Main Camera and pump-control script:

```
'''
Copyright (c) 2022 Matthew O'Brien

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
'''
#-------------------------------------------------------

#this script may need altering to work with the particular system
#(e.g. if a different pump is used etc.)
#it might have inefficiencies.
#there may be unused variables, imports and functions.
import sys
import time
import os.path
import cv2
import numpy as np
import pyqtgraph as pg
pg.setConfigOptions(antialias = True)
from PyQt5 import QtCore, QtGui, QtWidgets, Qt
from PyQt5.QtGui import QImage, QPixmap, QPainter, QPen
from PyQt5.QtCore import pyqtSlot, pyqtSignal, QObject, QThread
from PyQt5.QtCore import Qt, QTimer
from mainwindow import Ui_MainWindow
import subprocess


#the following properties will depend on the specifics of the camera
#the ability to set them will depend on the operating system etc.
#this is not needed if some other method of control is used
#(e.g. using some dedicated control software etc).
cam_props = {'zoom_absolute': 10,
             'exposure_auto': 1,
             'exposure_absolute': 5,
             'contrast': 5,
             'saturation': 80,
             'white_balance_temperature_auto': 0,
             'white_balance_temperature': 3500,
             'focus_auto': 0, 'focus_absolute': 0,
             'brightness': 255,
             'power_line_frequency': 0
             }

# the following should work on linux-type operating systems
# if v4l2-ctl is installed/available
# it is not required if another method of setting the camera settings is used
```

```
# or if automatic adjustment of settings is used.
'''
for key in cam_props:
    p = subprocess.Popen(['v4l2-ctl -d /dev/video0 -c {}={}'.format(key,
str(cam_props[key]))],
                         shell=True)
    time.sleep(0.1)
    p.kill()
'''

import serial
camnum = 1
#the following will depend on operating system
#/dev/ttyUSBX is typical on Linux-type systems.
serial_device = "/dev/ttyUSB0"
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))


flowrates = [0.1, 0.15, 0.2, 0.3, 0.5, 0.8, 1.2, 1.8, 2.8, 4.0, 5.0]

#whatever the filename will be to save the data:
filesave = 'xxx.npy'

resvol = 0.5

pause = 0
shutdown = 0

currentpixdata = 0
starttime = time.time()

class cam(QThread):

    vid = None
    sendframe = pyqtSignal(QImage)

    getdye = 0
    opening = 0
    closing = 0
    dyepix = pyqtSignal(int)

    def run(self):
        global pause
        while True:

            if shutdown:
                print ('shutdown')
                self.vid.release()

                break

            if pause != 1:
                ret, nextframe = self.vid.read()

            if ret:
                if self.getdye:
                    hsv = cv2.cvtColor(nextframe, cv2.COLOR_BGR2HSV)

                    lower = np.array([70, 30, 30])
                    higher = np.array([100, 255, 255])

                    #lower2 = np.array([0, 50, 50])
                    #higher2 = np.array([5, 255, 255])
```

```
                    mask = cv2.inRange(hsv, lower, higher)
                    mask3 = mask

                    if self.closing == 1:
                        mask3 = cv2.morphologyEx(mask3, cv2.MORPH_CLOSE, kernel)

                    if self.opening == 1:
                        mask3 = cv2.morphologyEx(mask3, cv2.MORPH_OPEN, kernel)

                    contours, hierarchy = cv2.findContours(mask3, 1,2)

                    total_width = 0
                    if len(contours)>0:
                        boxes = []

                        contours = sorted(contours, key=lambda x:
cv2.contourArea(x), reverse=True)
                        for contour in contours:
                            rect = cv2.minAreaRect(contour)
                            if abs(rect[2]) < 45:
                                width = rect[1][0]
                            else:
                                width = rect[1][1]
                            total_width += width
                            box = cv2.boxPoints(rect)
                            box = np.int32(box)
                            boxes.append(box)

                    numpix = cv2.countNonZero(mask3)
                    mask3 = cv2.cvtColor(mask3,cv2.COLOR_GRAY2RGB)
                    cv2.rectangle(mask3,(10,10), (40, 40),(0,255,255), 2)
                    #uncomment the following to put bounding boxes around
                    #filtered pixel clusters:
                    #if len(contours)>0:
                        #for box in boxes:
                            #cv2.polylines(mask3, [box], True, (0,255,255), 2)
                    frame = cv2.resize(mask3, (640,480))
                    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                    qtformat = QImage(frame.data, frame.shape[1],
frame.shape[0], QImage.Format_RGB888)

                    self.sendframe.emit(qtformat)

                    self.dyepix.emit(numpix)

                else:

                    frame = cv2.resize(nextframe, (640,480))
                    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                    qtformat = QImage(frame.data, frame.shape[1],
frame.shape[0], QImage.Format_RGB888)

                    self.sendframe.emit(qtformat)


class dataproc(QThread):

    data = []
    stop = False

    def run(self):
        pass
```

```
class MainWindow_EXEC():

    def __init__(self, parent = None):

        print("init")
        app = 0
        app = QtWidgets.QApplication(sys.argv)

        MainWindow = QtWidgets.QMainWindow()
        app.aboutToQuit.connect(self.cleanup)

        self.ui = Ui_MainWindow()
        self.ui.setupUi(MainWindow)

        self.flowpos = 0

        self.calibrating = 0
        self.calibrated = 0

        self.running = 0

        self.currentpixcount = 0
        self.currentdata = []
        self.totaldata = []

        self.calmean = [100,100]
        self.callinebounds = [0,2]

        self.datatime = []
        self.totaldatatime = []
        self.data = {}

        print(self.ui.plot1)

        self.ui.plot1.enableAutoScale()

        self.ui.plot2.enableAutoScale()

        self.curve1 = self.ui.plot1.plot(self.datatime, self.currentdata,
pen={'color':(0,0,255), 'width': 3})
        self.curve2 = self.ui.plot2.plot(self.totaldatatime, self.totaldata,
pen={'color':(0,255,0), 'width': 3})
        self.curve3 = self.ui.plot2.plot(self.callinebounds, self.calmean,
pen={'color':(255,0,255), 'width': 3})

        self.camera = cam()
        self.camera.sendframe.connect(self.update_frame)

        print('cv2.videocapture')

        if os.path.exists(f"/dev/video{camnum}"):
            self.camera.vid = cv2.VideoCapture(camnum)

            print('cam open')
            self.camera.start()
        else:
            print(f"no dev/video{camnum}")

        self.ui.button1.clicked.connect(self.getdye)
        self.ui.button1.setText('filter')

        self.ui.button2.setText('calibrate')
        self.ui.button2.clicked.connect(self.calibratebutton2)
```

```
    self.ui.button3.setText('do run')

    self.ui.button4.setText('GO')
    self.ui.button4.clicked.connect(self.go2)

    self.ui.button5.setText('open')
    self.ui.button5.clicked.connect(self.opening)

    self.ui.button6.setText('closing')
    self.ui.button6.clicked.connect(self.closing)

    self.ser = serial.Serial()
    self.ser.port = serial_device
    self.ser.baudrate = 9600
    try:
        self.ser.open()
    except:
        print(f"{serial_device} not available")

    timer = pg.QtCore.QTimer()
    timer.timeout.connect(self.updategraph)
    timer.start(100)

    MainWindow.show()
    print('before sys.exit line')

    (app.exec_())
    print('after app exec line')


def updategraph(self):

    self.curve1.setData(self.datatime, self.currentdata)
    self.curve2.setData(self.totaldatatime, self.totaldata)
    self.callinebounds = [0, time.time() - starttime]
    if self.calibrated:
        self.curve3.setData(self.callinebounds, self.calmean)

def opening(self):
    if self.camera.opening == True:
        self.camera.opening = False
    else:
        self.camera.opening = True

def closing(self):
    if self.camera.closing == True:
        self.camera.closing = False
    else:
        self.camera.closing = True

def calibrate(self, value):

    self.caldata.append(value)
    print('caldata append')
    print(len(self.caldata))
    if len(self.caldata) > 30:
        self.camera.dyepix.disconnect(self.calibrate)
        self.calibrated = 1
        self.data['cal'] = self.caldata.copy()
        print(self.data)
        print('calibrated!')
        self.calibrating = 0
        self.calmean[0] = np.mean(self.caldata)
        self.calmean[1] = np.mean(self.caldata)
```

```
    def calibratebutton2(self):
        if self.calibrating == 1:
            print ('calibrating in progress')
            return
        if self.calibrated == 1:
            print ('already calibrated')
            return
        if self.camera.getdye == 0:
            print ('turn on filtering')
            return
        else:
            print('starting calibration')
            self.calibrating = 1
            self.caldata = []

            self.camera.dyepix.connect(self.calibrate)


    def go2(self):
        if self.running == 1:
            print('already running')
            return
        if self.calibrating == 1:
            print('calibrating')
            return
        if self.calibrated == 0:
            print('need to calibrate')
            return
        if self.camera.getdye == 0:
            print('turn filter on')
        else:
            self.running = 1
            self.ui.button4.setText('Running')
            self.nextrun()


    def x(self):
        pass

    def nextrun(self):


        if self.flowpos < len(flowrates):
            flow = int(flowrates[self.flowpos]*1000)
            text = 'flow ' + str(flow) + ' \r'
            print(text)
            try:
                self.ser.write(bytes(text, encoding = 'utf-8'))
            except:
                pass
            time.sleep(0.1)
            try:
                self.ser.write(bytes(text, encoding = 'utf-8'))
            except:
                pass

            print('wrote')

            flowtime = 1800*60*resvol/(flowrates[self.flowpos])
            flowtime = max((flowtime, 50000))
            print('flowtime = ' + str(flowtime))
            #QTimer.singleShot(flowtime, lambda:
self.camera.dyepix.connect(self.pixcount))
```

```
            #use the following line instead if you want to just
            #quickly get the data collection going (without waiting the
            #required flow/flush time)
            QTimer.singleShot(5, lambda:
self.camera.dyepix.connect(self.pixcount))

        else:
            for i in range(30):
                print('done run')
            for key in self.data.keys():

                np.save(filesave, self.data)


    def pixcount(self, value):

        self.currentdata.append(value)
        self.datatime.append(time.time() - starttime)

        self.totaldata.append(value)
        self.totaldatatime.append(time.time()-starttime)
        if len(self.currentdata) > 900:

            print('> 10')
            print(self.currentdata)
            a = flowrates[self.flowpos]
            print (a)

            self.data[a] = self.currentdata.copy()
            self.camera.dyepix.disconnect(self.pixcount)

            self.currentdata.clear()
            self.datatime.clear()
            self.flowpos += 1
            self.nextrun()

    def picdata(self, data):
        self.data.append(data)
        print(len(self.data))

        pass

    def cleanup(self):
        print('start cleanup')
        global shutdown
        shutdown = True
        time.sleep(0.1)
        if self.camera.vid:
            self.camera.vid.release()
            print ("cam release")
            del self.camera
        print('cleanup')
        time.sleep(0.1)

    def update_frame(self, image):
        pixmap = QPixmap.fromImage(image)
        self.ui.label.setPixmap(pixmap)

    def getdye(self):
        if self.camera.getdye == True:
            self.camera.getdye = False
        else:
            self.camera.getdye = True
```

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

pyqt5 gui definition file (required by main control script)
Should be called mainwindow.py (if using control script provided)

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1104, 864)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.button1 = QtWidgets.QPushButton(self.centralwidget)
        self.button1.setGeometry(QtCore.QRect(20, 20, 121, 27))
        self.button1.setObjectName("button1")
        self.button2 = QtWidgets.QPushButton(self.centralwidget)
        self.button2.setGeometry(QtCore.QRect(150, 20, 131, 27))
        self.button2.setObjectName("button2")
        self.label = MyLabel(self.centralwidget)

        self.label.setGeometry(QtCore.QRect(70, 290, 640, 480))
        self.label.setFrameShape(QtWidgets.QFrame.WinPanel)
        self.label.setLineWidth(0)
        self.label.setObjectName("label")
        self.plot1 = PlotWidget(self.centralwidget)
        self.plot1.setGeometry(QtCore.QRect(780, 10, 321, 221))
        self.plot1.setObjectName("plot1")
        self.plot2 = PlotWidget(self.centralwidget)
        self.plot2.setGeometry(QtCore.QRect(450, 10, 321, 221))
        self.plot2.setObjectName("plot2")
        self.button3 = QtWidgets.QPushButton(self.centralwidget)
        self.button3.setGeometry(QtCore.QRect(150, 50, 131, 27))
        self.button3.setObjectName("button3")
        self.button4 = QtWidgets.QPushButton(self.centralwidget)
        self.button4.setGeometry(QtCore.QRect(20, 50, 121, 27))
        self.button4.setObjectName("button4")
        self.button5 = QtWidgets.QPushButton(self.centralwidget)
        self.button5.setGeometry(QtCore.QRect(150, 80, 131, 27))
        self.button5.setObjectName("button5")
        self.button6 = QtWidgets.QPushButton(self.centralwidget)
        self.button6.setGeometry(QtCore.QRect(20, 80, 121, 27))
        self.button6.setObjectName("button6")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1104, 37))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.button1.setText(_translate("MainWindow", "X"))
        self.button2.setText(_translate("MainWindow", "X"))
        self.label.setText(_translate("MainWindow", "TextLabel"))
        self.button3.setText(_translate("MainWindow", "X"))
        self.button4.setText(_translate("MainWindow", "X"))
        self.button5.setText(_translate("MainWindow", "X"))
        self.button6.setText(_translate("MainWindow", "X"))
```

```
from mylabel import MyLabel
from pyqtgraph import PlotWidget
```

Additional gui script (imported by mainwindow.py):

```
#required for mainwindow.py:

from PyQt5 import QtCore, QtGui, QtWidgets, Qt
from PyQt5.QtWidgets import QRubberBand
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtCore import pyqtSlot, pyqtSignal, QObject, QThread, QPoint, QRect,
QSize
from PyQt5.QtCore import Qt

class MyLabel(QtWidgets.QLabel):

      box = pyqtSignal(tuple)
      r2 = (0,0,0,0)

      def __init__(self, parent = None):
            QtWidgets.QLabel.__init__(self, parent)
            self.rubberBand = QRubberBand(QRubberBand.Rectangle, self)

      def mousePressEvent(self, event):

            if event.button() == Qt.LeftButton:
                  self.origin = QPoint(event.pos())
                  self.rubberBand.setGeometry(QRect(self.origin, QSize()))
                  self.rubberBand.show()

      def mouseMoveEvent(self, event):

            if not self.origin.isNull():
                  self.rubberBand.setGeometry(
                        QRect(self.origin, event.pos()).normalized())
                  print("moved")
                  r = (self.rubberBand.geometry())
                  #print(r)
                  self.r2 = (r.left(), r.right(), r.top(), r.bottom())
                  self.r3 = (r.top(), r.left(), r.right()-r.left(), r.bottom()-
r.top(),)



      def mouseReleaseEvent(self, event):

            if event.button() == Qt.LeftButton:

                  self.rubberBand.hide()
                  self.box.emit(self.r2)
```

Data processing script:

```
'''
Copyright (c) 2022 Matthew O'Brien

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
'''
#-------------------------------------------------------


#this script just opens two data files and then minimises the
#parameters to fit the CO2 out-gassing data
#it will need to be modified to produce all the various plots in the paper

import numpy as np
from matplotlib import pyplot as plt
import os
from scipy.optimize import leastsq as lsq


def RRMSE(true, pred):
    true = np.array(true)
    pred = np.array(pred)
    rrmse_loss = np.sqrt((np.sum(np.square(true -
pred)))/(np.sum(np.square(pred))))
    return rrmse_loss

radius = 0.03 #cm
height = 28 #cm
vol = np.pi*(radius**2)*height


gasvol = 24.0
molwt = 44.009
dens = 1.842
gasvol2 = molwt/dens
flowfact = 1.0


#whatever the filenames are to be opened/processed:
filenames = [
             'xxx.npy',
             'yyy.npy'
            ]

dataparams = {}
moddatas = {}
concdatas = {}
```

```
concslist = []

xspace = np.linspace(0, 10, 100)

def expo(t, k, S):
    return S*(1-np.exp(-k*t))

def expof(t, k, S, f):
    return np.maximum(((S*(1-np.exp(-k*t)) - f)), 0)

def expoflist(pars, t, concslist):
    #function used in scipy least-squares minimisation
    # pars in order: k, S, f (f is equivalent to residual R in manuscript)
    # note: pars always seems to get flattened to 1D array.
    # the following code resorts the k, S, f parameters:

    t = np.array(t)
    m = []
    r = []
    k = []
    S = []
    f = []
    offset = len(concslist)
    for i in range(int(len(pars)/3)):
        start = 0*offset
        k.append(pars[i+start])
    for i in range(int(len(pars)/3)):
        start = 1*offset
        S.append(pars[i+start])
    for i in range(int(len(pars)/3)):
        start = 2*offset
        f.append(pars[i+start])

    for i in range(len(concslist)):

        this_m = np.maximum((S[i]*(1-np.exp(-k[i]*t))-f[0]), 0)

        m.append(this_m)
        res = np.array(concslist[i]) - this_m
        r.append(res)

    ret = np.concatenate(r)
    return ret

for r in range(len(filenames)):

    data = np.load(filenames[r], allow_pickle=True)
    datas = {}

    #the flowrates must match the ones that were used to obtain the data
    flowrates = [0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0]

    flowrates = np.array(flowrates)
    means = []
    caldata = data.item().get('cal')
    calmean = np.mean(caldata)

    for flowrate in flowrates:
        datas[flowrate] = data.item().get(flowrate)
        mean = np.mean(datas[flowrate][:])
        means.append(mean)

    gaslens = calmean - means
    liqlens = calmean - gaslens
```

```
    concs = gaslens/liqlens
    concs = concs/gasvol2
    rettimes = vol/np.array(flowrates)
    rettimes = rettimes/flowfact

    concslist.append(np.array(concs))

#sets up list of initial parameters (guess) for
#scipy least squares optimisation
#note: this creates two initial values of f but only one
#is required (and minimised).
p = [[],[],[]]
ii = len(concslist)

for i in range(ii):
    p[0].append(2)
    p[1].append(4)
    p[2].append(2)

print('p = ')
print(p)

parms, cov = lsq(expoflist, p, args=(rettimes, concslist), maxfev = 50000)
print("parms: ")
print(parms)

xspace = np.linspace(0, 0.5, 200)
ii = len(concslist)

colors = ['blue', 'red']
colors2 = ['#33b8ff', 'pink']

pressures = [5, 10]

for i in range(len(concslist)):

    bubbleconcpred = expof(rettimes, parms[i], parms[ii+i], parms[4])
    bubble_rrmse = RRMSE(concslist[i], bubbleconcpred)
    print("pressure: ", pressures[i])
    print("rrmse: ", bubble_rrmse)

    plt.plot(xspace, expof(xspace, parms[i], parms[ii+i], parms[ii*2]),
linewidth=3, label=str(pressures[i])+" bar fitted")
    plt.scatter(rettimes, concslist[i], s=80, zorder=11, color = colors[i],
label=str(pressures[i])+" bar bubble counting")

plt.grid(b = True, which = 'major',
            color = 'grey', alpha = 0.3, linestyle = '--', linewidth = 1)

plt.xlabel('residence-time/min', fontsize=20)
plt.ylabel('[CO2]/(mol/L)', fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

plt.xlim(-0.005, 0.3)
plt.ylim(-0.05, 1.5)

plt.legend(loc="lower right", fontsize=15)

plt.show()
print (vol)
```