main_code

The app.py file is the program entry, the mainwnd.py file is the graphical interface of the program, the model.py file is the main logic of the program, and the controller.py is used to pass data and operation signals.

app.py

```
import os
import sys
from PySide6.QtWidgets import QApplication
from view.main_wnd import MainWnd
from model.model import Model
from controller.controller import Controller

class App(QApplication):
    def __init__(self, sys_argv):
        super(App, self).__init__(sys_argv)
        self.model = Model()
        self.view = MainWnd()
        self.controller = Controller(self.model, self.view)
        self.view.resize(800, 600)
        self.view.show()
        self.model.load_config()

if __name__ == "__main__":
    path = os.path.dirname(__file__)
    if path not in sys.path:
        sys.path.insert(0, os.path.dirname(__file__))
    app = App(sys.argv)
    sys.exit(app.exec())
```

mainwnd.py

```
import json
from PySide6.QtCore import Slot, Signal, Qt
from PySide6.QtGui import QPainter, QColor, QPen
from PySide6.QtWidgets import (
    QWidget, QLabel, QLineEdit, QPushButton, QHBoxLayout,
    QVBoxLayout
)
from PySide6.QtCharts import QChart, QChartView, QLineSeries
import numpy as np


class MainWnd(QWidget):
    set_spectrometer_signal = Signal(int, int, int)
    set_wavelength_signal = Signal(list, list)
    get_spectrum_signal = Signal()
    start_signal = Signal()
    stop_signal = Signal()
    wnd_close_signal = Signal()
    def __init__(self):
        super().__init__()


        self.boxcar_width_label = QLabel("Boxcar Width: ")
        self.boxcar_width_editor = QLineEdit()
        self.integration_time_label = QLabel("Integration Time(μs): ")
        self.integration_time_editor = QLineEdit()
        self.scans_to_average_label = QLabel("Scans To Average: ")
        self.scans_to_average_editor = QLineEdit()
        self.set_spectrometer_btn = QPushButton("设置采集参数")
        self.set_spectrometer_btn.clicked.connect(self.on_set_spectrometer_btn_clicked)
        self.get_spectrum_btn = QPushButton("光谱测试")
        self.get_spectrum_btn.clicked.connect(self.get_spectrometer_data)
        self.enhance_wavelength_label = QLabel("Enhance Wavelength: ")
        self.enhance_wavelength_editor = QLineEdit()
        self.suppress_wavelength_label = QLabel("Suppress Wavelength: ")
        self.suppress_wavelength_editor = QLineEdit()
        self.set_wavelength_btn = QPushButton("设置光谱波段")
        self.set_wavelength_btn.clicked.connect(self.on_set_wavelength_btn_clicked)
        self.start_btn = QPushButton("开始")
        self.start_btn.clicked.connect(self.on_start_btn_clicked)
        self.stop_btn = QPushButton("停止")
        self.stop_btn.clicked.connect(self.on_stop_btn_clicked)

        self.stop_btn.setEnabled(False)


        self.series = None
        self.chart = QChart()
        self.chart.legend().hide()
        self.chart.setTitle("Spectrum")
        self.chart_view = QChartView(self.chart)
        self.chart_view.setRenderHint(QPainter.Antialiasing)


        self.config_layout = QVBoxLayout()
```

```
        self.config_layout.addWidget(self.boxcar_width_label)
        self.config_layout.addWidget(self.boxcar_width_editor)
        self.config_layout.addWidget(self.integration_time_label)
        self.config_layout.addWidget(self.integration_time_editor)
        self.config_layout.addWidget(self.scans_to_average_label)
        self.config_layout.addWidget(self.scans_to_average_editor)
        self.config_layout.addWidget(self.set_spectrometer_btn)
        self.config_layout.addWidget(self.get_spectrum_btn)
        self.config_layout.addWidget(self.enhance_wavelength_label)
        self.config_layout.addWidget(self.enhance_wavelength_editor)
        self.config_layout.addWidget(self.suppress_wavelength_label)
        self.config_layout.addWidget(self.suppress_wavelength_editor)
        self.config_layout.addWidget(self.set_wavelength_btn)
        self.config_layout.addWidget(self.start_btn)
        self.config_layout.addWidget(self.stop_btn)
        self.config_layout.addStretch()

        self.layout = QHBoxLayout(self)
        self.layout.addLayout(self.config_layout)
        self.layout.addWidget(self.chart_view, 120)

        self.enhance_wavelength = []
        self.suppress_wavelength = []
        self.wavelength = np.array([])
        self.spectrum = np.array([])

    @Slot(dict)
    def on_load_config(self, config):
        spectrometer_cfg = config["spectrometer"]
        self.boxcar_width_editor.setText("{0:d}".format(spectrometer_cfg["boxcar_width"]))
        self.integration_time_editor.setText("{0:d}".format(spectrometer_cfg["integration_time"]))
        self.scans_to_average_editor.setText("{0:d}".format(spectrometer_cfg["scans_to_average"]))

        wavelength_cfg = config["wavelength"]
        self.enhance_wavelength = wavelength_cfg["enhance_wavelength"]
        self.suppress_wavelength = wavelength_cfg["suppress_wavelength"]
        self.enhance_wavelength_editor.setText(json.dumps(self.enhance_wavelength))
        self.suppress_wavelength_editor.setText(json.dumps(self.suppress_wavelength))

    @Slot()
    def on_set_spectrometer_btn_clicked(self):
        try:
            boxcar_width = int(self.boxcar_width_editor.text())
            integration_time = int(self.integration_time_editor.text())
            scans_to_average = int(self.scans_to_average_editor.text())
            self.set_spectrometer_signal.emit(boxcar_width, integration_time, scans_to_average)
        except Exception as e:
            print(repr(e))

    @Slot()
    def on_set_wavelength_btn_clicked(self):
        try:
            self.enhance_wavelength = list(set(
                json.loads(self.enhance_wavelength_editor.text().replace("，", ",").replace("。", "."))
            ))
            self.suppress_wavelength = list(set(
                json.loads(self.suppress_wavelength_editor.text().replace("，", ",").replace("。", "."))
            ))
            self.set_wavelength_signal.emit(self.enhance_wavelength, self.suppress_wavelength)
            self.update_chart()
        except Exception as e:
            print(repr(e))


    @Slot()
    def get_spectrometer_data(self):
        self.get_spectrum_signal.emit()

    @Slot()
    def update_chart(self, spectrum=None, wavelength=None):
        if spectrum is not None:
            self.spectrum = spectrum
        if wavelength is not None:
            self.wavelength = wavelength

        if self.spectrum.size == 0 or self.wavelength.size == 0:
            return

        if self.series is not None:
            self.chart.removeAllSeries()

        min_value = self.spectrum.min()
        max_value = self.spectrum.max()
        for value in self.enhance_wavelength:
            if value < 10:
                continue
            enhance_series = self.get_axis_series(
                value, min_value, max_value, QColor(180, 0, 0)
            )
            self.chart.addSeries(enhance_series)
        for value in self.suppress_wavelength:
            suppress_series = self.get_axis_series(
                value, min_value, max_value, QColor(0, 150, 0)
            )
```

```
            self.chart.addSeries(suppress_series)

        self.series= QLineSeries()
        self.series.setColor(QColor(32,159,223))
        for wave, intensity in zip(self.wavelength, self.spectrum):
            self.series.append(wave, intensity)

        self.chart.addSeries(self.series)

        self.chart.createDefaultAxes()

    def get_axis_series(self, pos, min_value, max_value, color):
        series = QLineSeries()
        series.setPen(Qt.DashLine)
        series.setColor(color)
        series.append(pos, min_value)
        series.append(pos, max_value)
        return series

    @Slot()
    def on_start_btn_clicked(self):
        self.set_spectrometer_btn.setEnabled(False)
        self.get_spectrum_btn.setEnabled(False)
        self.start_btn.setEnabled(False)
        self.stop_btn.setEnabled(True)
        self.start_signal.emit()

    @Slot()
    def on_stop_btn_clicked(self):
        self.set_spectrometer_btn.setEnabled(True)
        self.get_spectrum_btn.setEnabled(True)
        self.start_btn.setEnabled(True)
        self.stop_btn.setEnabled(False)
        self.stop_signal.emit()

    @Slot()
    def on_auto_search_finish(self):
        self.on_stop_btn_clicked()

    def closeEvent(self, event):
        self.wnd_close_signal.emit()
        event.accept()
```

model.py

```
import sys
import time
import datetime
import numpy as np
from threading import Thread, Lock
from scipy.optimize import minimize, Bounds
from scipy.signal import find_peaks
from sko.GA import GA
from PySide6.QtCore import Signal, Slot, QObject
from utils.utils import get_config, save_config
import os
os.add_dll_directory("C:/Program Files/Ocean Optics/OmniDriverSPAM/OOI_HOME")
from oceaninsightcamera import Camera
from alp4lib import *

np.set_printoptions(threshold=sys.maxsize, linewidth=sys.maxsize, precision=2, suppress=True)


class Model(QObject):
    lock = Lock()
    on_load_config_signal = Signal(dict)
    on_get_spectrum_signal = Signal(np.ndarray, np.ndarray)
    on_auto_search_finish_signal = Signal()
    def __init__(self):
        super().__init__()
        self.spectrometer_ret = -1
        self.init_spectrometer()
        self.dmd_ret = -1
        self.init_dmd()
        self.is_auto_searching = False
        self.bounds = Bounds(-1,1)
        self.score = None
        self.scoreValid = False
        # self.load_config()
        self.last_spectrum = np.array([])
        self.save_spectrum = False

    def init_dmd(self):
        try:
            self.dmd = ALP4(version = '4.3')
            self.dmd.Initialize()
            self.dmd_ret = 0
        except Exception as e:
            if "The specified ALP device has not been found or is not ready" in repr(e):
                self.dmd_ret = -1

    def init_spectrometer(self):
        self.spectrometer = Camera()
```

```
        self.spectrometer_ret = self.spectrometer.init_camera()
        if self.spectrometer_ret == 0:
            self.spectrometer.register_callback(self.on_spectrum_captured)

    def on_spectrum_captured(self, spectrum, wavelength):
        if wavelength is None:
            wavelength = self.default_wavelength_data
        if spectrum is None:
            spectrum = self.default_spectrum_data

        spectrum = np.array(spectrum, copy=False)
        if self.save_spectrum:
            self.last_spectrum = spectrum.copy()
        wavelength = np.array(wavelength, copy=False)
        start_wave, end_wave = self.wavelength_cfg["suppress_wavelength"]

        show_scope = (wavelength > start_wave-10)*(wavelength < end_wave+10)
        self.on_get_spectrum_signal.emit(spectrum[show_scope], wavelength[show_scope])
        if self.is_auto_searching:
            max_suppress_prominence = 0
            min_enhance_prominence = 0
            cal_scope = (wavelength > start_wave) * (wavelength < end_wave)
            cal_wavelength = wavelength[cal_scope]
            cal_spectrum = spectrum[cal_scope]
            peaks, properties = find_peaks(
                cal_spectrum, distance=self.peak_pixel_distance, prominence=self.peak_prominence, wlen=self.peak_pixel_distance*2)
            prominences = properties["prominences"]

            enhance_wavelength = self.wavelength_cfg["enhance_wavelength"]
            if enhance_wavelength[0] < 10:

                select_peak_num = enhance_wavelength[0]
                if prominences.size >= select_peak_num:
                    prominences.sort()
                    min_enhance_prominence = prominences[-select_peak_num]
                    if prominences.size > select_peak_num:
                        max_suppress_prominence = prominences[-(select_peak_num+1)]
            else:

                enhance_prominences = np.array([])
                for enhance_wavelength in self.wavelength_cfg["enhance_wavelength"]:
                    if peaks.size == 0:
                        continue

                    wavelength_index = abs(cal_wavelength - enhance_wavelength).argsort()[0]

                    abs_peaks = abs(peaks-wavelength_index)
                    peak_index = abs_peaks.argsort()[0]
                    if abs_peaks[peak_index] > self.peak_pixel_distance:
                        enhance_prominences = np.append(enhance_prominences, 0)
                    else:
                        enhance_prominences = np.append(enhance_prominences, prominences[peak_index])
                        prominences = np.delete(prominences, peak_index)
                        peaks = np.delete(peaks, peak_index)
                if enhance_prominences.size > 0:
                    min_enhance_prominence = enhance_prominences.min()
                if prominences.size > 0:
                    max_suppress_prominence = prominences.max()


            self.score = (500+max_suppress_prominence) / (min_enhance_prominence + 1e-3)
            self.scoreValid = True

    def load_config(self):
        self.config = get_config()
        assert(self.config)
        self.on_load_config_signal.emit(self.config)


        self.max_iter = self.config['max_iter']
        self.block_cols = self.config['block_cols']
        self.block_rows = self.config['block_rows']
        self.peak_pixel_distance = self.config['peak_pixel_distance']
        self.peak_prominence = self.config['peak_prominence']

        self.y_pixel = self.config['y_pixel']
        self.x_pixel = self.config['x_pixel']

        self.spectrometer_cfg = self.config["spectrometer"]
        self.wavelength_cfg = self.config["wavelength"]
        self.default_wavelength_data = np.array(self.config["default_wavelength_data"])
        self.default_spectrum_data = np.array(self.config["default_spectrum_data"])
        self.set_spectrometer()

    def save_config(self):
        save_config(self.config)

    def set_spectrometer(self):
        if self.spectrometer is not None and self.spectrometer_ret == 0:
            self.spectrometer.setup_camera(
                {
                    "BoxcarWidth" : self.spectrometer_cfg["boxcar_width"],
                    "IntegrationTime" : self.spectrometer_cfg["integration_time"],
                    "ScansToAverage" : self.spectrometer_cfg["scans_to_average"]
```

```
            }
        )

@Slot(int, int, int)
def save_spectrometer_config(
        self, boxcar_width, integration_time, scans_to_average):
    self.spectrometer_cfg["boxcar_width"] = boxcar_width
    self.spectrometer_cfg["integration_time"] = integration_time
    self.spectrometer_cfg["scans_to_average"] = scans_to_average
    self.save_config()
    self.set_spectrometer()

@Slot(list, list)
def save_wavelength_config(
        self, enhance_wavelength, suppress_wavelength):
    enhance_wavelength.sort()
    self.wavelength_cfg["enhance_wavelength"] = enhance_wavelength
    suppress_wavelength.sort()
    self.wavelength_cfg["suppress_wavelength"] = suppress_wavelength
    self.save_config()

@Slot()
def get_spectrum_data(self):
    if self.spectrometer_ret == 0:
        self.spectrometer.get_captured_data()

@Slot()
def start(self):
    if self.is_auto_searching:
        return
    self.epoch = 0
    self.is_auto_searching = True
    self.auto_search_thread = Thread(target=self.auto_search)
    self.auto_search_thread.daemon = True
    self.auto_search_thread.start()

def auto_search(self):
    self.ga()
    self.on_auto_search_finish_signal.emit()


def simplex(self):
    initial_simplex = np.random.rand(self.block_rows*self.block_cols+1, self.block_rows*self.block_cols) * 2 - 1
    minimize(
        self.cal_object_fun, initial_simplex[0], method="Nelder-Mead", bounds=self.bounds, callback=self.callback,
        options={"initial_simplex":initial_simplex, "adaptive":True, "maxiter":self.max_iter}
    )

def ga(self):
    self._ga =GA(
        func=self.cal_object_fun, n_dim=self.block_rows*self.block_cols, max_iter=self.max_iter,
        lb=0, ub=1, precision=1, callback=self.callback
    )
    best_x, best_y = self._ga.run()
    self.cal_object_fun(best_x, save_spectrum=True)


def cal_object_fun(self, address, save_spectrum=False):
    self.lock.acquire()
    if self.is_auto_searching:
        self.save_spectrum = save_spectrum
        if not isinstance(address, np.ndarray):
            address = np.array(address)
        address = (address > 0) + 0
        if save_spectrum:
            addressStr = np.array2string(address, separator=',')
        address = address.reshape(self.block_rows, self.block_cols) * 255
        if self.dmd_ret == 0:
            img = np.zeros([self.dmd.nSizeY, self.dmd.nSizeX])
            address = np.repeat(address, self.y_pixel, axis=0)
            address = np.repeat(address, self.x_pixel, axis=1)
            start_y = (self.dmd.nSizeY-self.y_pixel*self.block_rows) // 2
            start_x = (self.dmd.nSizeX-self.x_pixel*self.block_cols) // 2
            img[start_y:start_y+address.shape[0],start_x:start_x+address.shape[1]]=address
            self.dmd.SeqAlloc()
            self.dmd.SeqPut(imgData=img.ravel())
            self.dmd.Run()

        self.scoreValid = False
        if self.spectrometer_ret == 0:
            self.get_spectrum_data()
        else:
            self.on_spectrum_captured(None, None)
        while not self.scoreValid:
            time.sleep(0.2)

        self.save_spectrum = False
    self.lock.release()
    return self.score if self.is_auto_searching else sys.maxsize

def callback(self, result=None):
    self.epoch += 1
    self.cal_object_fun(result, save_spectrum=True)
    if not self.is_auto_searching:
```

```
            raise StopIteration()

    @Slot()
    def stop(self):
        self.is_auto_searching = False

    @Slot()
    def wnd_close(self):
        if self.spectrometer_ret == 0:
            self.spectrometer.close_camera()
        if self.dmd_ret == 0:
            try:
                self.dmd.Halt()
                self.dmd.FreeSeq()
                self.dmd.Free()
            except Exception as e:
                print(repr(e))
```

controller.py

```
from tokenize import Single
from PySide6.QtCore import QObject, Signal, Slot
from numpy import ndarray

class Controller(QObject):
    on_load_config_signal = Signal(dict)
    on_get_spectrum_signal = Signal(ndarray, ndarray)
    on_auto_search_finish_signal = Signal()

    save_spectrometer_config_signal = Signal(int, int, int)
    save_wavelength_config_signal = Signal(list, list)
    get_spectrum_signal = Signal()

    start_signal = Signal()
    stop_signal = Signal()
    wnd_close_signal = Signal()

    def __init__(self, model, view):
        super().__init__()
        self.model = model
        self.view = view

        self.model.on_load_config_signal[dict].connect(self.on_load_config_signal)
        self.model.on_get_spectrum_signal[ndarray, ndarray].connect(self.on_get_spectrum_signal)
        self.model.on_auto_search_finish_signal.connect(self.on_auto_search_finish_signal)
        self.on_load_config_signal[dict].connect(self.view.on_load_config)
        self.on_get_spectrum_signal[ndarray, ndarray].connect(self.view.update_chart)
        self.on_auto_search_finish_signal.connect(self.view.on_auto_search_finish)

        self.view.set_spectrometer_signal[int, int, int].connect(self.save_spectrometer_config_signal)
        self.view.set_wavelength_signal[list, list].connect(self.save_wavelength_config_signal)
        self.view.get_spectrum_signal.connect(self.get_spectrum_signal)
        self.view.start_signal.connect(self.start_signal)
        self.view.stop_signal.connect(self.stop_signal)
        self.view.wnd_close_signal.connect(self.wnd_close_signal)
        self.save_spectrometer_config_signal[int, int, int].connect(self.model.save_spectrometer_config)
        self.save_wavelength_config_signal[list, list].connect(self.model.save_wavelength_config)
        self.get_spectrum_signal.connect(self.model.get_spectrum_data)
        self.start_signal.connect(self.model.start)
        self.stop_signal.connect(self.model.stop)
        self.wnd_close_signal.connect(self.model.wnd_close)
```