f"Saving reference spectrum to: {fileName}")
        else:
            self.print_to_message_box("No filename was selected and therefore the reference spectrum...
            return
        with open(fileName,"w") as f:
            f.write(' '.join([str(x) for x in globals.wavelength[0:2048]])+'\n'+' '.join(str(x) for ...

        return

    @pyqtSlot()
    def on_loadRefBtn_clicked(self): #loads a reference spectrum previously saved
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getOpenFileName(self,"QFileDialog.getSaveFileName()","","Text File...

        if fileName:
            self.print_to_message_box(f"Loading reference spectrum from: {fileName}")
        else:
            self.print_to_message_box("No filename was selected and therefore the reference spectrum...
            return
        try:
            pddata_ref = pd.read_csv(fileName, sep=' ', header=0)
            globals.light_spectrum = np.array(pddata_ref.iloc[0,:])
            self.use_light = True
```

```
    except:
        self.print_to_message_box("The reference spectrum could not be loaded...")
        return
    self.showRefBtn.setEnabled(True)
    self.saveRefBtn.setEnabled(True)
    self.StartMeasBtn.setEnabled(True)
    return

@pyqtSlot()
def on_getLightBtn_clicked(self):
    self.StartMeasBtn.setEnabled(False)
    try:
        self.on_lampOnBtn_clicked()
        time.sleep(0.5)
    except:
        print("Lamp is not connected...")
    ret = AVS_UseHighResAdc(globals.dev_handle, True)
    measconfig = MeasConfigType
    measconfig.m_StartPixel = 0
    measconfig.m_StopPixel = globals.pixels - 1
    measconfig.m_IntegrationTime = float(self.IntTimeEdt.text())
    measconfig.m_IntegrationDelay = 0
    measconfig.m_NrAverages = int(1000/float(self.IntTimeEdt.text()))
    measconfig.m_CorDynDark_m_Enable = 1  # nesting of types does NOT work!!
    measconfig.m_CorDynDark_m_ForgetPercentage = 0
    measconfig.m_Smoothing_m_SmoothPix = 0
    measconfig.m_Smoothing_m_SmoothModel = 0
    measconfig.m_SaturationDetection = 0
    measconfig.m_Trigger_m_Mode = 0
    measconfig.m_Trigger_m_Source = 0
    measconfig.m_Trigger_m_SourceType = 0
    measconfig.m_Control_m_StrobeControl = 0
    measconfig.m_Control_m_LaserDelay = 0
    measconfig.m_Control_m_LaserWidth = 0
    measconfig.m_Control_m_LaserWaveLength = 0.0
    measconfig.m_Control_m_StoreToRam = 0
    ret = AVS_PrepareMeasure(globals.dev_handle, measconfig)
    nummeas = int(self.NumMeasEdt.text())
    self.cancelled = False
    timestamp = 0
    ret = AVS_Measure(globals.dev_handle, 0, 1)
    dataready = False
    while (dataready == False):
        dataready = (AVS_PollScan(globals.dev_handle) == True)
        time.sleep(0.001)
    if dataready == True:
        ret = AVS_GetScopeData(globals.dev_handle, timestamp, globals.spectraldata )
        wl = AVS_GetLambda(globals.dev_handle, globals.wavelength)
        timestamp = ret[0]
        x = 0
        while (x < globals.pixels): # 0 through 2047
            globals.spectraldata[x] = ret[1][x]
            x += 1

    light = [globals.spectraldata[x] for x in range(globals.pixels)]
    globals.light_spectrum = light

    self.use_light = True
    try:
        time.sleep(0.5)
        self.on_lampOffBtn_clicked()
    except:
        print("Lamp is still not connected...")
    self.StartMeasBtn.setEnabled(True)
    self.saveRefBtn.setEnabled(True)
    self.showRefBtn.setEnabled(True)
```

```
    self.on_showRefBtn_clicked() #show ref spectrum

    return

@pyqtSlot()
def on_laserOnBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 7, 1) #OPENS CHANNEL 1 WHICH CORRESPONDS TO DO8
        self.channel1_text.setText("Laser status: On")
    except:
        self.channel1_text.setText("ERROR")
    return

@pyqtSlot()
def on_laserOffBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 7, 0) #CLOSES CHANNEL 1 WHICH CORRESPONDS TO DO8
        self.channel1_text.setText("Laser status: Off")
    except:
        self.channel1_text.setText("ERROR")
    return

@pyqtSlot()
def on_lampOnBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 3, 1) #OPENS CHANNEL 2 WHICH CORRESPONDS TO DO5
        self.channel2_text.setText("Lamp status: On")
        # self.print_to_message_box("Lamp shutter opened.")
    except:
        self.channel2_text.setText("ERROR")
    return

@pyqtSlot()
def on_lampOffBtn_clicked(self):
    try:
        ret = AVS_SetDigOut(globals.dev_handle, 3, 0) #CLOSES CHANNEL 2 WHICH CORRESPONDS TO DO5
        self.channel2_text.setText("Lamp status: Off")
        # self.print_to_message_box("Lamp shutter closed.")
    except:
        self.channel2_text.setText("ERROR")
    return

@pyqtSlot()
def runQY(self):
    self.startQYBtn.setEnabled(False)
    self.qyRunning.setText("Experiment running...")
    if self.NumMeasEdt.text() != "1":
        self.NumMeasEdt.setText("1")
    self.t0 = time.time()
    self.time_off_total = 0
    self.time_off_start = 0
    self.time_off_end = 0
    self.time_spectrum = 0
    meas_interval = int(self.intervalLine.text())
    TBC_interval = int(self.TBC_interval_box.value())
    blank_measurements = self.Blank_meas_num.value()
    self.qy_cancelled = False
    self.TBC_started = False
    if self.qyPointsSpinBox.value() > 0:
        num_points = self.qyPointsSpinBox.value()
    else:
        num_points = 100000
    for i in range(num_points):
        if self.qy_cancelled == True: #Cancel if Stop button was pressed
            self.qyRunning.setText("Experiment not running.")
            self.on_lampOffBtn_clicked()
            self.on_LEDoff_clicked()
```

```
            self.startQYBtn.setEnabled(True)
            self.print_to_message_box("QY experiment stopped (manual).")
            return

        elif self.TBC_started == True: #Begin backconversion if button was pressed
            self.time_off_start = time.time()
            self.on_lampOnBtn_clicked()
            time.sleep(0.1)
            self.on_StartMeasBtn_clicked_old(scans=1)
            time.sleep(0.2)
            self.on_lampOffBtn_clicked()
            time.sleep(0.1)
            # self.time_off_end = time.time()-self.time_off_start

            self.time_off_end = 0 # This has to be 0 to obtain the proper time of the back conve...
            try:
                TBC_interval_new = int(self.TBC_interval_box.value())
            except:
                TBC_interval_new = TBC_interval
                pass
            for b in range(int(TBC_interval_new)):
                time.sleep(1)
                if self.qy_cancelled == True:
                    break

        else: #continue with isomerisation measurement
            self.time_off_start = time.time()
            time.sleep(0.1)
            self.on_lampOnBtn_clicked()
            time.sleep(0.1)
            self.on_StartMeasBtn_clicked_old(scans=1)
            time.sleep(0.1)
            self.on_lampOffBtn_clicked()
            time.sleep(0.1)
            if i > blank_measurements: #dont turn on LED for initial set of measurements
                self.on_setLED_clicked()
            self.time_off_end = time.time()-self.time_off_start
            print(f"Time off is: {self.time_off_end}")
            try:
                meas_interval_new = int(self.intervalLine.text())
            except:
                meas_interval_new = meas_interval
                pass
            lamp_warm_time = self.lampWarmSpin.value()
            if self.lampToggleCheck.isChecked():
                self.on_Shelly_off_clicked(check=False)
                print("Shelly off")
            for b in range(int(meas_interval_new)):
                time.sleep(1)
                if self.qy_cancelled == True:
                    break
                if b == int(meas_interval_new-lamp_warm_time):
                    if self.lampToggleCheck.isChecked():
                        self.on_Shelly_on_clicked()
                        print("Shelly on")
            self.on_LEDoff_clicked()




self.on_lampOffBtn_clicked()
self.on_LEDoff_clicked()
self.qyRunning.setText("Experiment not running.")
self.startQYBtn.setEnabled(True)
self.print_to_message_box("QY experiment finished succesfully.")
return
```

```
@pyqtSlot()
def on_TBC_start_clicked(self):
    self.start_TBC.emit()
    self.print_to_message_box("Thermal back-conversion experiment started")
    return

@pyqtSlot()
def set_start_TBC(self):
    self.TBC_started = True
    return

@pyqtSlot()
def set_qy_cancel(self):
    self.qy_cancelled = True
    return

@pyqtSlot()
def on_stopQYBtn_clicked(self):
    self.cancel_qy.emit()
    self.print_to_message_box("QY experiment end requested. Shutting down.")
    return

@pyqtSlot()
def on_startQYBtn_clicked(self):

    #This is where you select the log file name
    options = QFileDialog.Options()
    fileName, _ = QFileDialog.getSaveFileName(self,"QFileDialog.getSaveFileName()","","All Files...
    if fileName:
        self.print_to_message_box(f"Saving to: {fileName}")
        self.log_path = fileName
    else:
        self.print_to_message_box("No filename was selected and therefore the experiment has not...
        self.startQYBtn.setEnabled(True)
        return

    self.print_to_message_box("QY experiment started.")
    self.first = True
    meas_interval = self.intervalLine.text()
    self.get_abs_spectrum = True
    # self.ratio_t0 = time.time()
    self.t0 = time.time()

    try:
        if self.thread_qy.isRunning():
            print("Shutting down running thread.")
            self.thread_qy.terminate()
            time.sleep(1)
        else:
            print("No thread was running.")
    except:
        print("Didn't find thread.")
    self.thread_qy = QThread() # this created an additional computing thread for processes, so t...
    self.worker_qy = Worker() # this is a worker that will tell when the job is done
    self.worker_qy.func = self.runQY #here the job of the worker is defined. it should only be o...
    self.worker_qy.moveToThread(self.thread_qy) #the workers job is moved from the frontend to t...
    self.thread_qy.started.connect(self.worker_qy.run) # when the thread is started, the worker ...
    self.worker_qy.finished.connect(self.thread_qy.quit) # when the worker is finished, the the ...
    self.worker_qy.finished.connect(self.worker_qy.deleteLater)
    self.thread_qy.finished.connect(self.thread_qy.deleteLater)
    self.thread_qy.start() #here the thread is actually started
    return

##########################################################################################################...
### ALL BELOW IS VAPOURTEC
@pyqtSlot()
def on_OpenCommBtn_vap_clicked(self):
```

```
        if self.checkA.isChecked() == False and self.checkB.isChecked() == False and self.checkC.isC...
            print("Remember to pick which pumps you want to connect to")
            return
        print("This button should connect to the Vapourtec")
        self.client = rs.RSeriesClient('opc.tcp://localhost:43344')
        self.conState = self.client.connect()
        # print(conState)
        self.rseries = self.client.getRSeries()
        self.manualControl = self.rseries.getManualControl()
        # self.reactor = self.manualControl.getR4I().getReactors()['3']
        print(f"Pumps available are {self.manualControl.getR2Secondary().getPumps()}")
        # print(f"Pump selected is {self.pump_name.currentText()}")
        if self.checkA.isChecked():
            self.pumpA = self.manualControl.getR2Primary()._getPump(f"A")
        if self.checkB.isChecked():
            self.pumpB = self.manualControl.getR2Primary()._getPump(f"B")
        if self.checkC.isChecked():
            self.pumpC = self.manualControl.getR2Secondary()._getPump(f"A")
        if self.checkD.isChecked():
            self.pumpD = self.manualControl.getR2Secondary()._getPump(f"B")
        # self.pump = self.manualControl.getR2Secondary()._getPump(f"{self.pump_name.currentText()}"...
        # self.temperature = self.reactor.getTemperature()
        if self.pumpD.getValveSRState():
            self.switchSRBtn_D.setText(f"Switch SR. Current: R")
        else:
            self.switchSRBtn_D.setText("Switch SR. Current: S")
        time.sleep(1)
        # self.pump.setValveSRState(True)
        self.StartFlowBtn_D.setEnabled(True)
        self.StopFlowBtn_D.setEnabled(True)
        self.StartFlowBtn_C.setEnabled(True)
        self.StopFlowBtn_C.setEnabled(True)
        self.CloseCommBtn_vap.setEnabled(True)
        self.switchSRBtn_D.setEnabled(True)
        self.switchSRBtn_C.setEnabled(True)
        self.getFlow.setEnabled(True)
        return

    @pyqtSlot()
    def on_CloseCommBtn_vap_clicked(self):
        print("This should close the connection to Vapourtec")
        print('stopping manual control')
        self.manualControl.stopAll()
        print('turn off pump and reactor')
        self.pumpD.setFlowRate(0)
        # self.temperature.setTemperature(25) #This is turned off for testing

        if self.conState:
            self.client.disconnect()
        return

    @pyqtSlot()
    def show_help_oxy(self):
        QMessageBox.information(self, "Info", "Setup for Oxygen level experiment.\
             \nSet Integration time to 500 ms\nSet Number of averages to 5\nSet Number of measuremen...
             \nThe Pump chosen should be B, becuase pump D is apparently called B on the Secondary R...
             \nFor now you can change the flow rate manually by selecting the flow rate and clicking...
        return

class MplCanvas(FigureCanvasQTAgg):

    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)

class Worker(QObject):
```

```
    finished = pyqtSignal()
    func = None
    def run(self):
        self.func()
        self.finished.emit()
        return

def main():
    app = QApplication(sys.argv)
    app.lastWindowClosed.connect(app.quit)
    app.setApplicationName("Quantum Yield Program")
    form = MainWindow()
    form.show()
    app.exec_()

if __name__ == "__main__":
    main()
```

FILE 8/11: lib/forms/55617146_readme.md

```
forms for window
```

FILE 9/11: lib/windows/55617131_readme.md

[Empty file]

FILE 10/11: lib/windows/QYA_f_win.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'QYA_f.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1572, 825)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.runCalcBtn = QtWidgets.QPushButton(self.centralwidget)
        self.runCalcBtn.setGeometry(QtCore.QRect(1000, 210, 121, 41))
        self.runCalcBtn.setObjectName("runCalcBtn")
        self.logTextEdit = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.logTextEdit.setEnabled(True)
        self.logTextEdit.setGeometry(QtCore.QRect(20, 380, 341, 361))
        self.logTextEdit.setReadOnly(True)
        self.logTextEdit.setPlainText("")
        self.logTextEdit.setBackgroundVisible(False)
        self.logTextEdit.setObjectName("logTextEdit")
        self.getFileBtn = QtWidgets.QPushButton(self.centralwidget)
        self.getFileBtn.setGeometry(QtCore.QRect(10, 10, 71, 24))
        self.getFileBtn.setObjectName("getFileBtn")
        self.chosenLabel = QtWidgets.QLabel(self.centralwidget)
        self.chosenLabel.setGeometry(QtCore.QRect(90, 10, 831, 21))
        self.chosenLabel.setObjectName("chosenLabel")
        self.runSimBtn = QtWidgets.QPushButton(self.centralwidget)
        self.runSimBtn.setEnabled(False)
        self.runSimBtn.setGeometry(QtCore.QRect(1000, 260, 121, 41))
        self.runSimBtn.setObjectName("runSimBtn")
        self.simTimeLineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.simTimeLineEdit.setGeometry(QtCore.QRect(1010, 310, 51, 22))
        self.simTimeLineEdit.setObjectName("simTimeLineEdit")
        self.label_11 = QtWidgets.QLabel(self.centralwidget)
        self.label_11.setGeometry(QtCore.QRect(1070, 310, 49, 21))
        self.label_11.setObjectName("label_11")
        self.analysisStartTime = QtWidgets.QSpinBox(self.centralwidget)
        self.analysisStartTime.setGeometry(QtCore.QRect(1110, 140, 42, 22))
        self.analysisStartTime.setMaximum(100000009)
        self.analysisStartTime.setObjectName("analysisStartTime")
        self.analysisEndTime = QtWidgets.QSpinBox(self.centralwidget)
        self.analysisEndTime.setGeometry(QtCore.QRect(1110, 170, 42, 22))
        self.analysisEndTime.setMinimum(-1)
        self.analysisEndTime.setMaximum(100000000)
        self.analysisEndTime.setProperty("value", -1)
        self.analysisEndTime.setObjectName("analysisEndTime")
        self.label_14 = QtWidgets.QLabel(self.centralwidget)
        self.label_14.setGeometry(QtCore.QRect(980, 140, 101, 20))
        self.label_14.setObjectName("label_14")
        self.label_15 = QtWidgets.QLabel(self.centralwidget)
        self.label_15.setGeometry(QtCore.QRect(980, 170, 111, 20))
        self.label_15.setObjectName("label_15")
        self.groupBox_3 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_3.setGeometry(QtCore.QRect(20, 40, 391, 281))
```

```
self.groupBox_3.setObjectName("groupBox_3")
self.qyLineEdit = QtWidgets.QLineEdit(self.groupBox_3)
self.qyLineEdit.setGeometry(QtCore.QRect(130, 110, 61, 22))
self.qyLineEdit.setObjectName("qyLineEdit")
self.qyRadio = QtWidgets.QRadioButton(self.groupBox_3)
self.qyRadio.setGeometry(QtCore.QRect(10, 110, 121, 20))
self.qyRadio.setChecked(True)
self.qyRadio.setAutoExclusive(True)
self.qyRadio.setObjectName("qyRadio")
self.label_8 = QtWidgets.QLabel(self.groupBox_3)
self.label_8.setGeometry(QtCore.QRect(260, 140, 21, 21))
self.label_8.setObjectName("label_8")
self.fluxRadio = QtWidgets.QRadioButton(self.groupBox_3)
self.fluxRadio.setGeometry(QtCore.QRect(10, 140, 121, 20))
self.fluxRadio.setAutoExclusive(True)
self.fluxRadio.setObjectName("fluxRadio")
self.fluxLineEdit = QtWidgets.QLineEdit(self.groupBox_3)
self.fluxLineEdit.setEnabled(True)
self.fluxLineEdit.setGeometry(QtCore.QRect(130, 140, 61, 22))
self.fluxLineEdit.setObjectName("fluxLineEdit")
self.label_7 = QtWidgets.QLabel(self.groupBox_3)
self.label_7.setGeometry(QtCore.QRect(200, 140, 21, 20))
self.label_7.setObjectName("label_7")
self.fluxwlLineEdit = QtWidgets.QLineEdit(self.groupBox_3)
self.fluxwlLineEdit.setEnabled(True)
self.fluxwlLineEdit.setGeometry(QtCore.QRect(220, 140, 31, 22))
self.fluxwlLineEdit.setReadOnly(False)
self.fluxwlLineEdit.setObjectName("fluxwlLineEdit")
self.loadLEDBtn = QtWidgets.QPushButton(self.groupBox_3)
self.loadLEDBtn.setGeometry(QtCore.QRect(290, 140, 75, 24))
self.loadLEDBtn.setObjectName("loadLEDBtn")
self.label_17 = QtWidgets.QLabel(self.groupBox_3)
self.label_17.setGeometry(QtCore.QRect(200, 200, 21, 21))
self.label_17.setObjectName("label_17")
self.label_19 = QtWidgets.QLabel(self.groupBox_3)
self.label_19.setGeometry(QtCore.QRect(10, 200, 111, 21))
self.label_19.setObjectName("label_19")
self.pathLength = QtWidgets.QLineEdit(self.groupBox_3)
self.pathLength.setGeometry(QtCore.QRect(130, 200, 61, 22))
self.pathLength.setObjectName("pathLength")
self.abswlLineEdit = QtWidgets.QLineEdit(self.groupBox_3)
self.abswlLineEdit.setGeometry(QtCore.QRect(130, 230, 61, 22))
self.abswlLineEdit.setObjectName("abswlLineEdit")
self.label_2 = QtWidgets.QLabel(self.groupBox_3)
self.label_2.setGeometry(QtCore.QRect(10, 230, 91, 21))
self.label_2.setObjectName("label_2")
self.groupBox_2 = QtWidgets.QGroupBox(self.groupBox_3)
self.groupBox_2.setGeometry(QtCore.QRect(10, 20, 371, 71))
self.groupBox_2.setTitle("")
self.groupBox_2.setObjectName("groupBox_2")
self.label_5 = QtWidgets.QLabel(self.groupBox_2)
self.label_5.setGeometry(QtCore.QRect(310, 40, 49, 21))
self.label_5.setObjectName("label_5")
self.extincRadio = QtWidgets.QRadioButton(self.groupBox_2)
self.extincRadio.setGeometry(QtCore.QRect(10, 40, 131, 20))
self.extincRadio.setChecked(True)
self.extincRadio.setAutoExclusive(True)
self.extincRadio.setObjectName("extincRadio")
self.concRadio = QtWidgets.QRadioButton(self.groupBox_2)
self.concRadio.setGeometry(QtCore.QRect(10, 10, 121, 20))
self.concRadio.setChecked(False)
self.concRadio.setAutoExclusive(True)
self.concRadio.setObjectName("concRadio")
self.extincLineEdit = QtWidgets.QLineEdit(self.groupBox_2)
self.extincLineEdit.setEnabled(True)
self.extincLineEdit.setGeometry(QtCore.QRect(160, 40, 71, 22))
self.extincLineEdit.setObjectName("extincLineEdit")
```

```
self.concLineEdit = QtWidgets.QLineEdit(self.groupBox_2)
self.concLineEdit.setGeometry(QtCore.QRect(160, 10, 71, 22))
self.concLineEdit.setObjectName("concLineEdit")
self.label_4 = QtWidgets.QLabel(self.groupBox_2)
self.label_4.setGeometry(QtCore.QRect(250, 40, 21, 20))
self.label_4.setObjectName("label_4")
self.extincwlLineEdit = QtWidgets.QLineEdit(self.groupBox_2)
self.extincwlLineEdit.setGeometry(QtCore.QRect(270, 40, 31, 22))
self.extincwlLineEdit.setObjectName("extincwlLineEdit")
self.label_9 = QtWidgets.QLabel(self.groupBox_3)
self.label_9.setGeometry(QtCore.QRect(200, 110, 21, 20))
self.label_9.setObjectName("label_9")
self.label_10 = QtWidgets.QLabel(self.groupBox_3)
self.label_10.setGeometry(QtCore.QRect(260, 110, 21, 21))
self.label_10.setObjectName("label_10")
self.volLineEdit = QtWidgets.QLineEdit(self.groupBox_3)
self.volLineEdit.setGeometry(QtCore.QRect(130, 170, 61, 22))
self.volLineEdit.setObjectName("volLineEdit")
self.label_3 = QtWidgets.QLabel(self.groupBox_3)
self.label_3.setGeometry(QtCore.QRect(10, 170, 111, 21))
self.label_3.setObjectName("label_3")
self.label_21 = QtWidgets.QLabel(self.groupBox_3)
self.label_21.setGeometry(QtCore.QRect(200, 170, 21, 21))
self.label_21.setObjectName("label_21")
self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
self.groupBox.setGeometry(QtCore.QRect(430, 40, 291, 281))
self.groupBox.setObjectName("groupBox")
self.label_13 = QtWidgets.QLabel(self.groupBox)
self.label_13.setGeometry(QtCore.QRect(250, 40, 31, 21))
self.label_13.setObjectName("label_13")
self.zeroLineEdit = QtWidgets.QLineEdit(self.groupBox)
self.zeroLineEdit.setGeometry(QtCore.QRect(180, 40, 61, 22))
self.zeroLineEdit.setObjectName("zeroLineEdit")
self.label_12 = QtWidgets.QLabel(self.groupBox)
self.label_12.setGeometry(QtCore.QRect(20, 30, 131, 41))
self.label_12.setObjectName("label_12")
self.analysisNumPoints = QtWidgets.QSpinBox(self.groupBox)
self.analysisNumPoints.setGeometry(QtCore.QRect(180, 90, 61, 22))
self.analysisNumPoints.setMaximum(100000009)
self.analysisNumPoints.setProperty("value", 15)
self.analysisNumPoints.setObjectName("analysisNumPoints")
self.label_16 = QtWidgets.QLabel(self.groupBox)
self.label_16.setGeometry(QtCore.QRect(20, 90, 161, 21))
self.label_16.setObjectName("label_16")
self.analysiswlLineEdit = QtWidgets.QLineEdit(self.groupBox)
self.analysiswlLineEdit.setGeometry(QtCore.QRect(180, 140, 61, 22))
self.analysiswlLineEdit.setObjectName("analysiswlLineEdit")
self.label = QtWidgets.QLabel(self.groupBox)
self.label.setGeometry(QtCore.QRect(20, 140, 141, 21))
self.label.setObjectName("label")
self.manCorr = QtWidgets.QLineEdit(self.groupBox)
self.manCorr.setGeometry(QtCore.QRect(180, 190, 61, 22))
self.manCorr.setObjectName("manCorr")
self.label_18 = QtWidgets.QLabel(self.groupBox)
self.label_18.setGeometry(QtCore.QRect(20, 180, 131, 41))
self.label_18.setObjectName("label_18")
self.saveParams = QtWidgets.QPushButton(self.groupBox)
self.saveParams.setGeometry(QtCore.QRect(174, 240, 91, 24))
self.saveParams.setObjectName("saveParams")
self.label_20 = QtWidgets.QLabel(self.centralwidget)
self.label_20.setGeometry(QtCore.QRect(30, 350, 31, 21))
self.label_20.setObjectName("label_20")
self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
self.tabWidget.setGeometry(QtCore.QRect(380, 350, 551, 411))
self.tabWidget.setObjectName("tabWidget")
self.tab = QtWidgets.QWidget()
self.tab.setObjectName("tab")
```

```
self.verticalLayoutWidget = QtWidgets.QWidget(self.tab)
self.verticalLayoutWidget.setGeometry(QtCore.QRect(10, 10, 531, 361))
self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
self.fitFigLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
self.fitFigLayout.setSizeConstraint(QtWidgets.QLayout.SetNoConstraint)
self.fitFigLayout.setContentsMargins(0, 0, 0, 0)
self.fitFigLayout.setObjectName("fitFigLayout")
self.tabWidget.addTab(self.tab, "")
self.tab_2 = QtWidgets.QWidget()
self.tab_2.setObjectName("tab_2")
self.verticalLayoutWidget_2 = QtWidgets.QWidget(self.tab_2)
self.verticalLayoutWidget_2.setGeometry(QtCore.QRect(10, 10, 531, 361))
self.verticalLayoutWidget_2.setObjectName("verticalLayoutWidget_2")
self.specFigLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_2)
self.specFigLayout.setSizeConstraint(QtWidgets.QLayout.SetNoConstraint)
self.specFigLayout.setContentsMargins(0, 0, 0, 0)
self.specFigLayout.setObjectName("specFigLayout")
self.tabWidget.addTab(self.tab_2, "")
self.btnExport = QtWidgets.QPushButton(self.centralwidget)
self.btnExport.setGeometry(QtCore.QRect(1090, 340, 75, 24))
self.btnExport.setObjectName("btnExport")
self.groupBox_4 = QtWidgets.QGroupBox(self.centralwidget)
self.groupBox_4.setGeometry(QtCore.QRect(750, 40, 221, 281))
self.groupBox_4.setObjectName("groupBox_4")
self.kCheck = QtWidgets.QCheckBox(self.groupBox_4)
self.kCheck.setGeometry(QtCore.QRect(20, 30, 141, 41))
self.kCheck.setObjectName("kCheck")
self.kLine = QtWidgets.QLineEdit(self.groupBox_4)
self.kLine.setGeometry(QtCore.QRect(40, 80, 71, 22))
self.kLine.setObjectName("kLine")
self.label_6 = QtWidgets.QLabel(self.groupBox_4)
self.label_6.setGeometry(QtCore.QRect(20, 80, 21, 21))
self.label_6.setObjectName("label_6")
self.label_22 = QtWidgets.QLabel(self.groupBox_4)
self.label_22.setGeometry(QtCore.QRect(120, 80, 31, 21))
self.label_22.setObjectName("label_22")
self.prodAbsCheck = QtWidgets.QCheckBox(self.groupBox_4)
self.prodAbsCheck.setGeometry(QtCore.QRect(20, 120, 141, 41))
self.prodAbsCheck.setObjectName("prodAbsCheck")
self.epsProdLine = QtWidgets.QLineEdit(self.groupBox_4)
self.epsProdLine.setGeometry(QtCore.QRect(80, 160, 71, 22))
self.epsProdLine.setObjectName("epsProdLine")
self.label_23 = QtWidgets.QLabel(self.groupBox_4)
self.label_23.setGeometry(QtCore.QRect(10, 160, 61, 21))
self.label_23.setObjectName("label_23")
self.label_24 = QtWidgets.QLabel(self.groupBox_4)
self.label_24.setGeometry(QtCore.QRect(160, 160, 61, 21))
self.label_24.setObjectName("label_24")
self.epsReacLine = QtWidgets.QLineEdit(self.groupBox_4)
self.epsReacLine.setGeometry(QtCore.QRect(80, 190, 71, 22))
self.epsReacLine.setObjectName("epsReacLine")
self.label_27 = QtWidgets.QLabel(self.groupBox_4)
self.label_27.setGeometry(QtCore.QRect(10, 220, 151, 21))
self.label_27.setAlignment(QtCore.Qt.AlignCenter)
self.label_27.setObjectName("label_27")
self.label_26 = QtWidgets.QLabel(self.groupBox_4)
self.label_26.setGeometry(QtCore.QRect(160, 190, 61, 21))
self.label_26.setObjectName("label_26")
self.label_25 = QtWidgets.QLabel(self.groupBox_4)
self.label_25.setGeometry(QtCore.QRect(10, 190, 61, 21))
self.label_25.setObjectName("label_25")
self.epsReacAnal = QtWidgets.QLineEdit(self.centralwidget)
self.epsReacAnal.setGeometry(QtCore.QRect(1050, 90, 71, 22))
self.epsReacAnal.setObjectName("epsReacAnal")
self.label_28 = QtWidgets.QLabel(self.centralwidget)
self.label_28.setGeometry(QtCore.QRect(1130, 60, 61, 21))
self.label_28.setObjectName("label_28")
```

```
        self.label_29 = QtWidgets.QLabel(self.centralwidget)
        self.label_29.setGeometry(QtCore.QRect(980, 90, 61, 21))
        self.label_29.setObjectName("label_29")
        self.label_30 = QtWidgets.QLabel(self.centralwidget)
        self.label_30.setGeometry(QtCore.QRect(980, 60, 61, 21))
        self.label_30.setObjectName("label_30")
        self.epsProdAnal = QtWidgets.QLineEdit(self.centralwidget)
        self.epsProdAnal.setGeometry(QtCore.QRect(1050, 60, 71, 22))
        self.epsProdAnal.setObjectName("epsProdAnal")
        self.label_31 = QtWidgets.QLabel(self.centralwidget)
        self.label_31.setGeometry(QtCore.QRect(1130, 90, 61, 21))
        self.label_31.setObjectName("label_31")
        self.label_32 = QtWidgets.QLabel(self.centralwidget)
        self.label_32.setGeometry(QtCore.QRect(990, 30, 181, 21))
        self.label_32.setObjectName("label_32")
        self.paperCheck = QtWidgets.QCheckBox(self.centralwidget)
        self.paperCheck.setGeometry(QtCore.QRect(500, 330, 131, 31))
        self.paperCheck.setChecked(True)
        self.paperCheck.setObjectName("paperCheck")
        self.verticalLayoutWidget_3 = QtWidgets.QWidget(self.centralwidget)
        self.verticalLayoutWidget_3.setGeometry(QtCore.QRect(950, 390, 531, 361))
        self.verticalLayoutWidget_3.setObjectName("verticalLayoutWidget_3")
        self.resFigLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_3)
        self.resFigLayout.setSizeConstraint(QtWidgets.QLayout.SetNoConstraint)
        self.resFigLayout.setContentsMargins(0, 0, 0, 0)
        self.resFigLayout.setObjectName("resFigLayout")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1572, 22))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        self.tabWidget.setCurrentIndex(0)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.runCalcBtn.setText(_translate("MainWindow", "Run analysis"))
        self.getFileBtn.setText(_translate("MainWindow", "Load file..."))
        self.chosenLabel.setText(_translate("MainWindow", "File chosen: "))
        self.runSimBtn.setText(_translate("MainWindow", "Run simulation\n"
"(Requires all fields)"))
        self.simTimeLineEdit.setText(_translate("MainWindow", "100"))
        self.label_11.setText(_translate("MainWindow", "seconds"))
        self.label_14.setText(_translate("MainWindow", "Starting time"))
        self.label_15.setText(_translate("MainWindow", "End time -(1 for all)"))
        self.groupBox_3.setTitle(_translate("MainWindow", "Experimental data"))
        self.qyLineEdit.setToolTip(_translate("MainWindow", "<html><head/><body><p>e.g. 0.61</p></bo...
        self.qyLineEdit.setText(_translate("MainWindow", "0.61"))
        self.qyRadio.setText(_translate("MainWindow", "Quantum yield"))
        self.label_8.setText(_translate("MainWindow", "nm"))
        self.fluxRadio.setText(_translate("MainWindow", "Photon flux"))
        self.fluxLineEdit.setToolTip(_translate("MainWindow", "<html><head/><body><p>e.g. 1.24e14</p...
        self.fluxLineEdit.setWhatsThis(_translate("MainWindow", "<html><head/><body><p>1.24e14</p></...
        self.fluxLineEdit.setText(_translate("MainWindow", "1e15"))
        self.label_7.setText(_translate("MainWindow", "@"))
        self.fluxwlLineEdit.setText(_translate("MainWindow", "365"))
        self.loadLEDBtn.setText(_translate("MainWindow", "Load LED..."))
        self.label_17.setText(_translate("MainWindow", "cm"))
        self.label_19.setText(_translate("MainWindow", "Cuvette path length"))
        self.pathLength.setText(_translate("MainWindow", "1"))
        self.abswlLineEdit.setText(_translate("MainWindow", "365"))
```

```
        self.label_2.setText(_translate("MainWindow", "LED wavelength"))
        self.label_5.setText(_translate("MainWindow", "nm"))
        self.extincRadio.setText(_translate("MainWindow", "Extinction coefficient"))
        self.concRadio.setText(_translate("MainWindow", "Concentration (M)"))
        self.extincLineEdit.setText(_translate("MainWindow", "1284"))
        self.concLineEdit.setToolTip(_translate("MainWindow", "<html><head/><body><p>5.62e-4</p></bo...
        self.concLineEdit.setText(_translate("MainWindow", "9.6e-5"))
        self.label_4.setText(_translate("MainWindow", "@"))
        self.extincwlLineEdit.setText(_translate("MainWindow", "365"))
        self.label_9.setText(_translate("MainWindow", "@"))
        self.label_10.setText(_translate("MainWindow", "nm"))
        self.volLineEdit.setText(_translate("MainWindow", "80e-6"))
        self.label_3.setText(_translate("MainWindow", "Sample volume"))
        self.label_21.setText(_translate("MainWindow", "L"))
        self.groupBox.setTitle(_translate("MainWindow", "Hyperparameters"))
        self.label_13.setText(_translate("MainWindow", "nm"))
        self.zeroLineEdit.setText(_translate("MainWindow", "400"))
        self.label_12.setText(_translate("MainWindow", "Zero point wavelength\n"
"(for baseline correction)"))
        self.label_16.setText(_translate("MainWindow", "Number of points for fit"))
        self.analysiswlLineEdit.setText(_translate("MainWindow", "340"))
        self.label.setText(_translate("MainWindow", "Analysis wavelength (nm)"))
        self.manCorr.setText(_translate("MainWindow", "0"))
        self.label_18.setText(_translate("MainWindow", "Additional manual\n"
"correction"))
        self.saveParams.setText(_translate("MainWindow", "Save params..."))
        self.label_20.setText(_translate("MainWindow", "Log"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab), _translate("MainWindow", "Fit"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2), _translate("MainWindow", "Spec...
        self.btnExport.setText(_translate("MainWindow", "Export..."))
        self.groupBox_4.setTitle(_translate("MainWindow", "Corrections"))
        self.kCheck.setText(_translate("MainWindow", "Include back\n"
"reaction"))
        self.kLine.setText(_translate("MainWindow", "0.1"))
        self.label_6.setText(_translate("MainWindow", "k = "))
        self.label_22.setText(_translate("MainWindow", "1/s"))
        self.prodAbsCheck.setText(_translate("MainWindow", "Include product\n"
"absorption"))
        self.epsProdLine.setText(_translate("MainWindow", "300"))
        self.label_23.setText(_translate("MainWindow", "ε(prod) ="))
        self.label_24.setText(_translate("MainWindow", "L/(mol*cm)"))
        self.epsReacLine.setText(_translate("MainWindow", "22000"))
        self.label_27.setText(_translate("MainWindow", "At LED wavelength"))
        self.label_26.setText(_translate("MainWindow", "L/(mol*cm)"))
        self.label_25.setText(_translate("MainWindow", "ε(reac) ="))
        self.epsReacAnal.setText(_translate("MainWindow", "3581"))
        self.label_28.setText(_translate("MainWindow", "L/(mol*cm)"))
        self.label_29.setText(_translate("MainWindow", "ε(reac) ="))
        self.label_30.setText(_translate("MainWindow", "ε(prod) ="))
        self.epsProdAnal.setText(_translate("MainWindow", "74"))
        self.label_31.setText(_translate("MainWindow", "L/(mol*cm)"))
        self.label_32.setText(_translate("MainWindow", "Only at analysis wavelength"))
        self.paperCheck.setText(_translate("MainWindow", "Use paper-format\n"
"for graphs"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

FILE 11/11: lib/windows/QYEX_f2_win.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'QYEX_f2.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1768, 902)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout_9 = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout_9.setObjectName("gridLayout_9")
        self.logTextEdit = QtWidgets.QTextEdit(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.logTextEdit.sizePolicy().hasHeightForWidth())
        self.logTextEdit.setSizePolicy(sizePolicy)
        self.logTextEdit.setObjectName("logTextEdit")
        self.gridLayout_9.addWidget(self.logTextEdit, 9, 0, 1, 2)
        self.line = QtWidgets.QFrame(self.centralwidget)
        self.line.setFrameShape(QtWidgets.QFrame.VLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.gridLayout_9.addWidget(self.line, 0, 3, 10, 1)
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Ex...
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.groupBox.sizePolicy().hasHeightForWidth())
        self.groupBox.setSizePolicy(sizePolicy)
        self.groupBox.setObjectName("groupBox")
        self.gridLayout_3 = QtWidgets.QGridLayout(self.groupBox)
        self.gridLayout_3.setObjectName("gridLayout_3")
        self.timeLayout = QtWidgets.QGridLayout()
        self.timeLayout.setObjectName("timeLayout")
        self.gridLayout_3.addLayout(self.timeLayout, 0, 1, 1, 1)
        self.absLayout = QtWidgets.QGridLayout()
        self.absLayout.setObjectName("absLayout")
        self.gridLayout_3.addLayout(self.absLayout, 0, 0, 1, 1)
        self.gridLayout_9.addWidget(self.groupBox, 2, 4, 8, 5)
        self.tabWidget_5 = QtWidgets.QTabWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePo...
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.tabWidget_5.sizePolicy().hasHeightForWidth())
        self.tabWidget_5.setSizePolicy(sizePolicy)
        self.tabWidget_5.setObjectName("tabWidget_5")
        self.tabWidget_5Page1 = QtWidgets.QWidget()
        self.tabWidget_5Page1.setObjectName("tabWidget_5Page1")
        self.gridLayout_10 = QtWidgets.QGridLayout(self.tabWidget_5Page1)
        self.gridLayout_10.setObjectName("gridLayout_10")
        self.LEDlist = QtWidgets.QComboBox(self.tabWidget_5Page1)
        self.LEDlist.setObjectName("LEDlist")
```

```
self.LEDlist.addItem("")
self.gridLayout_10.addWidget(self.LEDlist, 2, 0, 1, 2)
self.setLED = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.setLED.setObjectName("setLED")
self.gridLayout_10.addWidget(self.setLED, 3, 0, 1, 2)
self.connectLED = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.connectLED.setEnabled(True)
self.connectLED.setObjectName("connectLED")
self.gridLayout_10.addWidget(self.connectLED, 1, 0, 1, 1)
self.channel2_text = QtWidgets.QLabel(self.tabWidget_5Page1)
self.channel2_text.setObjectName("channel2_text")
self.gridLayout_10.addWidget(self.channel2_text, 5, 6, 1, 1)
self.Shelly_off = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.Shelly_off.setEnabled(False)
self.Shelly_off.setObjectName("Shelly_off")
self.gridLayout_10.addWidget(self.Shelly_off, 6, 4, 1, 2)
self.Shelly_on = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.Shelly_on.setEnabled(False)
self.Shelly_on.setObjectName("Shelly_on")
self.gridLayout_10.addWidget(self.Shelly_on, 6, 0, 1, 2)
self.label_6 = QtWidgets.QLabel(self.tabWidget_5Page1)
self.label_6.setObjectName("label_6")
self.gridLayout_10.addWidget(self.label_6, 2, 5, 1, 1)
self.lampOffBtn = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.lampOffBtn.setObjectName("lampOffBtn")
self.gridLayout_10.addWidget(self.lampOffBtn, 5, 4, 1, 2)
self.LEDpower = QtWidgets.QSpinBox(self.tabWidget_5Page1)
self.LEDpower.setMaximum(2000)
self.LEDpower.setObjectName("LEDpower")
self.gridLayout_10.addWidget(self.LEDpower, 2, 4, 1, 1)
self.LEDoff = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.LEDoff.setObjectName("LEDoff")
self.gridLayout_10.addWidget(self.LEDoff, 3, 4, 1, 1)
self.lampOnBtn = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.lampOnBtn.setObjectName("lampOnBtn")
self.gridLayout_10.addWidget(self.lampOnBtn, 5, 0, 1, 2)
self.LEDcurrentpower = QtWidgets.QLabel(self.tabWidget_5Page1)
self.LEDcurrentpower.setObjectName("LEDcurrentpower")
self.gridLayout_10.addWidget(self.LEDcurrentpower, 2, 6, 1, 1)
self.disconnectLED = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.disconnectLED.setEnabled(False)
self.disconnectLED.setObjectName("disconnectLED")
self.gridLayout_10.addWidget(self.disconnectLED, 1, 1, 1, 4)
self.plugLabel = QtWidgets.QLabel(self.tabWidget_5Page1)
self.plugLabel.setObjectName("plugLabel")
self.gridLayout_10.addWidget(self.plugLabel, 6, 6, 1, 1)
self.LED_stat_text = QtWidgets.QLabel(self.tabWidget_5Page1)
self.LED_stat_text.setObjectName("LED_stat_text")
self.gridLayout_10.addWidget(self.LED_stat_text, 3, 6, 1, 1)
self.setPowerBtn = QtWidgets.QPushButton(self.tabWidget_5Page1)
self.setPowerBtn.setObjectName("setPowerBtn")
self.gridLayout_10.addWidget(self.setPowerBtn, 1, 6, 1, 1)
self.tabWidget_5.addTab(self.tabWidget_5Page1, "")
self.tab_2 = QtWidgets.QWidget()
self.tab_2.setObjectName("tab_2")
self.groupBox_8 = QtWidgets.QGroupBox(self.tab_2)
self.groupBox_8.setGeometry(QtCore.QRect(10, 10, 331, 171))
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Pr...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.groupBox_8.sizePolicy().hasHeightForWidth())
self.groupBox_8.setSizePolicy(sizePolicy)
self.groupBox_8.setObjectName("groupBox_8")
self.gridLayout_14 = QtWidgets.QGridLayout(self.groupBox_8)
self.gridLayout_14.setObjectName("gridLayout_14")
self.VersionBtn_vap = QtWidgets.QPushButton(self.groupBox_8)
self.VersionBtn_vap.setEnabled(False)
```

```
self.VersionBtn_vap.setObjectName("VersionBtn_vap")
self.gridLayout_14.addWidget(self.VersionBtn_vap, 0, 1, 1, 1)
spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePol...
self.gridLayout_14.addItem(spacerItem, 4, 0, 1, 2)
self.Shelly_IP = QtWidgets.QLineEdit(self.groupBox_8)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fi...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Shelly_IP.sizePolicy().hasHeightForWidth())
self.Shelly_IP.setSizePolicy(sizePolicy)
self.Shelly_IP.setObjectName("Shelly_IP")
self.gridLayout_14.addWidget(self.Shelly_IP, 2, 1, 1, 1)
self.label_27 = QtWidgets.QLabel(self.groupBox_8)
self.label_27.setObjectName("label_27")
self.gridLayout_14.addWidget(self.label_27, 2, 0, 1, 1)
self.disconnectLED_2 = QtWidgets.QPushButton(self.groupBox_8)
self.disconnectLED_2.setEnabled(False)
self.disconnectLED_2.setObjectName("disconnectLED_2")
self.gridLayout_14.addWidget(self.disconnectLED_2, 0, 0, 1, 1)
self.shellyConnect = QtWidgets.QPushButton(self.groupBox_8)
self.shellyConnect.setObjectName("shellyConnect")
self.gridLayout_14.addWidget(self.shellyConnect, 3, 0, 1, 1)
self.tabWidget_5.addTab(self.tab_2, "")
self.gridLayout_9.addWidget(self.tabWidget_5, 0, 4, 1, 2)
self.groupBox_7 = QtWidgets.QGroupBox(self.centralwidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePo...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.groupBox_7.sizePolicy().hasHeightForWidth())
self.groupBox_7.setSizePolicy(sizePolicy)
self.groupBox_7.setObjectName("groupBox_7")
self.gridLayout_12 = QtWidgets.QGridLayout(self.groupBox_7)
self.gridLayout_12.setObjectName("gridLayout_12")
self.label_7 = QtWidgets.QLabel(self.groupBox_7)
self.label_7.setObjectName("label_7")
self.gridLayout_12.addWidget(self.label_7, 1, 0, 1, 1)
self.label = QtWidgets.QLabel(self.groupBox_7)
self.label.setObjectName("label")
self.gridLayout_12.addWidget(self.label, 0, 0, 1, 1)
self.TBC_start = QtWidgets.QPushButton(self.groupBox_7)
self.TBC_start.setObjectName("TBC_start")
self.gridLayout_12.addWidget(self.TBC_start, 2, 0, 1, 3)
self.TBC_interval_box = QtWidgets.QSpinBox(self.groupBox_7)
self.TBC_interval_box.setMinimum(1)
self.TBC_interval_box.setMaximum(1000)
self.TBC_interval_box.setProperty("value", 20)
self.TBC_interval_box.setObjectName("TBC_interval_box")
self.gridLayout_12.addWidget(self.TBC_interval_box, 1, 1, 1, 2)
self.Temperature_box = QtWidgets.QSpinBox(self.groupBox_7)
self.Temperature_box.setProperty("value", 80)
self.Temperature_box.setObjectName("Temperature_box")
self.gridLayout_12.addWidget(self.Temperature_box, 0, 1, 1, 2)
self.label_9 = QtWidgets.QLabel(self.groupBox_7)
self.label_9.setObjectName("label_9")
self.gridLayout_12.addWidget(self.label_9, 3, 0, 1, 1)
self.gridLayout_13 = QtWidgets.QGridLayout()
self.gridLayout_13.setObjectName("gridLayout_13")
self.Pump_vol_Box = QtWidgets.QSpinBox(self.groupBox_7)
self.Pump_vol_Box.setMinimum(1)
self.Pump_vol_Box.setMaximum(500)
self.Pump_vol_Box.setProperty("value", 200)
self.Pump_vol_Box.setObjectName("Pump_vol_Box")
self.gridLayout_13.addWidget(self.Pump_vol_Box, 1, 0, 1, 1)
self.label_19 = QtWidgets.QLabel(self.groupBox_7)
self.label_19.setObjectName("label_19")
self.gridLayout_13.addWidget(self.label_19, 0, 0, 1, 1)
self.Pump_button = QtWidgets.QPushButton(self.groupBox_7)
```

```
self.Pump_button.setEnabled(False)
self.Pump_button.setObjectName("Pump_button")
self.gridLayout_13.addWidget(self.Pump_button, 2, 0, 1, 1)
self.gridLayout_12.addLayout(self.gridLayout_13, 3, 1, 1, 2)
self.gridLayout_9.addWidget(self.groupBox_7, 0, 8, 1, 1)
self.groupBox_3 = QtWidgets.QGroupBox(self.centralwidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fi...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.groupBox_3.sizePolicy().hasHeightForWidth())
self.groupBox_3.setSizePolicy(sizePolicy)
self.groupBox_3.setObjectName("groupBox_3")
self.gridLayout_7 = QtWidgets.QGridLayout(self.groupBox_3)
self.gridLayout_7.setObjectName("gridLayout_7")
self.gridLayout = QtWidgets.QGridLayout()
self.gridLayout.setObjectName("gridLayout")
self.label_14 = QtWidgets.QLabel(self.groupBox_3)
self.label_14.setObjectName("label_14")
self.gridLayout.addWidget(self.label_14, 0, 0, 1, 1)
self.Wavelength_line = QtWidgets.QLineEdit(self.groupBox_3)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePo...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Wavelength_line.sizePolicy().hasHeightForWidth())
self.Wavelength_line.setSizePolicy(sizePolicy)
self.Wavelength_line.setObjectName("Wavelength_line")
self.gridLayout.addWidget(self.Wavelength_line, 0, 1, 1, 1)
self.label_16 = QtWidgets.QLabel(self.groupBox_3)
self.label_16.setObjectName("label_16")
self.gridLayout.addWidget(self.label_16, 0, 2, 1, 1)
self.Background_line = QtWidgets.QLineEdit(self.groupBox_3)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePo...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Background_line.sizePolicy().hasHeightForWidth())
self.Background_line.setSizePolicy(sizePolicy)
self.Background_line.setMaximumSize(QtCore.QSize(133, 22))
self.Background_line.setObjectName("Background_line")
self.gridLayout.addWidget(self.Background_line, 0, 3, 1, 1)
self.Set_wavelength_btn = QtWidgets.QPushButton(self.groupBox_3)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Fixe...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Set_wavelength_btn.sizePolicy().hasHeightForWidth())
self.Set_wavelength_btn.setSizePolicy(sizePolicy)
self.Set_wavelength_btn.setObjectName("Set_wavelength_btn")
self.gridLayout.addWidget(self.Set_wavelength_btn, 0, 4, 1, 1)
self.gridLayout_7.addLayout(self.gridLayout, 0, 2, 1, 1)
self.gridLayout_2 = QtWidgets.QGridLayout()
self.gridLayout_2.setObjectName("gridLayout_2")
self.Min_wl_box = QtWidgets.QSpinBox(self.groupBox_3)
self.Min_wl_box.setMinimum(200)
self.Min_wl_box.setMaximum(1200)
self.Min_wl_box.setProperty("value", 250)
self.Min_wl_box.setObjectName("Min_wl_box")
self.gridLayout_2.addWidget(self.Min_wl_box, 0, 1, 1, 1)
self.label_17 = QtWidgets.QLabel(self.groupBox_3)
self.label_17.setObjectName("label_17")
self.gridLayout_2.addWidget(self.label_17, 0, 0, 1, 1)
self.Max_wl_box = QtWidgets.QSpinBox(self.groupBox_3)
self.Max_wl_box.setMinimum(200)
self.Max_wl_box.setMaximum(1200)
self.Max_wl_box.setProperty("value", 500)
self.Max_wl_box.setObjectName("Max_wl_box")
self.gridLayout_2.addWidget(self.Max_wl_box, 0, 3, 1, 1)
self.label_18 = QtWidgets.QLabel(self.groupBox_3)
self.label_18.setObjectName("label_18")
```

```
self.gridLayout_2.addWidget(self.label_18, 0, 2, 1, 1)
self.gridLayout_7.addLayout(self.gridLayout_2, 0, 0, 1, 1)
spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSize...
self.gridLayout_7.addItem(spacerItem1, 0, 1, 1, 1)
self.gridLayout_9.addWidget(self.groupBox_3, 1, 4, 1, 5)
self.groupBox_6 = QtWidgets.QGroupBox(self.centralwidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePo...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.groupBox_6.sizePolicy().hasHeightForWidth())
self.groupBox_6.setSizePolicy(sizePolicy)
self.groupBox_6.setObjectName("groupBox_6")
self.gridLayout_11 = QtWidgets.QGridLayout(self.groupBox_6)
self.gridLayout_11.setObjectName("gridLayout_11")
self.stopQYBtn = QtWidgets.QPushButton(self.groupBox_6)
self.stopQYBtn.setEnabled(True)
self.stopQYBtn.setObjectName("stopQYBtn")
self.gridLayout_11.addWidget(self.stopQYBtn, 4, 1, 1, 2)
self.Blank_meas_num = QtWidgets.QSpinBox(self.groupBox_6)
self.Blank_meas_num.setSingleStep(1)
self.Blank_meas_num.setProperty("value", 5)
self.Blank_meas_num.setObjectName("Blank_meas_num")
self.gridLayout_11.addWidget(self.Blank_meas_num, 0, 2, 1, 1)
self.lampToggleCheck = QtWidgets.QCheckBox(self.groupBox_6)
self.lampToggleCheck.setObjectName("lampToggleCheck")
self.gridLayout_11.addWidget(self.lampToggleCheck, 3, 0, 1, 2)
self.startQYBtn = QtWidgets.QPushButton(self.groupBox_6)
self.startQYBtn.setEnabled(True)
self.startQYBtn.setObjectName("startQYBtn")
self.gridLayout_11.addWidget(self.startQYBtn, 4, 0, 1, 1)
self.label_10 = QtWidgets.QLabel(self.groupBox_6)
self.label_10.setObjectName("label_10")
self.gridLayout_11.addWidget(self.label_10, 1, 0, 1, 2)
self.label_8 = QtWidgets.QLabel(self.groupBox_6)
self.label_8.setObjectName("label_8")
self.gridLayout_11.addWidget(self.label_8, 0, 0, 1, 1)
self.intervalLine = QtWidgets.QLineEdit(self.groupBox_6)
self.intervalLine.setObjectName("intervalLine")
self.gridLayout_11.addWidget(self.intervalLine, 1, 2, 1, 1)
self.qyRunning = QtWidgets.QLabel(self.groupBox_6)
self.qyRunning.setObjectName("qyRunning")
self.gridLayout_11.addWidget(self.qyRunning, 5, 0, 1, 3)
self.label_12 = QtWidgets.QLabel(self.groupBox_6)
self.label_12.setObjectName("label_12")
self.gridLayout_11.addWidget(self.label_12, 2, 0, 1, 1)
self.qyPointsSpinBox = QtWidgets.QSpinBox(self.groupBox_6)
self.qyPointsSpinBox.setMaximum(99999)
self.qyPointsSpinBox.setObjectName("qyPointsSpinBox")
self.gridLayout_11.addWidget(self.qyPointsSpinBox, 2, 2, 1, 1)
self.lampWarmSpin = QtWidgets.QSpinBox(self.groupBox_6)
self.lampWarmSpin.setObjectName("lampWarmSpin")
self.gridLayout_11.addWidget(self.lampWarmSpin, 3, 2, 1, 1)
self.gridLayout_9.addWidget(self.groupBox_6, 0, 6, 1, 1)
self.tabWidget_4 = QtWidgets.QTabWidget(self.centralwidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Pr...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.tabWidget_4.sizePolicy().hasHeightForWidth())
self.tabWidget_4.setSizePolicy(sizePolicy)
self.tabWidget_4.setObjectName("tabWidget_4")
self.UVVis = QtWidgets.QWidget()
self.UVVis.setObjectName("UVVis")
self.gridLayout_8 = QtWidgets.QGridLayout(self.UVVis)
self.gridLayout_8.setObjectName("gridLayout_8")
self.verticalLayout = QtWidgets.QVBoxLayout()
self.verticalLayout.setObjectName("verticalLayout")
self.OpenCommBtn = QtWidgets.QPushButton(self.UVVis)
```

```
self.OpenCommBtn.setObjectName("OpenCommBtn")
self.verticalLayout.addWidget(self.OpenCommBtn)
self.CloseCommBtn = QtWidgets.QPushButton(self.UVVis)
self.CloseCommBtn.setEnabled(False)
self.CloseCommBtn.setObjectName("CloseCommBtn")
self.verticalLayout.addWidget(self.CloseCommBtn)
self.getLightBtn = QtWidgets.QPushButton(self.UVVis)
self.getLightBtn.setEnabled(False)
self.getLightBtn.setObjectName("getLightBtn")
self.verticalLayout.addWidget(self.getLightBtn)
self.StartMeasBtn = QtWidgets.QPushButton(self.UVVis)
self.StartMeasBtn.setEnabled(False)
self.StartMeasBtn.setObjectName("StartMeasBtn")
self.verticalLayout.addWidget(self.StartMeasBtn)
self.StopMeasBtn = QtWidgets.QPushButton(self.UVVis)
self.StopMeasBtn.setEnabled(False)
self.StopMeasBtn.setObjectName("StopMeasBtn")
self.verticalLayout.addWidget(self.StopMeasBtn)
self.log_file_btn = QtWidgets.QPushButton(self.UVVis)
self.log_file_btn.setObjectName("log_file_btn")
self.verticalLayout.addWidget(self.log_file_btn)
self.log_name_label = QtWidgets.QLabel(self.UVVis)
self.log_name_label.setObjectName("log_name_label")
self.verticalLayout.addWidget(self.log_name_label)
self.gridLayout_8.addLayout(self.verticalLayout, 0, 0, 1, 1)
self.verticalGroupBox_2 = QtWidgets.QGroupBox(self.UVVis)
self.verticalGroupBox_2.setTitle("")
self.verticalGroupBox_2.setObjectName("verticalGroupBox_2")
self.gridLayout_6 = QtWidgets.QGridLayout(self.verticalGroupBox_2)
self.gridLayout_6.setObjectName("gridLayout_6")
self.lineEdit = QtWidgets.QLineEdit(self.verticalGroupBox_2)
self.lineEdit.setObjectName("lineEdit")
self.gridLayout_6.addWidget(self.lineEdit, 2, 1, 1, 1)
self.meas_num = QtWidgets.QLabel(self.verticalGroupBox_2)
self.meas_num.setObjectName("meas_num")
self.gridLayout_6.addWidget(self.meas_num, 8, 1, 1, 1)
self.label_15 = QtWidgets.QLabel(self.verticalGroupBox_2)
self.label_15.setObjectName("label_15")
self.gridLayout_6.addWidget(self.label_15, 2, 0, 1, 1)
self.label_2 = QtWidgets.QLabel(self.verticalGroupBox_2)
self.label_2.setObjectName("label_2")
self.gridLayout_6.addWidget(self.label_2, 0, 0, 1, 1)
self.IntTimeEdt = QtWidgets.QLineEdit(self.verticalGroupBox_2)
self.IntTimeEdt.setObjectName("IntTimeEdt")
self.gridLayout_6.addWidget(self.IntTimeEdt, 0, 1, 1, 1)
self.label_4 = QtWidgets.QLabel(self.verticalGroupBox_2)
self.label_4.setObjectName("label_4")
self.gridLayout_6.addWidget(self.label_4, 3, 0, 1, 1)
self.label_3 = QtWidgets.QLabel(self.verticalGroupBox_2)
self.label_3.setObjectName("label_3")
self.gridLayout_6.addWidget(self.label_3, 1, 0, 1, 1)
self.NumMeasEdt = QtWidgets.QLineEdit(self.verticalGroupBox_2)
self.NumMeasEdt.setInputMethodHints(QtCore.Qt.ImhDigitsOnly)
self.NumMeasEdt.setObjectName("NumMeasEdt")
self.gridLayout_6.addWidget(self.NumMeasEdt, 3, 1, 1, 1)
self.NumAvgEdt = QtWidgets.QLineEdit(self.verticalGroupBox_2)
self.NumAvgEdt.setInputMethodHints(QtCore.Qt.ImhDigitsOnly)
self.NumAvgEdt.setObjectName("NumAvgEdt")
self.gridLayout_6.addWidget(self.NumAvgEdt, 1, 1, 1, 1)
self.time_elapsed = QtWidgets.QLabel(self.verticalGroupBox_2)
self.time_elapsed.setObjectName("time_elapsed")
self.gridLayout_6.addWidget(self.time_elapsed, 4, 1, 1, 1)
self.verticalLayout_2 = QtWidgets.QVBoxLayout()
self.verticalLayout_2.setObjectName("verticalLayout_2")
self.label_13 = QtWidgets.QLabel(self.verticalGroupBox_2)
self.label_13.setObjectName("label_13")
self.verticalLayout_2.addWidget(self.label_13)
```

```
self.loadRefBtn = QtWidgets.QPushButton(self.verticalGroupBox_2)
self.loadRefBtn.setEnabled(False)
self.loadRefBtn.setObjectName("loadRefBtn")
self.verticalLayout_2.addWidget(self.loadRefBtn)
self.showRefBtn = QtWidgets.QPushButton(self.verticalGroupBox_2)
self.showRefBtn.setEnabled(False)
self.showRefBtn.setObjectName("showRefBtn")
self.verticalLayout_2.addWidget(self.showRefBtn)
self.saveRefBtn = QtWidgets.QPushButton(self.verticalGroupBox_2)
self.saveRefBtn.setEnabled(False)
self.saveRefBtn.setObjectName("saveRefBtn")
self.verticalLayout_2.addWidget(self.saveRefBtn)
self.gridLayout_6.addLayout(self.verticalLayout_2, 8, 0, 1, 1)
self.gridLayout_8.addWidget(self.verticalGroupBox_2, 0, 1, 1, 1)
self.groupBox_2 = QtWidgets.QGroupBox(self.UVVis)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Ex...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.groupBox_2.sizePolicy().hasHeightForWidth())
self.groupBox_2.setSizePolicy(sizePolicy)
self.groupBox_2.setObjectName("groupBox_2")
self.gridLayout_5 = QtWidgets.QGridLayout(self.groupBox_2)
self.gridLayout_5.setObjectName("gridLayout_5")
self.monitorLayout = QtWidgets.QGridLayout()
self.monitorLayout.setObjectName("monitorLayout")
self.gridLayout_5.addLayout(self.monitorLayout, 0, 0, 1, 1)
self.gridLayout_8.addWidget(self.groupBox_2, 1, 0, 1, 2)
self.tabWidget_4.addTab(self.UVVis, "")
self.tab = QtWidgets.QWidget()
self.tab.setObjectName("tab")
self.verticalLayoutWidget = QtWidgets.QWidget(self.tab)
self.verticalLayoutWidget.setGeometry(QtCore.QRect(10, 10, 111, 71))
self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
self.verticalLayout_3.setContentsMargins(0, 0, 0, 0)
self.verticalLayout_3.setObjectName("verticalLayout_3")
self.qpodConnect = QtWidgets.QPushButton(self.verticalLayoutWidget)
self.qpodConnect.setObjectName("qpodConnect")
self.verticalLayout_3.addWidget(self.qpodConnect)
self.qpodDisconnect = QtWidgets.QPushButton(self.verticalLayoutWidget)
self.qpodDisconnect.setEnabled(False)
self.qpodDisconnect.setObjectName("qpodDisconnect")
self.verticalLayout_3.addWidget(self.qpodDisconnect)
self.groupBox_4 = QtWidgets.QGroupBox(self.tab)
self.groupBox_4.setGeometry(QtCore.QRect(140, 10, 191, 161))
self.groupBox_4.setObjectName("groupBox_4")
self.tempOn = QtWidgets.QPushButton(self.groupBox_4)
self.tempOn.setEnabled(False)
self.tempOn.setGeometry(QtCore.QRect(10, 70, 75, 24))
self.tempOn.setObjectName("tempOn")
self.tempOff = QtWidgets.QPushButton(self.groupBox_4)
self.tempOff.setEnabled(False)
self.tempOff.setGeometry(QtCore.QRect(100, 70, 75, 24))
self.tempOff.setObjectName("tempOff")
self.tempCheck = QtWidgets.QCheckBox(self.groupBox_4)
self.tempCheck.setEnabled(False)
self.tempCheck.setGeometry(QtCore.QRect(10, 130, 161, 20))
self.tempCheck.setObjectName("tempCheck")
self.tempSet = QtWidgets.QPushButton(self.groupBox_4)
self.tempSet.setGeometry(QtCore.QRect(100, 30, 75, 24))
self.tempSet.setObjectName("tempSet")
self.temp_label = QtWidgets.QLabel(self.groupBox_4)
self.temp_label.setGeometry(QtCore.QRect(10, 30, 81, 21))
self.temp_label.setObjectName("temp_label")
self.qpodStatus = QtWidgets.QLabel(self.groupBox_4)
self.qpodStatus.setGeometry(QtCore.QRect(10, 100, 141, 16))
self.qpodStatus.setObjectName("qpodStatus")
```

```
self.tabWidget_4.addTab(self.tab, "")
self.tab_4 = QtWidgets.QWidget()
self.tab_4.setObjectName("tab_4")
self.layoutWidget = QtWidgets.QWidget(self.tab_4)
self.layoutWidget.setGeometry(QtCore.QRect(10, 45, 431, 466))
self.layoutWidget.setObjectName("layoutWidget")
self.gridLayout_26 = QtWidgets.QGridLayout(self.layoutWidget)
self.gridLayout_26.setContentsMargins(0, 0, 0, 0)
self.gridLayout_26.setObjectName("gridLayout_26")
self.Sample_name_6 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_6.setObjectName("Sample_name_6")
self.gridLayout_26.addWidget(self.Sample_name_6, 7, 1, 1, 1)
self.Valve_50 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_50.sizePolicy().hasHeightForWidth())
self.Valve_50.setSizePolicy(sizePolicy)
self.Valve_50.setObjectName("Valve_50")
self.gridLayout_26.addWidget(self.Valve_50, 16, 0, 1, 1)
self.Valve_61 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_61.sizePolicy().hasHeightForWidth())
self.Valve_61.setSizePolicy(sizePolicy)
self.Valve_61.setObjectName("Valve_61")
self.gridLayout_26.addWidget(self.Valve_61, 12, 0, 1, 1)
self.Sample_name_11 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_11.setObjectName("Sample_name_11")
self.gridLayout_26.addWidget(self.Sample_name_11, 12, 1, 1, 1)
self.Valve_63 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_63.sizePolicy().hasHeightForWidth())
self.Valve_63.setSizePolicy(sizePolicy)
self.Valve_63.setObjectName("Valve_63")
self.gridLayout_26.addWidget(self.Valve_63, 11, 0, 1, 1)
self.Sample_name_10 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_10.setObjectName("Sample_name_10")
self.gridLayout_26.addWidget(self.Sample_name_10, 11, 1, 1, 1)
self.Valve_56 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_56.sizePolicy().hasHeightForWidth())
self.Valve_56.setSizePolicy(sizePolicy)
self.Valve_56.setObjectName("Valve_56")
self.gridLayout_26.addWidget(self.Valve_56, 6, 0, 1, 1)
self.Valve_53 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_53.sizePolicy().hasHeightForWidth())
self.Valve_53.setSizePolicy(sizePolicy)
self.Valve_53.setObjectName("Valve_53")
self.gridLayout_26.addWidget(self.Valve_53, 17, 0, 1, 1)
self.Sample_name_5 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_5.setObjectName("Sample_name_5")
self.gridLayout_26.addWidget(self.Sample_name_5, 6, 1, 1, 1)
self.Sample_name_2 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_2.setObjectName("Sample_name_2")
self.gridLayout_26.addWidget(self.Sample_name_2, 3, 1, 1, 1)
self.Sample_name_1 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_1.setObjectName("Sample_name_1")
self.gridLayout_26.addWidget(self.Sample_name_1, 1, 1, 2, 1)
```

```
self.Sample_name_7 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_7.setObjectName("Sample_name_7")
self.gridLayout_26.addWidget(self.Sample_name_7, 8, 1, 1, 1)
self.Sample_name_9 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_9.setObjectName("Sample_name_9")
self.gridLayout_26.addWidget(self.Sample_name_9, 10, 1, 1, 1)
self.Valve_62 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_62.sizePolicy().hasHeightForWidth())
self.Valve_62.setSizePolicy(sizePolicy)
self.Valve_62.setObjectName("Valve_62")
self.gridLayout_26.addWidget(self.Valve_62, 3, 0, 1, 1)
self.Valve_59 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_59.sizePolicy().hasHeightForWidth())
self.Valve_59.setSizePolicy(sizePolicy)
self.Valve_59.setObjectName("Valve_59")
self.gridLayout_26.addWidget(self.Valve_59, 14, 0, 1, 1)
self.label_54 = QtWidgets.QLabel(self.layoutWidget)
self.label_54.setObjectName("label_54")
self.gridLayout_26.addWidget(self.label_54, 0, 0, 1, 1)
self.Valve_52 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_52.sizePolicy().hasHeightForWidth())
self.Valve_52.setSizePolicy(sizePolicy)
self.Valve_52.setObjectName("Valve_52")
self.gridLayout_26.addWidget(self.Valve_52, 9, 0, 1, 1)
self.Valve_57 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_57.sizePolicy().hasHeightForWidth())
self.Valve_57.setSizePolicy(sizePolicy)
self.Valve_57.setObjectName("Valve_57")
self.gridLayout_26.addWidget(self.Valve_57, 4, 0, 1, 1)
self.Sample_name_4 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_4.setObjectName("Sample_name_4")
self.gridLayout_26.addWidget(self.Sample_name_4, 5, 1, 1, 1)
self.Sample_name_12 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_12.setObjectName("Sample_name_12")
self.gridLayout_26.addWidget(self.Sample_name_12, 13, 1, 1, 1)
self.Sample_name_16 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_16.setObjectName("Sample_name_16")
self.gridLayout_26.addWidget(self.Sample_name_16, 17, 1, 1, 1)
self.Valve_49 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_49.sizePolicy().hasHeightForWidth())
self.Valve_49.setSizePolicy(sizePolicy)
self.Valve_49.setObjectName("Valve_49")
self.gridLayout_26.addWidget(self.Valve_49, 5, 0, 1, 1)
self.Sample_name_8 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_8.setObjectName("Sample_name_8")
self.gridLayout_26.addWidget(self.Sample_name_8, 9, 1, 1, 1)
self.Sample_name_14 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_14.setObjectName("Sample_name_14")
self.gridLayout_26.addWidget(self.Sample_name_14, 15, 1, 1, 1)
self.Valve_51 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
```

```
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_51.sizePolicy().hasHeightForWidth())
self.Valve_51.setSizePolicy(sizePolicy)
self.Valve_51.setObjectName("Valve_51")
self.gridLayout_26.addWidget(self.Valve_51, 13, 0, 1, 1)
self.Sample_name_3 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_3.setObjectName("Sample_name_3")
self.gridLayout_26.addWidget(self.Sample_name_3, 4, 1, 1, 1)
self.label_37 = QtWidgets.QLabel(self.layoutWidget)
self.label_37.setObjectName("label_37")
self.gridLayout_26.addWidget(self.label_37, 0, 1, 1, 1)
self.Valve_60 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_60.sizePolicy().hasHeightForWidth())
self.Valve_60.setSizePolicy(sizePolicy)
self.Valve_60.setObjectName("Valve_60")
self.gridLayout_26.addWidget(self.Valve_60, 7, 0, 1, 1)
self.Sample_name_15 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_15.setObjectName("Sample_name_15")
self.gridLayout_26.addWidget(self.Sample_name_15, 16, 1, 1, 1)
self.Valve_64 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_64.sizePolicy().hasHeightForWidth())
self.Valve_64.setSizePolicy(sizePolicy)
self.Valve_64.setObjectName("Valve_64")
self.gridLayout_26.addWidget(self.Valve_64, 8, 0, 1, 1)
self.Sample_name_13 = QtWidgets.QLineEdit(self.layoutWidget)
self.Sample_name_13.setObjectName("Sample_name_13")
self.gridLayout_26.addWidget(self.Sample_name_13, 14, 1, 1, 1)
self.Valve_54 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_54.sizePolicy().hasHeightForWidth())
self.Valve_54.setSizePolicy(sizePolicy)
self.Valve_54.setObjectName("Valve_54")
self.gridLayout_26.addWidget(self.Valve_54, 15, 0, 1, 1)
self.Valve_55 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_55.sizePolicy().hasHeightForWidth())
self.Valve_55.setSizePolicy(sizePolicy)
self.Valve_55.setObjectName("Valve_55")
self.gridLayout_26.addWidget(self.Valve_55, 10, 0, 1, 1)
self.Valve_58 = QtWidgets.QLabel(self.layoutWidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Pref...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Valve_58.sizePolicy().hasHeightForWidth())
self.Valve_58.setSizePolicy(sizePolicy)
self.Valve_58.setObjectName("Valve_58")
self.gridLayout_26.addWidget(self.Valve_58, 1, 0, 2, 1)
self.Save_Valve_Presets_Button = QtWidgets.QPushButton(self.tab_4)
self.Save_Valve_Presets_Button.setGeometry(QtCore.QRect(270, 10, 75, 24))
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Fixe...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Save_Valve_Presets_Button.sizePolicy().hasHeightForWidth()...
self.Save_Valve_Presets_Button.setSizePolicy(sizePolicy)
self.Save_Valve_Presets_Button.setObjectName("Save_Valve_Presets_Button")
self.Clear_Valve_Presets_Button = QtWidgets.QPushButton(self.tab_4)
self.Clear_Valve_Presets_Button.setGeometry(QtCore.QRect(360, 10, 81, 24))
```

```
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Fixe...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Clear_Valve_Presets_Button.sizePolicy().hasHeightForWidth(...
self.Clear_Valve_Presets_Button.setSizePolicy(sizePolicy)
self.Clear_Valve_Presets_Button.setObjectName("Clear_Valve_Presets_Button")
self.layoutWidget_2 = QtWidgets.QWidget(self.tab_4)
self.layoutWidget_2.setGeometry(QtCore.QRect(0, 520, 441, 91))
self.layoutWidget_2.setObjectName("layoutWidget_2")
self.gridLayout_27 = QtWidgets.QGridLayout(self.layoutWidget_2)
self.gridLayout_27.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
self.gridLayout_27.setContentsMargins(0, 0, 0, 0)
self.gridLayout_27.setObjectName("gridLayout_27")
self.label_58 = QtWidgets.QLabel(self.layoutWidget_2)
self.label_58.setObjectName("label_58")
self.gridLayout_27.addWidget(self.label_58, 1, 1, 1, 1)
self.label_57 = QtWidgets.QLabel(self.layoutWidget_2)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Prefer...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.label_57.sizePolicy().hasHeightForWidth())
self.label_57.setSizePolicy(sizePolicy)
self.label_57.setObjectName("label_57")
self.gridLayout_27.addWidget(self.label_57, 4, 8, 1, 1)
self.label_55 = QtWidgets.QLabel(self.layoutWidget_2)
self.label_55.setObjectName("label_55")
self.gridLayout_27.addWidget(self.label_55, 1, 4, 1, 1)
self.label_56 = QtWidgets.QLabel(self.layoutWidget_2)
self.label_56.setObjectName("label_56")
self.gridLayout_27.addWidget(self.label_56, 4, 5, 1, 1)
self.label_59 = QtWidgets.QLabel(self.layoutWidget_2)
self.label_59.setObjectName("label_59")
self.gridLayout_27.addWidget(self.label_59, 3, 5, 1, 1)
self.Pump_flowrate_4 = QtWidgets.QDoubleSpinBox(self.layoutWidget_2)
self.Pump_flowrate_4.setDecimals(1)
self.Pump_flowrate_4.setMaximum(3.0)
self.Pump_flowrate_4.setProperty("value", 1.0)
self.Pump_flowrate_4.setObjectName("Pump_flowrate_4")
self.gridLayout_27.addWidget(self.Pump_flowrate_4, 3, 6, 1, 1)
self.ValveList_4 = QtWidgets.QComboBox(self.layoutWidget_2)
self.ValveList_4.setEnabled(False)
self.ValveList_4.setObjectName("ValveList_4")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.ValveList_4.addItem("")
self.gridLayout_27.addWidget(self.ValveList_4, 3, 1, 1, 1)
self.label_60 = QtWidgets.QLabel(self.layoutWidget_2)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Prefer...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.label_60.sizePolicy().hasHeightForWidth())
self.label_60.setSizePolicy(sizePolicy)
self.label_60.setObjectName("label_60")
self.gridLayout_27.addWidget(self.label_60, 3, 8, 1, 1)
```

```
self.Valve_Select_Button_4 = QtWidgets.QPushButton(self.layoutWidget_2)
self.Valve_Select_Button_4.setEnabled(False)
self.Valve_Select_Button_4.setObjectName("Valve_Select_Button_4")
self.gridLayout_27.addWidget(self.Valve_Select_Button_4, 4, 1, 1, 1)
self.Load_Volume_4 = QtWidgets.QDoubleSpinBox(self.layoutWidget_2)
self.Load_Volume_4.setDecimals(1)
self.Load_Volume_4.setMaximum(10.0)
self.Load_Volume_4.setProperty("value", 1.2)
self.Load_Volume_4.setObjectName("Load_Volume_4")
self.gridLayout_27.addWidget(self.Load_Volume_4, 4, 6, 1, 1)
self.Load_Button_4 = QtWidgets.QPushButton(self.layoutWidget_2)
self.Load_Button_4.setEnabled(False)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixe...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Load_Button_4.sizePolicy().hasHeightForWidth())
self.Load_Button_4.setSizePolicy(sizePolicy)
self.Load_Button_4.setObjectName("Load_Button_4")
self.gridLayout_27.addWidget(self.Load_Button_4, 4, 4, 1, 1)
self.pumpConnect = QtWidgets.QPushButton(self.layoutWidget_2)
self.pumpConnect.setObjectName("pumpConnect")
self.gridLayout_27.addWidget(self.pumpConnect, 3, 4, 1, 1)
self.Clean_Button = QtWidgets.QPushButton(self.layoutWidget_2)
self.Clean_Button.setEnabled(False)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Fixe...
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.Clean_Button.sizePolicy().hasHeightForWidth())
self.Clean_Button.setSizePolicy(sizePolicy)
self.Clean_Button.setObjectName("Clean_Button")
self.gridLayout_27.addWidget(self.Clean_Button, 1, 5, 1, 1)
self.Clean_Volume = QtWidgets.QDoubleSpinBox(self.layoutWidget_2)
self.Clean_Volume.setDecimals(1)
self.Clean_Volume.setMinimum(1.0)
self.Clean_Volume.setMaximum(5.0)
self.Clean_Volume.setSingleStep(0.1)
self.Clean_Volume.setObjectName("Clean_Volume")
self.gridLayout_27.addWidget(self.Clean_Volume, 1, 6, 1, 1)
self.label_5 = QtWidgets.QLabel(self.layoutWidget_2)
self.label_5.setObjectName("label_5")
self.gridLayout_27.addWidget(self.label_5, 1, 8, 1, 1)
self.samplerConnect = QtWidgets.QPushButton(self.tab_4)
self.samplerConnect.setGeometry(QtCore.QRect(10, 10, 111, 24))
self.samplerConnect.setObjectName("samplerConnect")
self.tabWidget_4.addTab(self.tab_4, "")
self.gridLayout_9.addWidget(self.tabWidget_4, 0, 0, 9, 3)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 1768, 22))
self.menubar.setObjectName("menubar")
self.menuCalc_setup = QtWidgets.QMenu(self.menubar)
self.menuCalc_setup.setObjectName("menuCalc_setup")
self.menuHelp = QtWidgets.QMenu(self.menubar)
self.menuHelp.setObjectName("menuHelp")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)
self.actionHelpOxy = QtWidgets.QAction(MainWindow)
self.actionHelpOxy.setCheckable(False)
self.actionHelpOxy.setObjectName("actionHelpOxy")
self.menuHelp.addAction(self.actionHelpOxy)
self.menubar.addAction(self.menuCalc_setup.menuAction())
self.menubar.addAction(self.menuHelp.menuAction())

self.retranslateUi(MainWindow)
self.tabWidget_5.setCurrentIndex(0)
```

```
        self.tabWidget_4.setCurrentIndex(0)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.groupBox.setTitle(_translate("MainWindow", "Absorption and dynamics"))
        self.LEDlist.setItemText(0, _translate("MainWindow", "Choose LED..."))
        self.setLED.setText(_translate("MainWindow", "LED On"))
        self.connectLED.setText(_translate("MainWindow", "Connect LEDs"))
        self.channel2_text.setText(_translate("MainWindow", "Shutter status: N/A"))
        self.Shelly_off.setText(_translate("MainWindow", "Shelly off"))
        self.Shelly_on.setText(_translate("MainWindow", "Shelly on"))
        self.label_6.setText(_translate("MainWindow", "mA"))
        self.lampOffBtn.setText(_translate("MainWindow", "Close lamp shutter"))
        self.LEDoff.setText(_translate("MainWindow", "LED Off"))
        self.lampOnBtn.setText(_translate("MainWindow", "Open lamp shutter"))
        self.LEDcurrentpower.setText(_translate("MainWindow", "Current power: 0"))
        self.disconnectLED.setText(_translate("MainWindow", "Disconnect LEDs"))
        self.plugLabel.setText(_translate("MainWindow", "Plug status: N/A"))
        self.LED_stat_text.setText(_translate("MainWindow", "LED status: N/A"))
        self.setPowerBtn.setText(_translate("MainWindow", "Set power"))
        self.tabWidget_5.setTabText(self.tabWidget_5.indexOf(self.tabWidget_5Page1), _translate("Mai...
        self.groupBox_8.setTitle(_translate("MainWindow", "LED and Lamp settings"))
        self.VersionBtn_vap.setText(_translate("MainWindow", "Show Version info"))
        self.Shelly_IP.setText(_translate("MainWindow", "192.168.137.96"))
        self.label_27.setText(_translate("MainWindow", "Shelly plug IP address:"))
        self.disconnectLED_2.setText(_translate("MainWindow", "Disconnect LEDs"))
        self.shellyConnect.setText(_translate("MainWindow", "Connect Shelly"))
        self.tabWidget_5.setTabText(self.tabWidget_5.indexOf(self.tab_2), _translate("MainWindow", "...
        self.groupBox_7.setTitle(_translate("MainWindow", "Thermal Back-conversion"))
        self.label_7.setText(_translate("MainWindow", "TBC Interval, s"))
        self.label.setText(_translate("MainWindow", "Temperature C"))
        self.TBC_start.setText(_translate("MainWindow", "Start TBC experiment"))
        self.label_9.setText(_translate("MainWindow", "*TBC only works\n"
" after QY experiment\n"
" is initiated.To stop\n"
"press Stop QY "))
        self.label_19.setText(_translate("MainWindow", "Pump volume in ul:"))
        self.Pump_button.setText(_translate("MainWindow", "Pump"))
        self.groupBox_3.setTitle(_translate("MainWindow", "Absorption spectra and absorption dynamic...
        self.label_14.setText(_translate("MainWindow", "Wavelengths (nm) "))
        self.Wavelength_line.setText(_translate("MainWindow", "300, 400"))
        self.label_16.setText(_translate("MainWindow", "Background (nm)"))
        self.Background_line.setText(_translate("MainWindow", "1000"))
        self.Set_wavelength_btn.setText(_translate("MainWindow", "Set"))
        self.label_17.setText(_translate("MainWindow", "Min (nm)"))
        self.label_18.setText(_translate("MainWindow", "Max (nm)"))
        self.groupBox_6.setTitle(_translate("MainWindow", "Photoisomerisation measurement"))
        self.stopQYBtn.setText(_translate("MainWindow", "Stop QY experiment"))
        self.lampToggleCheck.setText(_translate("MainWindow", "Turn off lamp between \n"
"measurements"))
        self.startQYBtn.setText(_translate("MainWindow", "Start QY experiment"))
        self.label_10.setText(_translate("MainWindow", "Interval (seconds between\n"
" each absorption spectrum)"))
        self.label_8.setText(_translate("MainWindow", "Blank measurements #"))
        self.intervalLine.setText(_translate("MainWindow", "5"))
        self.qyRunning.setText(_translate("MainWindow", "Experiment status: not running."))
        self.label_12.setText(_translate("MainWindow", "Number of points\n"
"(0 for inf)"))
        self.OpenCommBtn.setText(_translate("MainWindow", "Connect AvaSpec"))
        self.CloseCommBtn.setText(_translate("MainWindow", "Disconnect AvaSpec"))
        self.getLightBtn.setText(_translate("MainWindow", "Get Light spectrum"))
        self.StartMeasBtn.setText(_translate("MainWindow", "Start measurement"))
        self.StopMeasBtn.setText(_translate("MainWindow", "Stop measurement"))
        self.log_file_btn.setText(_translate("MainWindow", "Set Log file destination"))
        self.log_name_label.setText(_translate("MainWindow", "Saving to: Not set"))
```

```
self.meas_num.setText(_translate("MainWindow", "Measurement: 0"))
self.label_15.setText(_translate("MainWindow", "Interval (s)"))
self.label_2.setText(_translate("MainWindow", "Exposure [ms]"))
self.IntTimeEdt.setText(_translate("MainWindow", "2"))
self.label_4.setText(_translate("MainWindow", "Number of measurements"))
self.label_3.setText(_translate("MainWindow", "Number of averages"))
self.NumMeasEdt.setText(_translate("MainWindow", "1"))
self.NumAvgEdt.setText(_translate("MainWindow", "100"))
self.time_elapsed.setText(_translate("MainWindow", "Time elapsed: 0"))
self.label_13.setText(_translate("MainWindow", "Ref spectrum:"))
self.loadRefBtn.setText(_translate("MainWindow", "Load..."))
self.showRefBtn.setText(_translate("MainWindow", "Show"))
self.saveRefBtn.setText(_translate("MainWindow", "Save..."))
self.groupBox_2.setTitle(_translate("MainWindow", "Raw Spectrum"))
self.tabWidget_4.setTabText(self.tabWidget_4.indexOf(self.UVVis), _translate("MainWindow", "...
self.qpodConnect.setText(_translate("MainWindow", "Connect"))
self.qpodDisconnect.setText(_translate("MainWindow", "Disconnect"))
self.groupBox_4.setTitle(_translate("MainWindow", "Tempature seeking"))
self.tempOn.setText(_translate("MainWindow", "Enable"))
self.tempOff.setText(_translate("MainWindow", "Disable"))
self.tempCheck.setText(_translate("MainWindow", "Current temp:"))
self.tempSet.setText(_translate("MainWindow", "Set"))
self.temp_label.setText(_translate("MainWindow", "T (C°):"))
self.qpodStatus.setText(_translate("MainWindow", "Status: ?"))
self.tabWidget_4.setTabText(self.tabWidget_4.indexOf(self.tab), _translate("MainWindow", "Te...
self.Valve_50.setText(_translate("MainWindow", "#15"))
self.Valve_61.setText(_translate("MainWindow", "#11"))
self.Valve_63.setText(_translate("MainWindow", "#10"))
self.Valve_56.setText(_translate("MainWindow", "#5"))
self.Valve_53.setText(_translate("MainWindow", "#16"))
self.Valve_62.setText(_translate("MainWindow", "#2"))
self.Valve_59.setText(_translate("MainWindow", "#13"))
self.label_54.setText(_translate("MainWindow", "Valve"))
self.Valve_52.setText(_translate("MainWindow", "#8"))
self.Valve_57.setText(_translate("MainWindow", "#3"))
self.Sample_name_16.setText(_translate("MainWindow", "Cleaning solvent"))
self.Valve_49.setText(_translate("MainWindow", "#4"))
self.Valve_51.setText(_translate("MainWindow", "#12"))
self.label_37.setText(_translate("MainWindow", "Sample name"))
self.Valve_60.setText(_translate("MainWindow", "#6"))
self.Sample_name_15.setText(_translate("MainWindow", "Reference solvent"))
self.Valve_64.setText(_translate("MainWindow", "#7"))
self.Valve_54.setText(_translate("MainWindow", "#14"))
self.Valve_55.setText(_translate("MainWindow", "#9"))
self.Valve_58.setText(_translate("MainWindow", "#1"))
self.Save_Valve_Presets_Button.setText(_translate("MainWindow", "Save presets"))
self.Clear_Valve_Presets_Button.setText(_translate("MainWindow", "Clear presets"))
self.label_58.setText(_translate("MainWindow", "Select valve:"))
self.label_57.setText(_translate("MainWindow", "mL"))
self.label_55.setText(_translate("MainWindow", "SF10 Pump controls:"))
self.label_56.setText(_translate("MainWindow", "Volume:"))
self.label_59.setText(_translate("MainWindow", "Flow rate:"))
self.ValveList_4.setItemText(0, _translate("MainWindow", "1"))
self.ValveList_4.setItemText(1, _translate("MainWindow", "2"))
self.ValveList_4.setItemText(2, _translate("MainWindow", "3"))
self.ValveList_4.setItemText(3, _translate("MainWindow", "4"))
self.ValveList_4.setItemText(4, _translate("MainWindow", "5"))
self.ValveList_4.setItemText(5, _translate("MainWindow", "6"))
self.ValveList_4.setItemText(6, _translate("MainWindow", "7"))
self.ValveList_4.setItemText(7, _translate("MainWindow", "8"))
self.ValveList_4.setItemText(8, _translate("MainWindow", "9"))
self.ValveList_4.setItemText(9, _translate("MainWindow", "10"))
self.ValveList_4.setItemText(10, _translate("MainWindow", "11"))
self.ValveList_4.setItemText(11, _translate("MainWindow", "12"))
self.ValveList_4.setItemText(12, _translate("MainWindow", "13"))
self.ValveList_4.setItemText(13, _translate("MainWindow", "14"))
self.ValveList_4.setItemText(14, _translate("MainWindow", "15"))
```

```
        self.ValveList_4.setItemText(15, _translate("MainWindow", "16"))
        self.label_60.setText(_translate("MainWindow", "mL/min"))
        self.Valve_Select_Button_4.setText(_translate("MainWindow", "Select valve"))
        self.Load_Button_4.setText(_translate("MainWindow", "Load single"))
        self.pumpConnect.setText(_translate("MainWindow", "Connect"))
        self.Clean_Button.setText(_translate("MainWindow", "Clean"))
        self.label_5.setText(_translate("MainWindow", "mL"))
        self.samplerConnect.setText(_translate("MainWindow", "Connect Sampler"))
        self.tabWidget_4.setTabText(self.tabWidget_4.indexOf(self.tab_4), _translate("MainWindow", "...
        self.menuCalc_setup.setTitle(_translate("MainWindow", "Calc setup"))
        self.menuHelp.setTitle(_translate("MainWindow", "Help"))
        self.actionHelpOxy.setText(_translate("MainWindow", "How to run oxygen levels"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```