BoxFishHost 上位机代码文档

1 基本介绍

1.1 功能简介

本程序基于Python+PyQt5开发，在Linux及Windows下均可使用，但需要配置Python环境，主要包括以下几项：

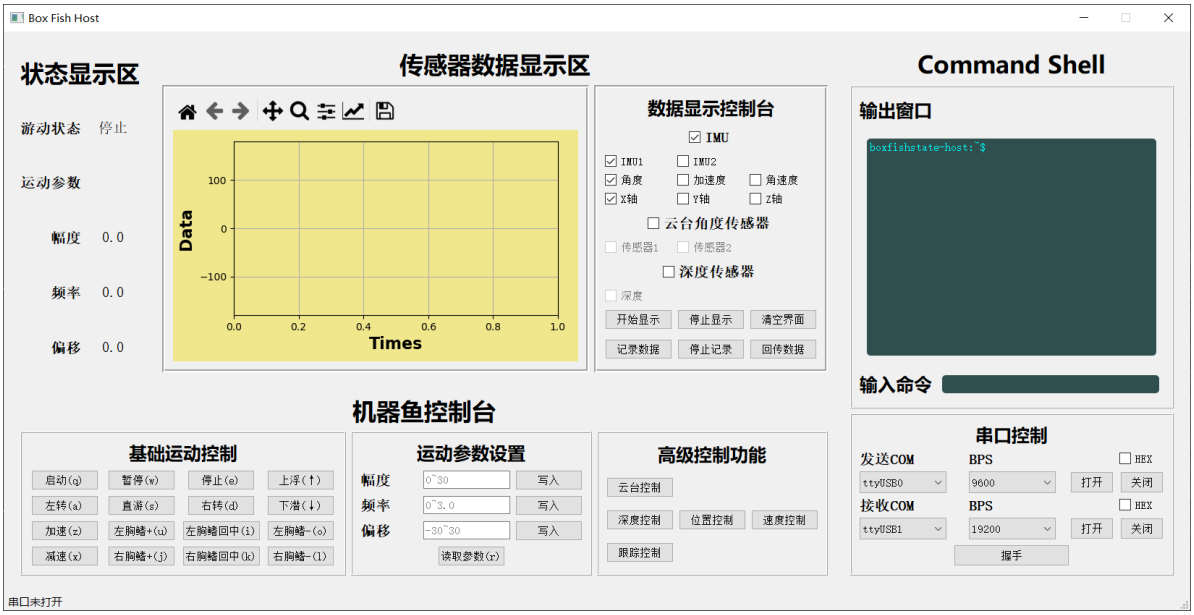
- Python 3
- PyQt5
- pyserial
- matplotlib

此外，本程序需配合BoxFish单片机程序共同使用，并且通过RFLink通讯协议与BoxFish下位机通讯。

本软件具备的功能包括：

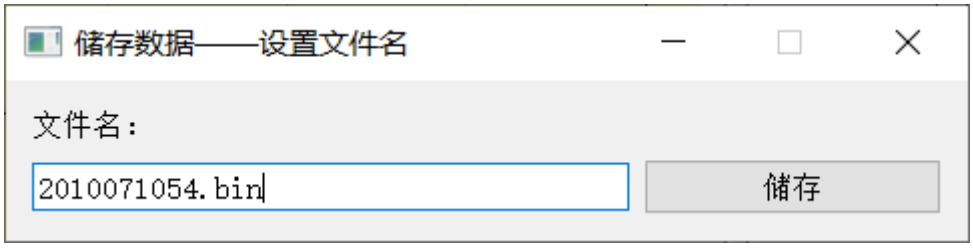
- 机器鱼状态显示
- 传感器数据显示
- 传感器数据保存和回传
- 串口的打开、波特率的设置
- 简易的命令行界面，可由命令行下发指令
- 机器鱼基础运动控制指令的下达，包括直游、转向、上浮、下潜等等
- 机器鱼运动参数的设置
- 可定制的高级控制功能，例如，云台控制、深度控制等等，可由用户自定义新的窗口

1.2 界面视图

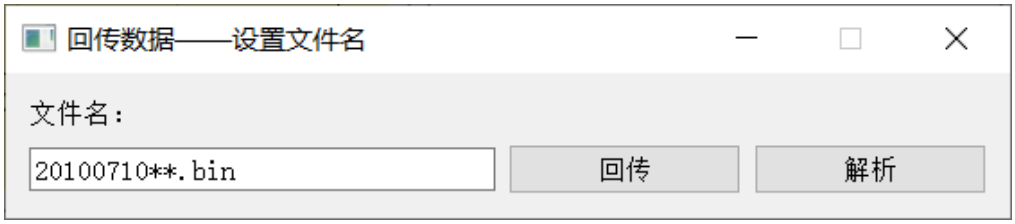


1.3 软件使用

- 状态显示区：显示机器鱼运动状态
- 传感器数据显示区：显示传感器数据
  - 数据显示控制台：
    - 使用方法：选择想要查看的数据，打钩，然后点击开始显示，即开始显示数据；点击停止显示，就暂停显示数据；点击清空界面，则清除左侧的整个画布
    - **记录数据**按钮：点击以后，会出现如下对话框，设置好文件的名字，然后点击储存，可以通知机器鱼开始记录数据，但是记录什么数据，需要在下位机的代码中编写



    - **停止记录**按钮：点击以后，会发送一个指令，通知机器鱼停止记录数据
    - **回传数据**按钮：点击以后，会出现如下对话框，设置好文件的名字，点击回传，机器鱼就会将对应文件名的文件回传回来。此外，还有一个解析按钮，这里面开始分析的功能需要用户针对文件中记录的数据修改代码。



- Comman Shell
  - 输入命令窗口：手动输入指令，下发给机器鱼，但是需要打开串口才能使用
    - ls 命令：罗列出机器鱼SD卡上所有的文件名
    - help 命令：显示帮助信息
    - clear 命令：清除输出窗口中的所有信息
    - SET * 系列命令：这是所有设置机器鱼状态的命令，包括SET_SWIM_RUN、SET_SWIM_STOP、SET_SWIM_FORCESTOP等等
    - READ * 系列命令：这是所有读取机器鱼状态的命令，包括READ_ROBOT_STATUS、READ_CPG_PARAM、READ_SINE_MOTION_PARAM等等
    - GOTO * 系列命令：这是所有控制机器鱼执行某些操作的命令，包括GOTO_ATTACH、GOTO_DETACH、GOTO_STORAGE_DATA等等
- 机器鱼控制台：
  - 基础运动控制：包括了多个按钮，按钮中文名表明了按钮的功能，后面括号中的字母代表按钮的快捷键
  - 运动参数设置：这里用于设置机器鱼尾鳍运动的幅度、频率和偏移，允许的输入范围显示在对话框中，点击写入则发送至机器鱼。点击读取参数按钮，则会读取机器鱼当前的运动参数
  - 高级控制功能： 这里有多个按钮，每一个都对应一个新的对话框，用户可以在这里添加自己的应用，比如要添加一个姿态控制的功能，就可以设计一个新的对话框，然后再这里增加一个新的姿态控制按钮
- 串口控制：
  - 选择串口设备和波特率
  - 握手按钮：每次在使用上位机时，最初都需要点击这个握手按钮，和机器鱼建立连接，之后才可以继续向机器鱼发送其他指令

2 代码结构

主要文件：

- boxfishhost.py

    - 这个文件是最主要的，所有关于上位机主界面的操作都在这个文件中
    - 定义了四个线程（包括主界面线程在内）：
        - PollingStateThread：以固定周期给下位机发送READ_ROBOT_STATUS指令，读取机器人信息，默认为1秒读一次
        - ReceiveDataThread：接收串口数据，每当串口接收到一个数据就进入这个线程，并且进入RFLink状态机。当接收到一帧完整的RFLink message的时候，就会唤醒AnalysisDataThread线程
        - AnalysisDataThread：分析接收到RFLink message中的数据，每次消息一来，就会调用analysis_data函数去进行数据分析，并将分析所得发送给BoxFishWindow
        - BoxFishWindow：主界面窗口，等待按键、编辑器等等UI控件的响应
            - 核心函数：除了一些界面初始化的函数，最重要的函数就是newdata_comming_slot
            - newdata_comming_slot：
                - 这个函数中定义了，当上位机接收到下位机发送的RFLink消息后，进行何种操作
- robotstate.py
    - 这个文件主要定义了机器人核心的一些状态，包括游动状态，游动参数等等
- serctl.py
    - 定义了一个RobotSerial串口类，包括打开/关闭串口，接收/发送数据等功能
- rflink.py
    - 定义了Command枚举类型，这里的Command列表需要与下位机代码中完全一致
    - 定义了RFLink工具类，包括RFLink消息接收状态机，RFLink数据与指令打包函数
- sensor_data_canvas.py
    - 定义了数据显示窗口类
- _btn_win.py
    - 各种高级控制功能的子界面，子窗口，用户可以自己去定义属于自己的子界面

3 代码使用说明

3.1 添加新的控制窗口的方法

3.1.1 第一步

新建一个子窗口内，比如说深度控制窗口类

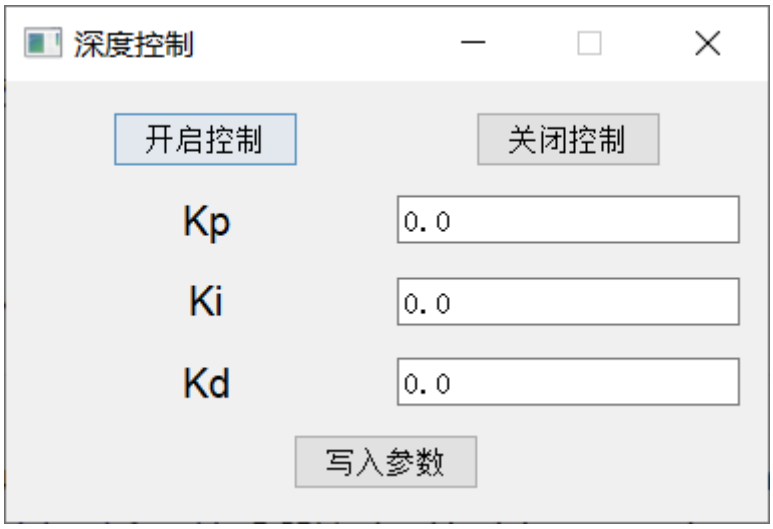
新建一个文件，取名为：depth_control_btn_win.py，保存在childwindows文件夹下，内容可以如下：

```
import os
import sys
from PyQt5 import QtCore,QtGui,QtWidgets

### 子界面类
class DepthControlBtnWin(QtWidgets.QWidget):
    _signal = QtCore.pyqtSignal(str)

    ## 初始化函数
    def __init__(self, parent=None):
        super(DepthControlBtnWin, self).__init__(parent)
        self.init_ui()

    ## 初始化串口UI界面
    def init_ui(self):

        # 窗口设置
        self.setFixedSize(380, 220)  # 设置窗体大小
        self.setWindowTitle('深度控制')  # 设置窗口标题

        # 布局
        self.main_layout = QtWidgets.QGridLayout()
        self.setLayout(self.main_layout)

        # 定义UI界面，这里由用户自己定义
        self.depthctl_start_button = QtWidgets.QPushButton('开启控制')
        self.main_layout.addWidget(self.depthctl_start_button, 0, 0, 1, 1, QtCore.Qt.AlignCenter)
        # 这里需要给按钮设置一个名字，因为之后需要将这个按钮与一个函数链接起来，那个函数会根据这个名字给下位机发送指令
        # 所以这个按钮的名字，应该作为一条指令添加到RFLink的Command中

 self.depthctl_start_button.setObjectName("SET_DEPTHCTL_START")
```

```
        self.depthctl_stop_button = QtWidgets.QPushButton('关闭控
制')
        self.main_layout.addWidget(self.depthctl_stop_button, 0,
1, 1, 1, QtCore.Qt.AlignCenter)

 self.depthctl_stop_button.setObjectName("SET_DEPTHCTL_STOP")

        self.depthctl_param_kp_label = QtWidgets.QLabel('Kp')
        self.depthctl_param_kp_label.setFont(QtGui.QFont('Arial',
12))
        self.main_layout.addWidget(self.depthctl_param_kp_label,
1, 0, 1, 1, QtCore.Qt.AlignCenter)

        self.depthctl_param_kp_edit = QtWidgets.QLineEdit()
        self.main_layout.addWidget(self.depthctl_param_kp_edit,
1, 1, 1, 1, QtCore.Qt.AlignCenter)
        self.depthctl_param_kp_edit.setText('0.0')

        self.depthctl_param_ki_label = QtWidgets.QLabel('Ki')
        self.depthctl_param_ki_label.setFont(QtGui.QFont('Arial',
12))
        self.main_layout.addWidget(self.depthctl_param_ki_label,
2, 0, 1, 1, QtCore.Qt.AlignCenter)

        self.depthctl_param_ki_edit = QtWidgets.QLineEdit()
        self.main_layout.addWidget(self.depthctl_param_ki_edit,
2, 1, 1, 1, QtCore.Qt.AlignCenter)
        self.depthctl_param_ki_edit.setText('0.0')

        self.depthctl_param_kd_label = QtWidgets.QLabel('Kd')
        self.depthctl_param_kd_label.setFont(QtGui.QFont('Arial',
12))
        self.main_layout.addWidget(self.depthctl_param_kd_label,
3, 0, 1, 1, QtCore.Qt.AlignCenter)

        self.depthctl_param_kd_edit = QtWidgets.QLineEdit()
        self.main_layout.addWidget(self.depthctl_param_kd_edit,
3, 1, 1, 1, QtCore.Qt.AlignCenter)
        self.depthctl_param_kd_edit.setText('0.0')

        self.depthctl_writeparam_button =
QtWidgets.QPushButton('写入参数')

 self.main_layout.addWidget(self.depthctl_writeparam_button, 4,
0, 1, 2, QtCore.Qt.AlignCenter)
```

```
 self.depthctl_writeparam_button.setObjectName("SET_DEPTHCTL_PARAM")


    # 用来开启窗口的函数
    def handle_click(self):
        if not self.isVisible():
            self.show()

    # 用来关闭窗口的函数
    def handle_close(self):
        self.close()
```

子窗口界面如下图所示：



3.1.2 第二步

在boxfishhost.py文件中，添加一个按钮，并将我们写的子窗口和这个按钮关联起来：

首先，要引入我们写的子窗口：

```
import childwindows.depth_control_btn_win # 导入深度控制窗口
```

其次，在BoxFishWindow类的`init_console_panel()`函数中，初始化深度控制按钮：

```
# 创建深度控制按钮
self.open_depth_control_button = QtWidgets.QPushButton('深度控制')
# 创建按钮
self.advancedcc_layout.addWidget(self.open_depth_control_button,
3, 0, 1, 1, QtCore.Qt.AlignCenter) # 将按钮加入界面
```

其次，在BoxFishWindow类的`__init__()`的函数中，初始化子窗口类

```
## 深度控制子窗口
self.DCBW =
childwindows.depth_control_btn_win.DepthControlBtnWin() # 初始化子
窗口类
self.open_depth_control_button.clicked.connect(self.DCBW.handle_c
lick) # 将按钮与子窗口的开启函数关联起来
self.DCBW.depthctl_start_button.clicked.connect(self.console_butt
on_clicked) # 将子窗口中的按钮与console_button_clicked函数关联起来
self.DCBW.depthctl_stop_button.clicked.connect(self.console_butto
n_clicked) # 将子窗口中的按钮与console_button_clicked函数关联起来
self.DCBW.depthctl_writeparam_button.clicked.connect(self.console
_button_clicked) # 将子窗口中的按钮与console_button_clicked函数关联起来
self.close_signal.connect(self.DCBW.handle_close) # 将主窗口关闭信号
与子窗口关闭函数关联起来，这样只要主窗口关闭，子窗口也会关闭
```

这里需要注意的是，子窗口中的按钮可以和任意函数关联，不一定要是console_button_clicked函数。

但是console_button_clicked是用来发送RFLink指令的，并且这个按钮也是要发送RFLink消息的话，一般都与这个函数进行关联。

至于如何发送消息，比如如何把kp，ki，kd放进消息里，用户就可以在console_button_clicked函数中进一步设置

3.1.3 第三步

在rflink.py的Command中添加刚才定义的命令，即第一步中那三个按钮的名字：

SET_DEPTHCTL_START、SET_DEPTHCTL_STOP、SET_DEPTHCTL_PARAM

如下：

```
Command = Enum('Command',(\
    'SHAKING_HANDS',\
    ...
    'SET_DEPTHCTL_START',\
    'SET_DEPTHCTL_STOP',\
    'SET_DEPTHCTL_PARAM',\
    ...
    'LAST_COMMAND_FLAG'))
```

注意，如果按钮并不是与RFLink消息发送相关的，就可以不用做这一步了

注意，这里一定要在下位机的RFlink的Command列表中也同时加入这些命令

3.1.4 第四步

这一步，就是在console_button_clicked()函数中，添加我们想要实现的效果。

```
if *******
elif rflink.Command[sender_button.objectName()] is
rflink.Command.SET_DEPTHCTL_PARAM: # 关于SET_DEPTHCTL_PARAM的处理
    data = struct.pack('<f',
float(self.DCBW.depthctl_param_kp_edit.text())) + \
        struct.pack('<f',
float(self.DCBW.depthctl_param_ki_edit.text())) + \
        struct.pack('<f',
float(self.DCBW.depthctl_param_kd_edit.text())) # data就是我们要发送
的数据
```

另外，如果子窗口需要显示一些机器鱼传回来的数据，那么可以在newdata_comming_slot()修改，以获得想要的效果

3.2 添加RFLink命令的方法

比如要添加三条命令：SET_DEPTHCTL_START、SET_DEPTHCTL_STOP、SET_DEPTHCTL_PARAM

首先，在rflink.py的Command中添加刚才定义的命令，如下：

```
Command = Enum('Command',(\
    'SHAKING_HANDS',\
    ...
    'SET_DEPTHCTL_START',\
    'SET_DEPTHCTL_STOP',\
    'SET_DEPTHCTL_PARAM',\
    ...
    'LAST_COMMAND_FLAG'))
```

注意，如果按钮并不是与RFLink消息发送相关的，就可以不用做这一步了

注意，这里一定要在下位机的RFlink的Command列表中也同时加入这些命令

其次，如果需要接收该命令，那么就在newdata_comming_slot函数中，添加对应的一些语句：

```
if rflink.Command(command_id) is rflink.Command.SHAKING_HANDS:
    *******
elif ********
# 在函数的if语句中增加下面这条判断
elif rflink.Command(command_id) is
rflink.Command.SET_DEPTHCTL_START:
    # 添加你所需要处理的工作
```

3.3 RFLink发送命令和数据的方法

发送命令和数据有三步：

- 首先是，将要发送的数据转换成byte类型，例如

```
# 把浮点数转换为二进制表示
a = 1.1
data = struct.pack('<f',a)
# 把字符串转换为ascii码
b = '1.1'
data = b.encode('ascii')
# 需要注意的是，用户可以自己选择转换的方式，但是上位机和下位机的转换方式要对应，这样才能保证数据传输正确
# 例如，如果上位机上选择了字符串转换为ascii码，那么下位机就要从ascii转换回字符串
```

- 其次是，打包数据。利用RFLink_packdata函数，将指令和数据一起打包

```
datapack =
rftool.RFLink_packdata(rflink.Command['SET_DEPTHCTL_START'].va
lue, data)
```

- 最后是，利用串口发送数据

```
 # 数据发送
 with QtCore.QMutexLocker(ser_mutex):#需要锁住串口资源，避免发送过程
中，其他线程抢占资源
    try:
        send_sertool.write_cmd(datapack)
    except serial.serialutil.SerialException:
        self.statusBar().showMessage('串口未打开,无法发送')
```

- 总之这部分代码，可以参考console_button_clicked函数

3.4 处理接收到的RFLink消息的方法

在newdata_comming_slot函数中，添加对应的一些语句：

```
if rflink.Command(command_id) is rflink.Command.SHAKING_HANDS:
    *******
elif ********
# 在函数的if语句中增加下面这条判断
elif rflink.Command(command_id) is
rflink.Command.SET_DEPTHCTL_START:
    # 添加你所需要处理的工作
```

4 其它说明