PHAWD

用户手册

何祜宁

目录

1 引言 **1**

1.1 开发目的 1

1.2 软件简介 1

2 安装与编译 **2**

2.1 Windows 2

2.2 Linux 3

2.3 Python 包安装 3

3 界面与功能 **4**

3.1 主要界面 4

3.2 功能与设置 6

3.2.1 机器人名称设置 6

3.2.2 控制参数设置 6

3.2.3 控制参数保存与读取 8

3.2.4 波形参数个数 8

3.2.5 数据写入检测以及终端提示功能 8

3.2.6 波形显示功能 9

3.2.7 手柄测试功能 10

4 机器人端使用例程 **13**

4.1 C++ 13

4.1.1 共享内存 14

4.1.2 TCP/IP 16

4.2 Python 19

4.2.1 共享内存 19

4.2.2 TCP/IP 20

5 API Reference **22**

5.1 ParameterKind 22

5.2 ParameterValue 23

5.2.1 C++ . . . 23
5.2.2 Python . . . 23
5.3 Parameter . . . 24
5.3.1 C++ . . . 24
5.3.2 Python . . . 25
5.4 ParameterCollection . . . 27
5.4.1 C++ . . . 27
5.4.2 Python . . . 28
5.5 GamepadCommand . . . 28
5.5.1 C++ . . . 28
5.5.2 Python . . . 29
5.6 SharedParameters . . . 30
5.6.1 C++ . . . 30
5.6.2 Python . . . 32
5.6.3 C++ 使用案例 . . . 33
5.7 SocketFromPhawd . . . 34
5.7.1 C++ . . . 34
5.7.2 Python . . . 35
5.8 SocketToPhawd . . . 35
5.8.1 C++ . . . 35
5.8.2 Python . . . 36
5.9 SharedMemory . . . 37
5.9.1 C++ . . . 37
5.9.2 Python . . . 38
5.10 SocketConnect . . . 39
5.10.1 C++ . . . 39
5.10.2 Python . . . 40

6 声明 **41**

7 联系作者 **41**

8 待改进事项 **42**

1 引言

1.1 开发目的

PHAWD 面向领域为机器人仿真与控制，开发目的是简化机器人控制参数调试过程，控制工程师能及时看到不同控制参数对机器人运动状态的影响，避免不断修改参数重新编译控制程序，提高参数调试效率，同时帮助控制算法工程师记录最优参数，方便下次测试。

1.2 软件简介

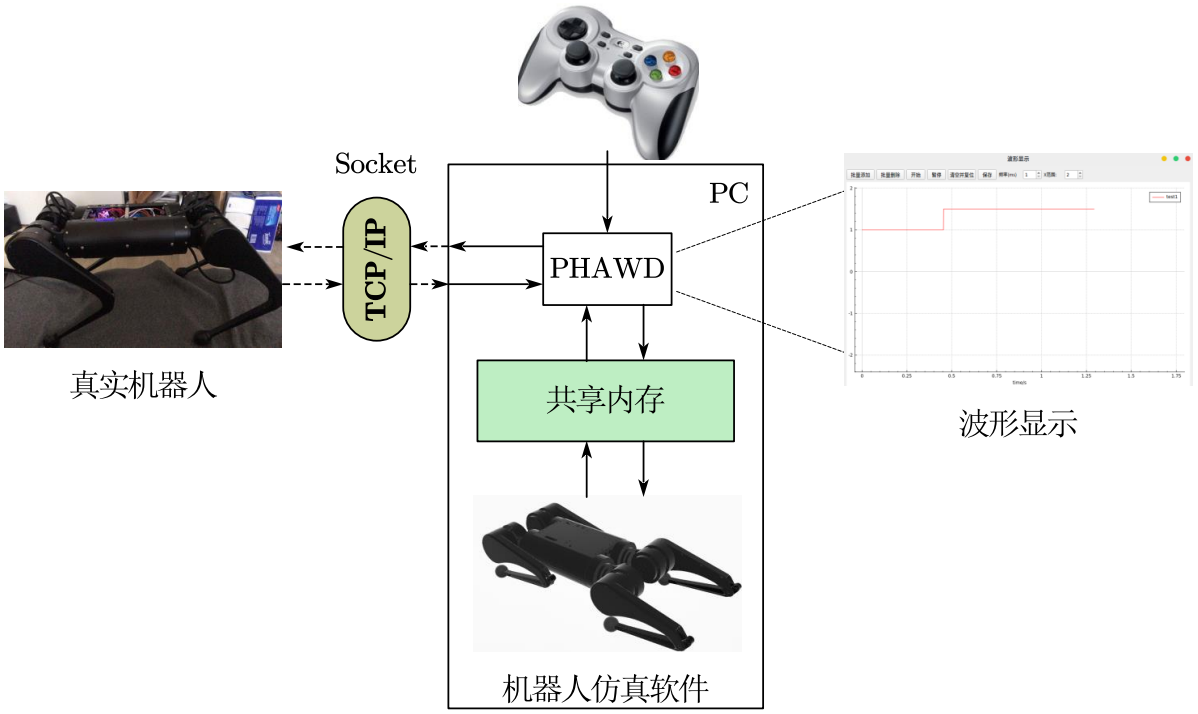

图 1.1 PHAWD 总体示意图

PHAWD 是与机器人仿真软件以及实物机器人控制器进行数据交换的上位机软件，如图1.1所示，使用简单，小巧稳定。其中 PHAWD 有两种通信方式供用户自行选择，分别是共享内存和 Socket 通信。

其中基于共享内存的方式，一般用于机器人仿真中，PHAWD 会开辟一块共享内存并将所有控制参数相关数据以及手柄数据放入其中，机器人控制程序不断从共享内存中读取更新的控制参数，同时又可以将机器人状态如欧拉角等数据写入共享内存，PHAWD 检测到写入后，用户可以选择绘制所需波形。

基于 Socket 通信的方式，一般用于实物机器人的调试，机器人控制器与 PHAWD 通过 TCP/IP 连接，参数设置完毕，点击确认后，PHAWD 进入监听状态，当 PHAWD 与机器人建立链接后，只有参数值或手柄更新才会发送数据给机器人。同时 PHAWD 不断读取机器人控制程序传来的状态信息用于波形显示，没读取到新数据也不会阻塞。

用户可将最优的一组参数保存至 yaml 文件，也可以将 yaml 文件中的参数配置直接读取到 PHAWD 中，便于反复测试。同时 PHAWD 还支持手柄测试，将手柄控制数据发送给机器人控制程序，以弥补部分机器人仿真器不支持手柄控制，以及实物机器人控制板手柄读取操作复杂等不足。

波形显示功能中，用户可以选择批量添加数据源，一次性添加一至四条曲线，同时用户也可以根据需要，对不需要的波形进行删除清空等，波形数据采样频率也可由用户设置。鼠标左键框选局部放大波形，右键拖动平移波形，滚轮可以整体缩放。

2 安装与编译

2.1 Windows

✓ 安装:

点击 链接 下载 phawd_setup_*.exe，并按步骤进行安装，注意尽量避免安装在系统盘中，避免没有权限创建共享内存，如果你更偏爱安装在系统盘中，请使用管理员身份运行 PHAWD。默认安装为英文界面，如果喜欢中文界面，可以在链接中下载 phawd_windows_zhCN.exe，将该文件重命名为 phawd.exe，并替换 PHAWD 安装目录 bin 文件夹下的同名文件。

✓ 编译:

注意修改 CMakeLists.txt 中 Qt 路径为你的 Qt 安装路径。使用 Qt5 自带的 MinGW 编译器，配置与下类似，此处不再赘述。

```
git --recursive https://github.com/HuNingHe/phawd.git
cd phawd
cmake -S . -B build -G "Visual Studio 16 2019" -A x64
# If you are using 2017 or older version of Visual Studio,
# using the below command instead:
# cmake -S . -B build -G "Visual Studio 1x 201x Win64"
cmake --build build --config Release
# if you want to install PHAWD:
```

```
# cmake --build build --config Release --target install
```

2.2 Linux

✓ Ubuntu 安装:

点击 链接 下载 phawd_*_amd64.deb，并按步骤进行安装即可。默认安装为英文界面，如果喜欢中文界面，可以在链接中下载 phawd_linux_zhCN，将该文件重命名为 phawd，并替换 PHAWD 安装目录 bin 文件夹下的同名文件，Linux 下位于/usr/local/bin。

✓ 编译:

注意修改 CMakeLists.txt 中 Qt 路径为你的 Qt 安装路径。

```
git --recursive https://github.com/HuNingHe/phawd.git
cd phawd
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build --config Release
# To install PHAWD:
# sudo cmake --build build --target install
#                or
# cd build
# sudo make install
```

2.3 Python 包安装

见 链接，安装如下：

```
pip install phawd
```

从源码安装如下：

```
git clone --recursive https://github.com/HuNingHe/phawd_py.git
pip install ./phawd_py
```

3 界面与功能

3.1 主要界面

PHAWD 初始界面如下图3.1所示，其中左上角有三个选项卡，分别是参数、波形和关于。点击“关于”选项卡将切换到关于页面，该页面包含 PHAWD 作者等相关信息，如下图3.2 所示。点击“波形”选项卡将打开一个新窗口，用于波形绘制，如下图3.3所示。

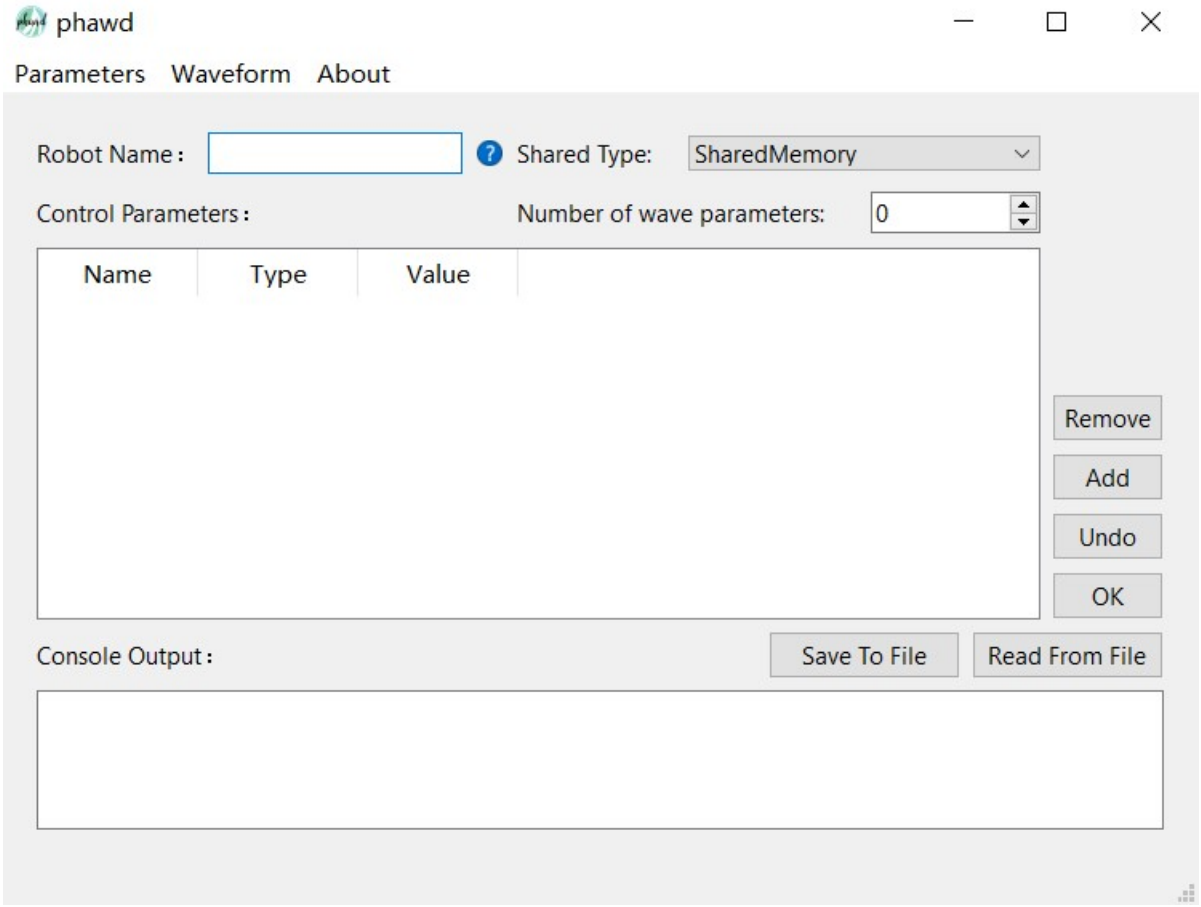

图 3.1 PHAWD 主界面

点击“参数”选项卡，将有两个选项，分别是参数设置和手柄测试，如下图3.4所示。其中点击“参数设置”回到初始界面，点击“手柄测试”进入手柄测试页面，会显示手柄连接状态与当前按键状态。

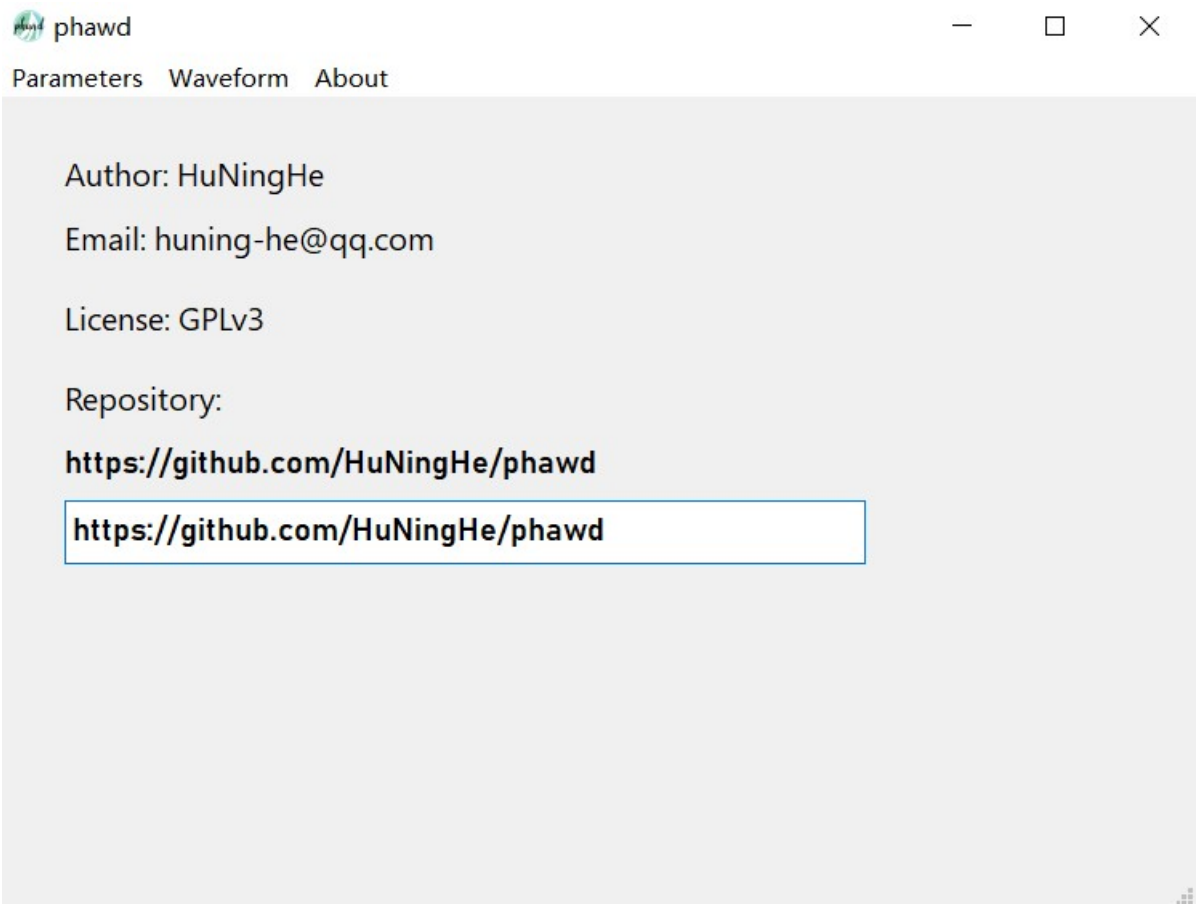

图 3.2 关于界面

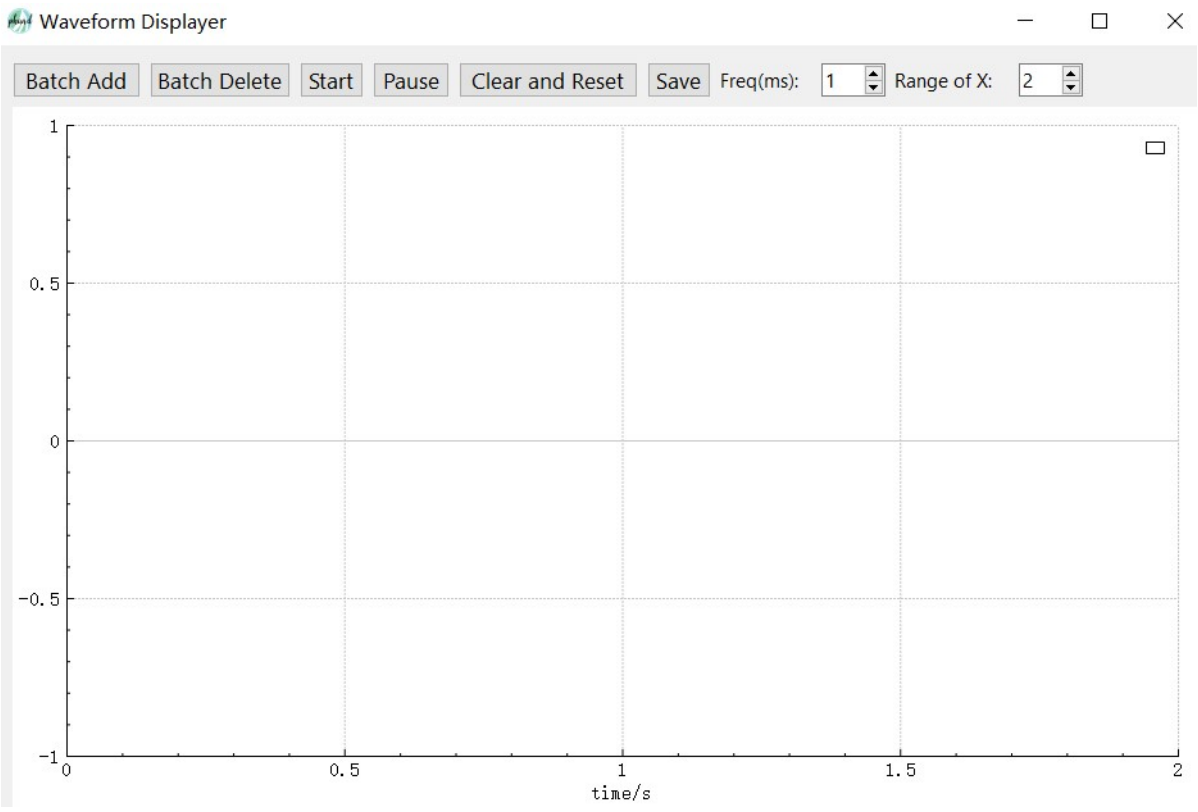

图 3.3 波形显示界面

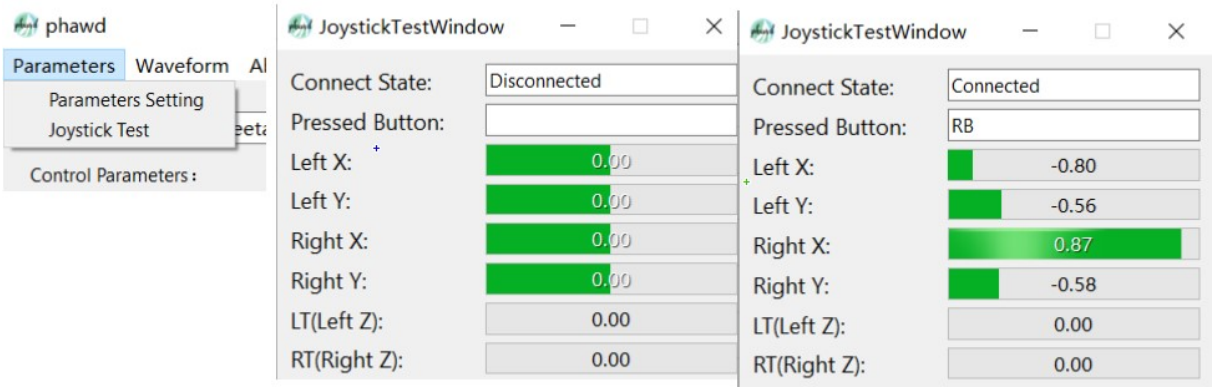

图 3.4 手柄测试

3.2 功能与设置

本节主要针对图3.1参数相关功能以及图3.3波形显示相关功能，对机器人名称的设置，控制参数设置 (添加、删除、清空、保存文件以及从文件读取)，波形参数个数的设置，共享方式的切换，终端提示以及波形显示以及手柄测试功能进行介绍。

3.2.1 机器人名称设置

机器人名称应该根据不同共享方式来设置，当用户使用默认共享方式即共享内存方式时，机器人名称最多 16 个字符，只能由字母开头，可包含数字下划线，如“MiniCheetah”，“Cheetah3”，“BD_Spot123”等。此名称即共享内存映射文件的名称，该文件一般生成在 Ubuntu 系统/dev/shm 文件夹下，Windows10 系统则生成在 phawd.exe 所在目录下。

若用户选择 Socket 通信，机器人名称便只能输入非 0 开头的至多六位整数，这个整数即 Socket 通信的端口号，用户设置完端口号后，机器人控制程序便能找到 PHAWD，建立 Socket 通信。机器人名称已有正则表达式验证，如果无法输入需确认输入是否规范。若不设置机器人名称，则无法点击确认，无法开辟共享内存，无法进入端口监听状态。

3.2.2 控制参数设置

一个参数包含三个信息: 参数名称，参数类型，参数值。其中参数名称最长 16 个字符，命名规则机器人名称设置规则。双击参数类型列下的单元格将有五种类型可选择: FLOAT、DOUBLE、S64、VEC3_FLOAT、VEC3_DOUBLE，如下图3.5所示。

其中 FLOAT 为单精度浮点数，DOUBLE 为双精度浮点数，S64 为 64 位有符号整数，VEC3_FLOAT 为三维 FLOAT 数组，VEC3_DOUBLE 为三维 DOUBLE 数组。注意: 如

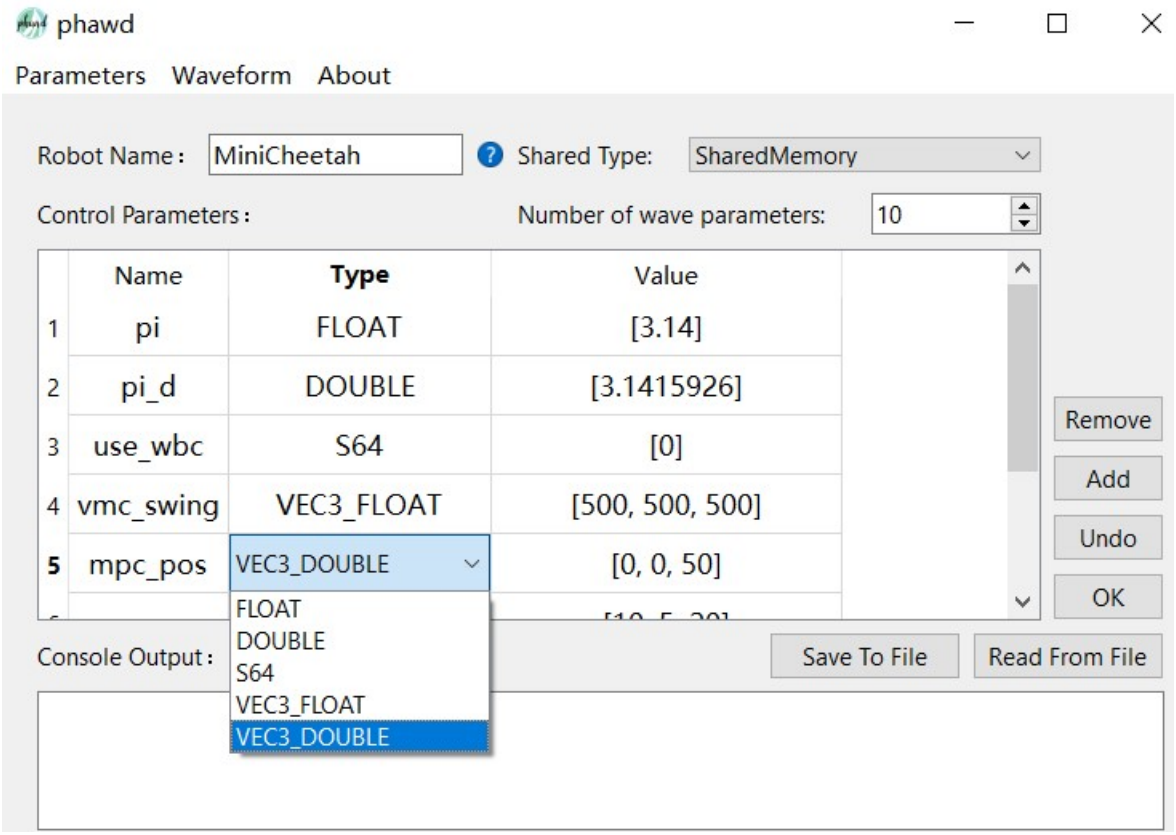

图 3.5 参数类型设置

果你使用 **Python** 编写机器人控制器，那么 **FLOAT** 和 **VEC3_FLOAT** 将无法使用。

参数值可以中括号“[”开头和“]”结束，也可没有中括号。对于 VEC3 数组类型，数组中的三个数必须以逗号分隔，如 [1.1, 2.2, 3.3]，其中逗号与数字间可以存在任意数量空格。**如果参数值和参数名称未按规范填写，将无法确认成功，若已确认成功，此时参数值修改不规范，则新值不会被写入共享内存也不会通过网络发送出去。**

其中参数的删除与添加，可以点击“添加”/“删除”按钮，也可以鼠标右键单击表格内部，将弹出菜单栏，如下图3.6，将有五个选项: Add Row，Remove Row，Clear All，Load From File，Save to File。其中 Add Row 功能与“添加”按钮相同，在当前选中行后添加一行，亦即添加一个参数，Remove Row 功能与“删除”按钮相同，删除当前选中行，Clear All 是清空所有参数。

机器人名称，共享方式，波形参数个数以及所有参数设置完成后，便可点击确认开辟共享内存或监听指定网络端口。**一旦点击确认，初始界面下就不能再修改参数类型、参数名称、共享内存名称、通信方式，Socket 通信端口，只能点击取消重新设置，此时只允许点击取消、保存参数至文件**，一定要先停止运行机器人程序，再点击取消。

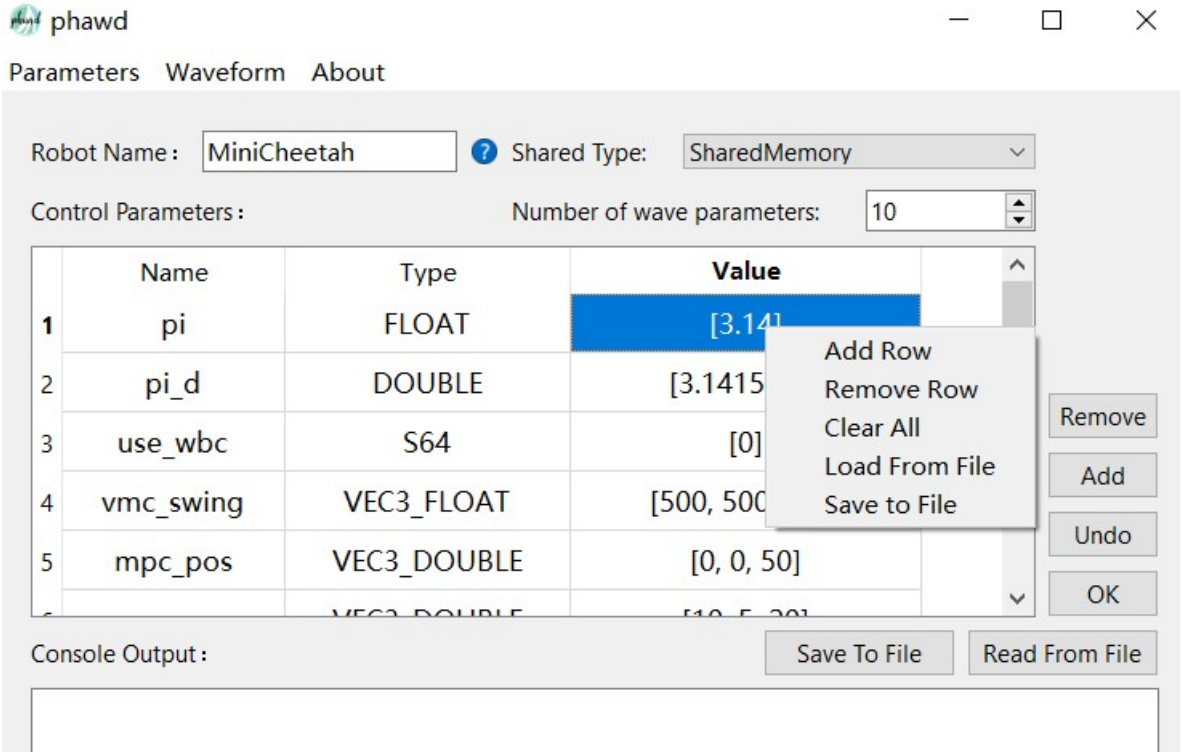

图 3.6 参数操作选项卡

3.2.3 控制参数保存与读取

如图3.6可见，鼠标右键菜单栏中还有保存至文件以及从文件读取参数，功能与相应按钮对应。设置好参数后，可以保存至 yaml 文件中，如下图3.7 所示，用户可以参考由 PHAWD 生成的 yaml 文件，自己编写 yaml 文件，如果不添加参数就保存，PHAWD 将自动生成一个模板 yaml 文件，也可参考图3.7的 yaml 文件进行编写。之后就可以直接从 yaml 读取控制参数，提高效率。

3.2.4 波形参数个数

波形参数个数需提前告知 PHAWD，以提前开辟好足够多的内存，存放波形数据，根据用户需要设置，最大为 29999。需要注意的是，如果没有添加控制参数，并且波形参数个数也为零，则无法点击确认开始通信。也可以不设置控制参数，只设置波形参数，将 PHAWD 当示波器使用。另外设置时，最好考虑电脑内存大小，避免内存不足。

3.2.5 数据写入检测以及终端提示功能

PHAWD 数据写入检测线程检测到机器人控制程序将共享内存中的标志位置 1 时，线程自动退出，并将所有波形参数名称交给波形显示窗口，后续可在波形显示窗口中选择相应数据源，显示波形。其中检测到数据后，会在终端输出打印如下图3.8所示的提示。

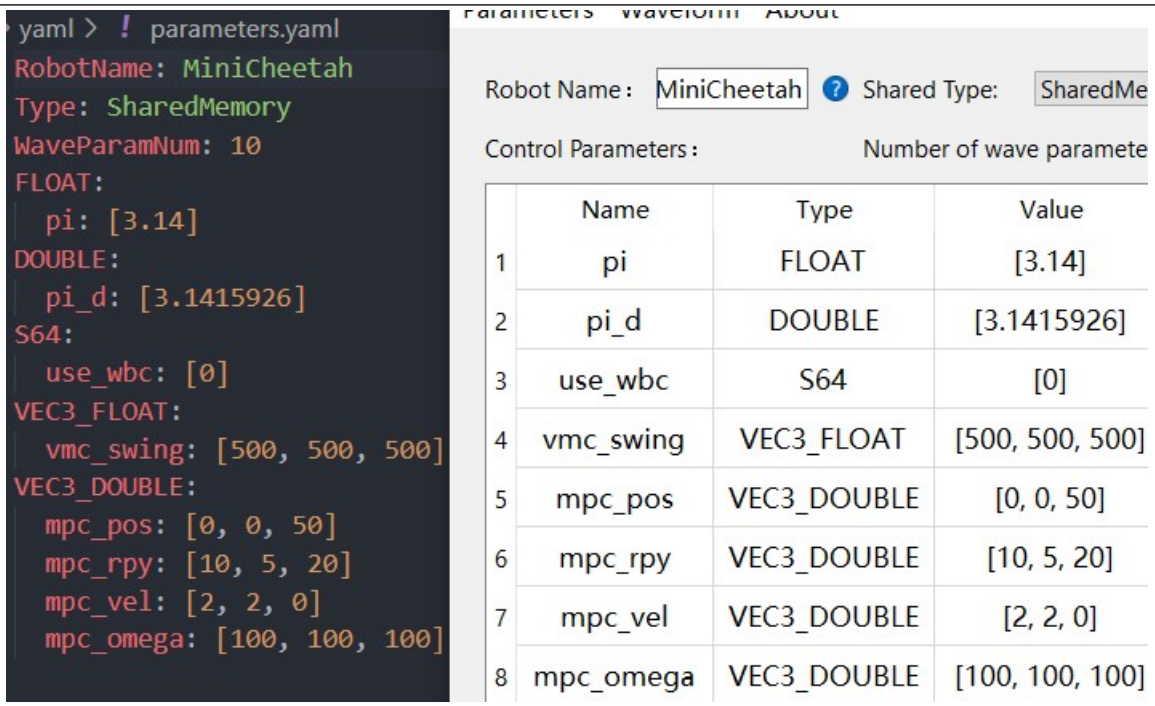

图 3.7 参数配置 YAML 文件

终端会输出很多提示，比如开辟共享内存后提示包含多少参数，占用多大内存。再如，当用户从文件读取参数时，可能存在部分类型参数不存在，或者机器人名称未设置，都将进行黑字提示。

若机器人名称未设置以及没有添加参数就点击确认或者保存至文件，亦或是由于权限不足开启共享内存失败，已有同名共享内存映射文件时将提示红色警告，用户需要观察终端的一些红色警告提示进行操作。比如去/dev/shm 文件夹下删除同名映射文件，或者当前 Socket 端口被系统占用等英文提示。

鼠标右键点击终端内部，有清空终端、复制以及全选选项卡，如图3.8。

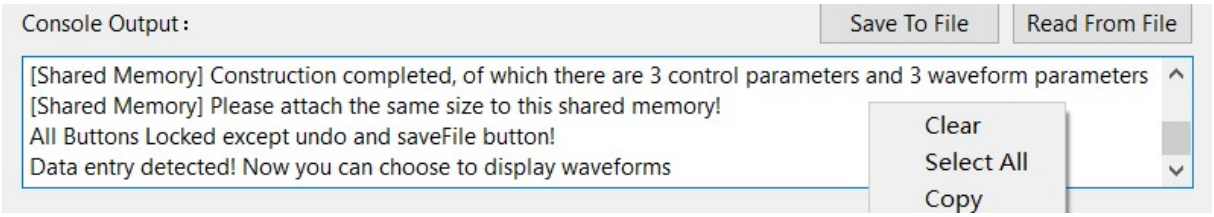

图 3.8 终端输出提示

3.2.6 波形显示功能

在检测到机器人控制程序的数据写入后，PHAWD 中可以选择添加所需要的波形。波形显示支持鼠标右键按下拖动平移，左键按下框选局部放大，同时支持鼠标滚轮对波形图进行整体缩放，效果如下图3.9和图3.10。点击保存按钮，可将当前画面保存为

“.png”格式图片，横轴范围可以自由设置，对应英文界面 Range of X。清空复位会一次性清除所有曲线，并且下次绘图从 0 时刻开始。

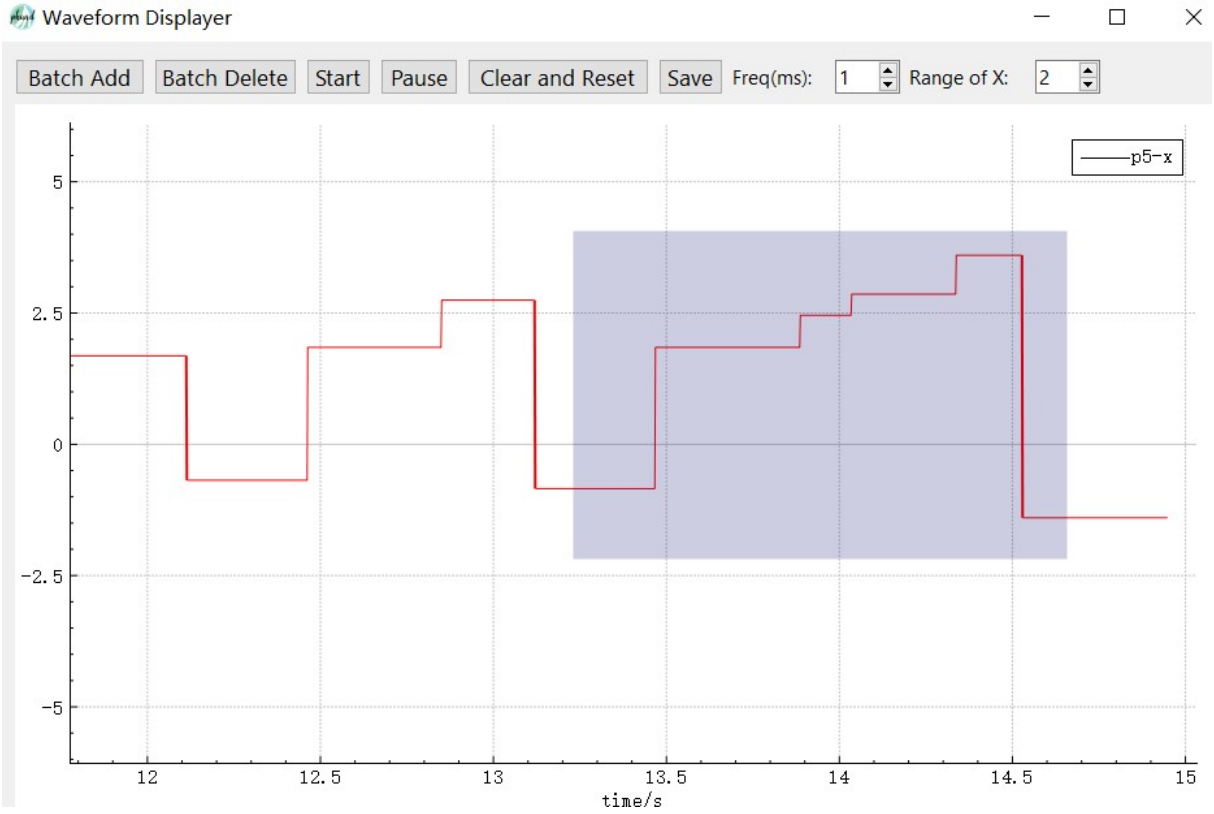

图 3.9 鼠标左键框选放大

用户点击批量添加可选择添加 0 至 4 条不同曲线，对于 VEC3 类型的数组参数，会自动在参数名后按顺序追加 -x/-y/-z，如图3.11所示。点击批量删除可以选择删除 0 至 4 条不同曲线，如图3.12所示。添加曲线时，可以选择曲线粗细，样式和颜色，其中粗细范围为 [1, 5]。样式包含 SolidLine(实线)、DashLine(虚线)、DotLine(点线)、DashDotLine(点虚线)、DashDotDotLine(点点虚线)，如下图3.13所示。 点击图3.11中的选择按钮，进入曲线颜色选择界面，如图3.14所示。**一个曲线只有颜色和数据源均选择完成才能添加该曲线**，一次可只添加一条，当然如果你本身不想添加曲线，只是误点，直接关闭该窗口即可 (删除同理)。点击开始便按指定频率开始采样数据绘制波形，修改频率只在波形开始绘制后有效，其中频率对应英文界面 Freq，点击暂停可以暂停曲线绘制。

3.2.7 手柄测试功能

作者使用的罗技 F710 手柄，将手柄调到 X 模式即可，Mode 灯不亮，如果 PHAWD 无法检测到你的手柄，不妨换一个 USB 接口试试，如果依旧不行可以使用 Windows 自带的手柄测试功能进行测试，确保手柄正常连接电脑。

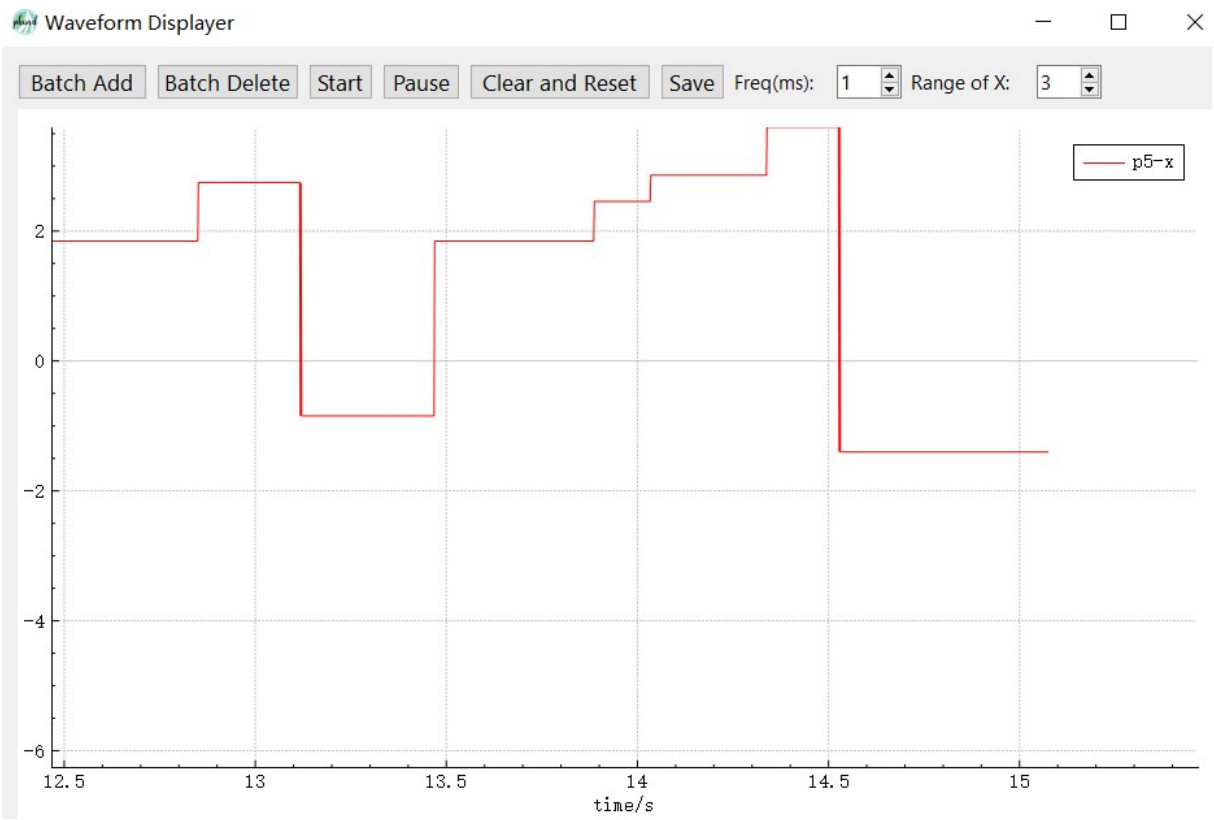

图 3.10 鼠标右键拖动平移

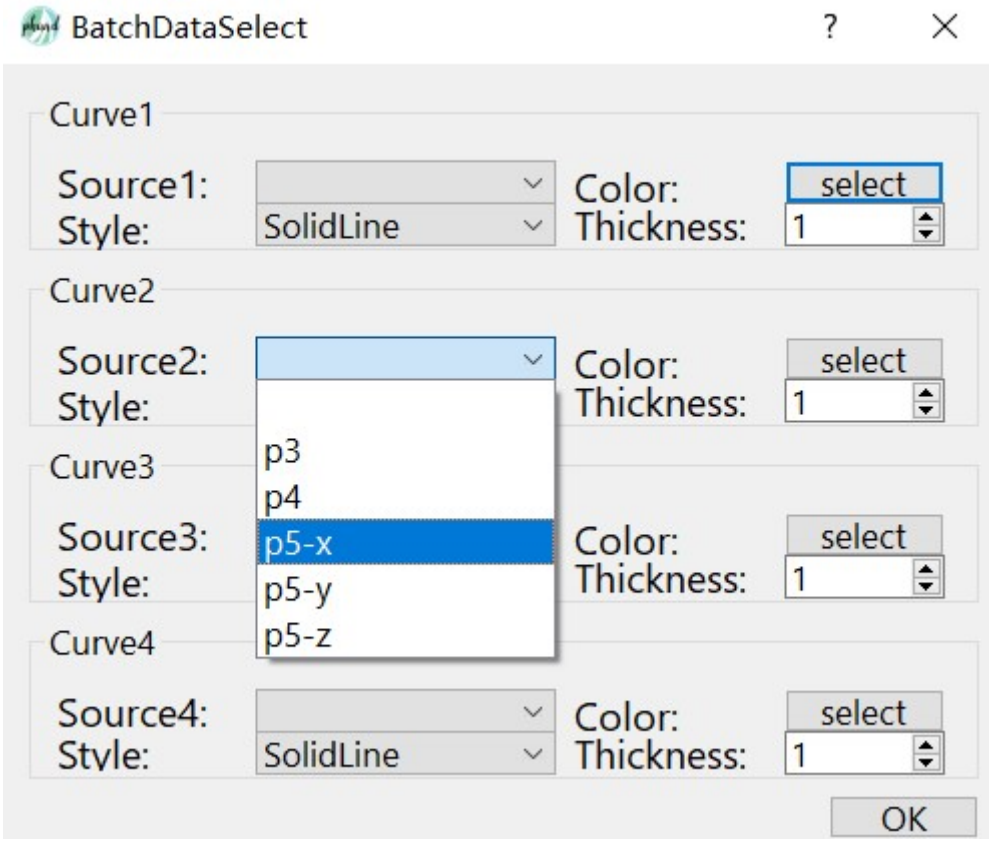

图 3.11 批量添加示意图

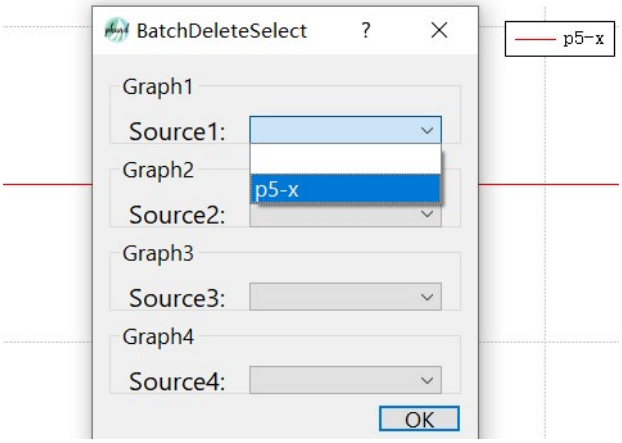

图 3.12 批量删除示意图

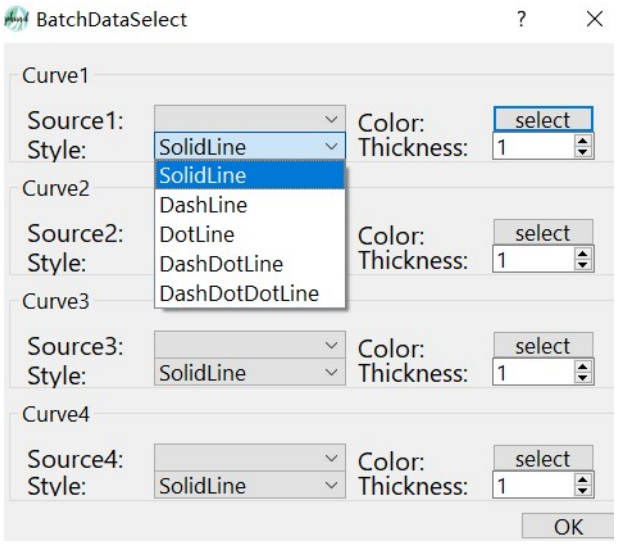

图 3.13 曲线样式选择

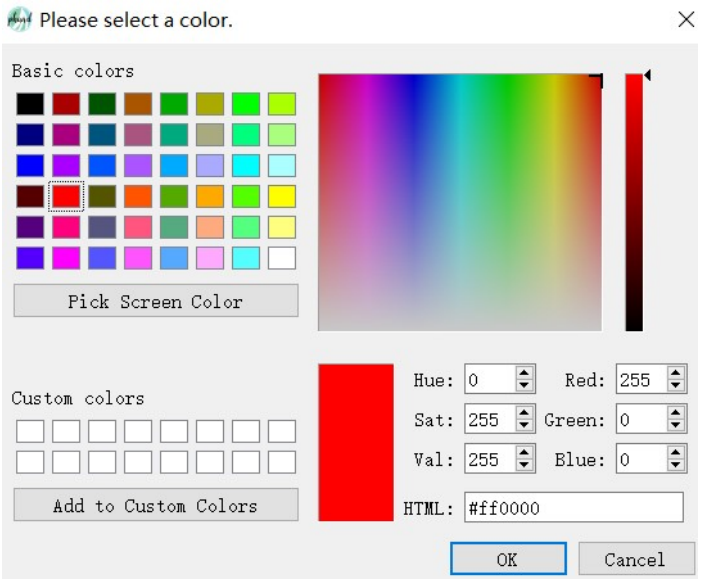

图 3.14 曲线颜色选择

4 机器人端使用例程

在 链接 中可以查看最新的例程，会陆续推出 Webots 和 MuJoCo 的例程。PHAWD 安装目录下 examples 目录中会包含以下示例代码。下面介绍两个基本的例程，包含共享内存方式和 TCP/IP 网络通信方式的机器人控制程序，其中手柄的数据可以在共享内存中的 SharedParameters 结构体中读取，也可以在 SocketFromPhawd 结构体中读取，此处不演示，可参考第5.6.3节。

4.1 C++

首先在 PHAWD 中读取如下 yaml 文件：

```yaml
RobotName: demo # Change To 5230, If you are running Socket demo
Type: SharedMemory # Change To Socket, If you are running Socket demo
WaveParamNum: 5
FLOAT:
  pf: [1.5]
DOUBLE:
  pd: [2]
S64:
  ps64: [123]
VEC3_FLOAT:
  pvec3f: [1, 2, 3]
VEC3_DOUBLE:
  pvec3d: [1, 2, 3]
```

两个例子均使用这个参数配置，使用 Socket 方式时，需要将 Type 对应内容修改为 Socket 以及 RobotName 对应内容改为 5230。两个例子使用共同的 CMakeLists.txt 如下，修改 CMAKE_PREFIX_PATH 为你的安装路径：

```cmake
cmake_minimum_required(VERSION 3.15)
project(demo)
# change to your path
list(APPEND CMAKE_PREFIX_PATH "D:/phawd/lib/cmake")
find_package(phawd REQUIRED)

set(CMAKE_CXX_STANDARD 11)
```

```
add_executable(demo main.cpp)

target_include_directories(demo PUBLIC ${PHAWD_INCLUDE_DIR})

target_link_libraries(demo phawd::phawd-shared)
#                       or
# target_link_libraries(demo ${PHAWD_SHARED_LIB})

# target_link_libraries(demo phawd::phawd-static)
#                       or
# target_link_libraries(demo ${PHAWD_STATIC_LIB})
# if you prefer to use static lib in windows,
# PHAWD_STATIC definition is mandatory:
# target_compile_definitions(test PRIVATE PHAWD_STATIC)
```

需要注意的是,在 Windows 端使用 phawd 静态库时,项目必须添加 PHAWD_STATIC 宏，其中共享内存对应的 main.cpp 如下：

4.1.1 共享内存

```
#include <vector>
#include <memory>
#include <iostream>
#include <phawd/phawd.h>

using namespace phawd;

int main() {
    bool usingPhawd = true;
    size_t waveParamNum = 5;
    size_t controlParamNum = 5;
    auto shm = std::make_shared<SharedMemory<SharedParameters>>();
    auto paramCollection = std::make_shared<ParameterCollection>();
    try {
        shm->attach("demo", sizeof(SharedParameters) +
        (waveParamNum + controlParamNum) * sizeof(Parameter));
    } catch(std::runtime_error &err) {
        printf("%s\n", err.what());
```

```
        printf("Attach shared memory error, don't use phawd here \n");
        usingPhawd = false;
    }
    if (usingPhawd){
        std::string nameList[5] = {"pf", "pd", "ps64",
                                   "pvec3f", "pvec3d"};
        controlParamNum = shm->get()->numControlParams;
        shm->get()->connected += 1;
        shm->get()->numWaveParams = waveParamNum;
        for (int i = 0; i < waveParamNum; ++i) {
            // Waveform parameters are appended after
            // Control parameters in SharedMemory
            shm->get()->parameters[controlParamNum +i].
                                    setName(nameList[i]);
        }
        for (int i = 0; i < controlParamNum; ++i){
            paramCollection->addParameter(&shm->get()->parameters[i]);
        }
    }

    size_t iter = 0;
    float pf;
    double pd;
    long int ps64;
    std::vector<float> pvec3f = {0, 0, 0};
    std::vector<double> pvec3d = {0, 0, 0};

    while (iter < 500000 && usingPhawd) {
        pf = paramCollection->lookup("pf").getFloat();
        pd = paramCollection->lookup("pd").getDouble();
        ps64 = paramCollection->lookup("ps64").getS64();
        pvec3f = paramCollection->lookup("pvec3f").getVec3f();
        pvec3d = paramCollection->lookup("pvec3d").getVec3d();
        shm->get()->parameters[controlParamNum + 0].setValue(pf);
        shm->get()->parameters[controlParamNum + 1].setValue(pd);
        shm->get()->parameters[controlParamNum + 2].setValue(ps64);
        shm->get()->parameters[controlParamNum + 3].setValue(pvec3f);
        shm->get()->parameters[controlParamNum + 4].setValue(pvec3d);
        if (iter % 20 == 0){
            std::cout << "pf:" << pf<< std::endl;
```

```
            std::cout << "pd:" << pd<< std::endl;
            std::cout << "ps64:" << ps64<< std::endl;
            std::cout << "pvec3f:" << pvec3f[0]
                      << " " << pvec3f[1] << " "
                      << pvec3f[2] << std::endl;
            std::cout << "pvec3d:" << pvec3d[0]
                      << " " << pvec3d[1] << " "
                      << pvec3d[2] << std::endl;
        }
        iter++;
    }
    shm->get()->connected -= 1;
    return 0;
}
```

运行步骤：

- 使用 PHAWD 读取上述 yaml 文件，并点击确认按钮；
- 编译该 C++ 项目 (Windows 下使用 VS 或 MinGW，Linux 下使用 g++)，运行该程序；
- 打开软件的波形显示界面，添加对应五条曲线；
- 在 PHAWD 参数设置界面任意修改参数值，此时对应曲线随之改变。

4.1.2 TCP/IP

此例程运行于 PHAWD 所在电脑，因此使用本机 IP 地址 127.0.0.1，用户可自行使用其他 (Ubuntu/Windows) 设备运行，设置好 IP 地址并将相应动态库 (phawd.dll 或 libphawd.so) 拷贝到可执行文件所在目录即可，对应的 main.cpp 如下：

```
#include <vector>
#include <memory>
#include <iostream>
#include "phawd/phawd.h"

using namespace phawd;
int main() {
    bool usingPhawd = true;
    size_t iter = 0;
```

```
    size_t waveParamNum = 5;
    size_t controlParamNum = 5;
    size_t sendSize = sizeof(SocketToPhawd) + waveParamNum *
                      sizeof(Parameter);
    size_t readSize = sizeof(SocketFromPhawd) + controlParamNum *
                      sizeof(Parameter);
    auto socket = std::make_shared<SocketConnect
                           <SocketToPhawd, SocketFromPhawd>>();
    try {
        socket->Init(sendSize, readSize);
        socket->connectToServer("127.0.0.1", 5230);
    } catch(std::runtime_error &err) {
        printf("%s\n", err.what());
        printf("Connect server error, don't use phawd here \n");
        usingPhawd = false;
    }

    if (usingPhawd) {
        std::string nameList[5] = {"pf", "pd", "ps64",
                                   "pvec3f", "pvec3d"};
        auto send_data = socket->getSend();
        send_data->numWaveParams = waveParamNum;
        for (int i = 0; i < waveParamNum; ++i) {
            send_data->parameters[i].setName(nameList[i]);
        }
        float f_value = 1.456;
        double d_value = 3.1516926;
        long s64_value = 12;
        std::vector<float> vec3f_value{1, 2 , 3};
        std::vector<double> vec3d_value{3, 2, 1};

        send_data->parameters[0].setValue(f_value);
        send_data->parameters[1].setValue(d_value);
        send_data->parameters[2].setValue(s64_value);
        send_data->parameters[3].setValue(vec3f_value);
        send_data->parameters[4].setValue(vec3d_value);
    }
    while (iter < 5000000 && usingPhawd){
        iter++;
        int read_count = socket->Read();
```

```
        if(read_count > 0) {
            float pf = socket->getRead()->parameters[0].getFloat();
            double pd = socket->getRead()->parameters[1].getDouble();
            long ps64 = socket->getRead()->parameters[2].getS64();
            std::vector<float> pvec3f = socket->getRead()->
                                        parameters[3].getVec3f();
            std::vector<double> pvec3d = socket->getRead()->
                                        parameters[4].getVec3d();

            auto send_data = socket->getSend();
            send_data->parameters[0].setValue(pf);
            send_data->parameters[1].setValue(pd);
            send_data->parameters[2].setValue(ps64);
            send_data->parameters[3].setValue(pvec3f);
            send_data->parameters[4].setValue(pvec3d);
            socket->Send();

            std::cout << "pf:" << pf<< std::endl;
            std::cout << "pd:" << pd<< std::endl;
            std::cout << "ps64:" << ps64<< std::endl;
            std::cout << "pvec3f:" << pvec3f[0] << " "
                      << pvec3f[1] << " "
                      << pvec3f[2] << std::endl;
            std::cout << "pvec3d:" << pvec3d[0] << " "
                      << pvec3d[1] << " "
                      << pvec3d[2] << std::endl;
        }
    }
    return 0;
}
```

运行步骤：

- 修改上述 yaml 文件并使用 PHAWD 读取，点击确认按钮；
- 编译该 C++ 项目 (Windows 下使用 VS 或 MinGW，Linux 下使用 g++)，运行该程序；
- 打开软件的波形显示界面，添加对应五条曲线；
- 在 PHAWD 参数设置界面任意修改参数值，此时对应曲线随之改变；

- 并且终端不停输出 PHAWD 发送过来的参数值。

4.2 Python

PHAWD 的 Python 包不支持 FLOAT 和 VEC3_FLOAT 类型参数。

与 C++ 例程类似，使用 PHAWD 读取如下 yaml 文件：

```yaml
RobotName: demo # Change To 5230, If you are running Socket demo
Type: SharedMemory # Change To Socket, If you are running Socket demo
WaveParamNum: 3
DOUBLE:
    pd: [2]
S64:
    ps64: [123]
VEC3_DOUBLE:
    pvec3d: [1, 2, 3]
```

4.2.1 共享内存

例程如下：

```python
from phawd import SharedMemory, SharedParameters
from phawd import ParameterCollection, Parameter

if __name__ == '__main__':
    p0 = Parameter("pw_d", 12.13)
    p1 = Parameter("pw_s64", 1213)
    p2 = Parameter("pw_vec3d", [-1.4, 2, 3])

    shm = SharedMemory()
    pc = ParameterCollection()

    p_size = Parameter.__sizeof__()
    sp_size = SharedParameters.__sizeof__()
    num_control_params = 3
    num_wave_params = 3
    shm_size = sp_size + (num_control_params + num_wave_params) * p_size
```

```
shm.attach("demo", shm_size)
sp = shm.get()
sp.setParameters([p0, p1, p2])
sp.connected += 1
ctr_params = sp.getParameters()
sp.collectParameters(pc)

run_iter = 0

while run_iter < 500000:
    run_iter += 1
    p0.setValue(pc.lookup("pd").getDouble())
    p1.setValue(pc.lookup("ps64").getS64())
    p2.setValue(pc.lookup("pvec3d").getVec3d())
    sp.setParameters([p0, p1, p2])

    if run_iter % 20:
        print(pc.lookup("ps64").getName(), ":")
        print(pc.lookup("ps64").getS64())
        print(pc.lookup("pd").getName(), ":")
        print(pc.lookup("pd").getDouble())
        print(pc.lookup("pvec3d").getName(), ":")
        print(pc.lookup("pvec3d").getVec3d())

sp.connected -= 1
```

运行步骤：

- 使用 PHAWD 读取上述 Python 示例对应 yaml 文件，点击确认按钮；
- 运行该 Python 程序；
- 打开软件的波形显示界面，添加对应三条曲线；
- 在软件参数设置界面任意修改参数值，此时对应曲线随之改变。

4.2.2 TCP/IP

此例程运行于 PHAWD 所在电脑，因此使用本机 IP 地址 127.0.0.1，用户可自行使用其他 (Ubuntu/Windows) 设备运行，例程如下：

```
from phawd import SocketToPhawd, SocketFromPhawd
from phawd import SocketConnect, Parameter

if __name__ == '__main__':
    p0 = Parameter("pw_d", 1.213)
    p1 = Parameter("pw_s64", 12)
    p2 = Parameter("pw_vec3d", [-1.4, 2, 3])

    send_params_num = 3
    read_params_num = 3
    send_size = SocketToPhawd.__sizeof__() + send_params_num *
                Parameter.__sizeof__()
    read_size = SocketFromPhawd.__sizeof__() + read_params_num *
                Parameter.__sizeof__()

    client = SocketConnect()

    client.init(send_size, read_size)
    client.connectToServer("127.0.0.1", 5230)
    run_iter = 0

    socket_to_phawd = client.getSend()
    socket_to_phawd.numWaveParams = 3
    socket_to_phawd.parameters = [p0, p1, p2]

    while run_iter < 5000000:
        run_iter += 1
        socket_to_phawd = client.getSend()

        ret = client.read()

        if ret > 0:
            socket_from_phawd = client.getRead()

            p0.setValue(socket_from_phawd.parameters[0].getDouble())
            p1.setValue(socket_from_phawd.parameters[1].getS64())
            p2.setValue(socket_from_phawd.parameters[2].getVec3d())
            socket_to_phawd.parameters = [p0, p1, p2]
            client.send()
```

```
        if run_iter % 20 == 0:
            print(socket_from_phawd.parameters[0].getName(), ":")
            print(socket_from_phawd.parameters[0].getDouble())
            print(socket_from_phawd.parameters[1].getName(), ":")
            print(socket_from_phawd.parameters[1].getS64())
            print(socket_from_phawd.parameters[2].getName(), ":")
            print(socket_from_phawd.parameters[2].getVec3d())
```

运行步骤：

- 修改上述 Python 示例对应 yaml 文件并使用 PHAWD 读取，点击确认按钮；
- 运行该 Python 程序；
- 打开软件的波形显示界面，添加对应三条曲线；
- 在 PHAWD 参数设置界面任意修改参数值，此时对应曲线随之改变；
- 终端输出控制参数值信息。

5 API Reference

此部分包含 PHAWD 的 API 介绍，将介绍相关函数参数及作用，可在 Wiki中下载最新文档。

5.1 ParameterKind

用于管理参数类型，Python 中不支持 FLOAT 和 VEC3_FLOAT，其余与 C++ 相同。

```
enum class ParameterKind: unsigned short int {
    FLOAT = 0,
    DOUBLE = 1,
    S64 = 2,
    VEC3_FLOAT = 3,
    VEC3_DOUBLE = 4
};
```

```
// usage in python:
from phawd import ParameterKind
ParameterKind.DOUBLE
// usage in cpp
ParameterKind::DOUBLE
```

getParameterKindFromString 仅用于 C++ 中，如将字符串“DOUBLE”转为 ParameterKind::DOUBLE，ParameterKindToString 仅用于 C++ 中，如将 ParameterKind::DOUBLE 转为字符串“DOUBLE”，由于无法 cout 打印 ParameterKind 对象，可使用 ParameterKindToString 将该参数类型转为字符串输出，Python 下能直接 print 输出。

```
ParameterKind getParameterKindFromString(const std::string& str);
std::string ParameterKindToString(ParameterKind kind);
// print ParameterKind
// std::cout << ParameterKindToString(ParameterKind::DOUBLE) << std::endl;
```

5.2 ParameterValue

存储参数值，确保每个参数占用内存相同。

5.2.1 C++

```
union ParameterValue {
    float f; // deprecated in python
    double d;
    long int i;
    float vec3f[3];  // deprecated in python
    double vec3d[3];
    // Constructor of ParameterValue, set all values to zero
    ParameterValue();
    void init(); // set all values to zero
};
```

5.2.2 Python

d，i，vec3d 在 Python 中直接当属性进行赋值和读取操作，，对应 setter 不再列出。

```python
from phawd import ParameterValue
class ParameterValue():
    @property
    def d(self):
    @property
    def i(self):
    @property
    def vec3d(self):
    def __init__(self):
```

5.3 Parameter

存放于第5.6节、第5.7节和第5.8节的 SharedParameters、SocketFromPhawd 和 SocketToPhawd 柔性数组中，存储参数信息。

5.3.1 C++

```cpp
class Parameter {
public:
    // defaut constructor: call ParameterValue::init()
    // and set parameter name to null, and set ParameterKind::DOUBLE
    Parameter();
    Parameter(const Parameter& parameter);
    Parameter(Parameter&& parameter) noexcept;
    Parameter &operator=(const Parameter &p);
    Parameter &operator=(Parameter &&p) noexcept;

    Parameter(const std::string& name, ParameterKind &kind);
    Parameter(const std::string& name,
              ParameterKind &kind,
              ParameterValue &value);
    Parameter(const std::string &name, float value);
    Parameter(const std::string &name, double value);
    Parameter(const std::string &name, long int value);
    Parameter(const std::string &name, const double* value);
    Parameter(const std::string &name, const float* value);
```

```
    explicit Parameter(const std::string &name,
                       const std::vector<double>& value);
    explicit Parameter(const std::string &name,
                       const std::vector<float>& value);

    bool setName(const std::string& name);
    void setValue(float value);
    void setValue(double value);
    void setValue(long int value);
    void setValue(const double* value);
    void setValue(const float* value);
    void setValue(const std::vector<float>& value);
    void setValue(const std::vector<double>& value);
    void setValue(ParameterKind kind, const ParameterValue& value);
    void setValueKind(ParameterKind kind);

    ParameterValue getValue(ParameterKind kind);
    ParameterKind getValueKind();
    std::string getName();
    float getFloat();
    double getDouble();
    long int getS64();
    double getFromVec3dByIndex(int idx);
    float getFromVec3fByIndex(int idx);

    std::vector<double> getVec3d();
    std::vector<float> getVec3f();

    bool& isSet(); // check if ParameterName and ParameterKind are set
    void set(bool set);
};
```

5.3.2 Python

使用案例见第4.2.1节。

```
from phawd import Parameter
class Parameter():
```

```
    def getDouble(self) -> float:
    def getName(self) -> str:
    def getS64(self) -> int:
    def getValue(self, kind:ParameterKind) -> ParameterValue:
    def getValueKind(self) -> ParameterKind:
    def getVec3d(self) -> numpy.ndarray[numpy.float64]:
    # Check if ParameterName and ParameterKind are set
    def isSet(self) -> bool:

    def setName(self, name): # set parameter name
    def setValue(self, *args, **kwargs):
        """
        Overloaded function.

        1. setValue(self, value: numpy.ndarray[numpy.float64])
        2. setValue(self, value: int)
        3. setValue(self, value: float)
        4. setValue(self, kind: ParameterKind, value: ParameterValue)
        """

    def setValueKind(self, kind: ParameterKind):

    def __init__(self, *args, **kwargs):
        """
        Overloaded function.

        1. __init__(self)
        2. __init__(self, name: str, kind: ParameterKind)
        3. __init__(self, name: str,
                        kind: ParameterKind,
                        value: ParameterValue)
        4. __init__(self, name: str, value: float)
        5. __init__(self, name: str, value: int)
        6. __init__(self, name: str,
                        value: numpy.ndarray[numpy.float64])
        """

    def __sizeof__(self) -> int:
        """return sizeof(Parameter)"""
```

5.4 ParameterCollection

5.4.1 C++

用法可参考第4.1.1节中的例程。

```
class ParameterCollection {
public:
    explicit ParameterCollection(std::string name = "");
    /*!
     * Use this to add a parameter for the first time in the collection
     * Throws exception if you try to add the same parameter twice.
     */
    void addParameter(Parameter *param);

    /*!
     * Lookup a control parameter by its name.
     * This does not modify the set field of the control parameter!
     *
     * Throws exception if parameter wasn't found
     */
    Parameter &lookup(const std::string &name);

    //!< check if all the control parameters initialized?
    bool checkIfAllSet();

    /*!
     * Mark all parameters as not set
     */
    void clearAllSet();

    /*!
     * Remove all parameters
     */
    void clearAllParameters();
};
```

5.4.2 Python

用法可参考第4.2.1节中的例程。

```
from phawd import ParameterCollection
class ParameterCollection():
    """
        Add one parameter to collection
    """
    def addParameter(self, param :Parameter):

    """
        Check whether all parameter values in the collection are set
    """
    def checkIfAllSet(self):

    """
        delete all parameters in collection
    """
    def clearAllParameters(self):

    """
        Clear set of all parameters
    """
    def clearAllSet(self):
    """
        Find a parameter in collection
    """
    def lookup(self, name: str) -> Parameter:

    def __init__(self, name=''):
```

5.5 GamepadCommand

5.5.1 C++

```
struct GamepadCommand {
    bool down, left, up, right, LB, RB,
```

```
        back, start, A, B, X, Y;

    double axisLeftXY[2], axisRightXY[2];
    double LT, RT;
    // set all bool variable to false and double variable to 0
    void init();
    // call init()
    GamepadCommand();
};
```

5.5.2 Python

Python 中可以直接当属性进行赋值和读取操作，对应 setter 不再列出。

```
from phawd import GamepadCommand

class GamepadCommand():
    def init(self):
    def __init__(self):

    @property
    def A(self):
    @property
    def B(self):
    @property
    def X(self):
    @property
    def Y(self):
    @property
    def axisLeftX(self):
    @property
    def axisLeftY(self):
    @property
    def axisRightX(self):
    @property
    def axisRightY(self):

    @property
```

```
    def BACK(self):
    @property
    def START(self):

    @property
    def UP(self):
    @property
    def DOWN(self):
    @property
    def LEFT(self):
    @property
    def RIGHT(self):

    @property
    def LT(self):
    @property
    def RT(self):
    @property
    def DOWN(self):
    @property
    def LB(self):
    @property
    def RB(self):
```

5.6 SharedParameters

存放于共享内存中的数据结构体，一般由 SharedMemory 对象直接创建，无需用户创建，用户只需了解如何通过 SharedParameters 对共享内存进行数据读写即可。

5.6.1 C++

```
class SharedParameters {
public:
    // Number of connected objects,
    // and whenever an object is connected,
    // the value of "connected" should manually increment 1
    int connected;
```

```
    size_t numControlParams;            // Number of control parameters
    size_t numWaveParams;               // Number of waveform parameters
    phawd::GamepadCommand gameCommand;  // Commands from joystick
    // (control parameters) index range in [0, numControlParams) and
    // (waveform parameters) index range in
    // [numControlParams, numControlParams + numWaveParams)
    Parameter parameters[];
private:
    SharedParameters();

    SharedParameters(const SharedParameters &p);

    SharedParameters(SharedParameters &&p) noexcept;
public:
    SharedParameters& operator=(const SharedParameters &p);

    SharedParameters& operator=(SharedParameters &&p) noexcept;

    void collectParameters(ParameterCollection *pc);

    static SharedParameters* create(int num_control_params,
                                    int num_wave_params);
    static void destroy(SharedParameters* p);
};
```

说明：

- SharedParameters 对象只能创建在堆区，由 create 创建，并手动通过 destroy 函数销毁，其中 create 函数中的参数 num_control_params 和 num_wave_params 用于确定控制参数与波形参数个数，这两个函数基本用不到；
- collectParameters 函数用于将 SharedParameters 中的所有参数添加到 ParameterCollection 中，见第4.2.1节中的例程；
- connected 属性用于指明连接共享内存的进程数量，每次连接共享内存时，需手动将该值加一，以保证 PHAWD 能检测到数据写入，见第4.1.1节中的例程；
- numControlParams 和 numWaveParams 存储控制参数个数与波形参数个数，由 PHAWD 自动写入，用户只能读取，不要随意修改；

- 可直接获取 parameters 数组，读写参数；
- gameCommand 存储手柄数据。

5.6.2 Python

使用例程见第4.2.1节。

```
class SharedParameters():
    def collectParameters(self,parameter_collection:ParameterCollection):
    """
        create instance of SharedParameters
    """
    def create(self, num_control_params: int, num_wave_params: int)->
                                                        SharedParameters:
    """
        destroy instance of SharedParameters
    """
    def destroy(self, instance: SharedParameters):

    def getParameters(self, get_control_params=True)->list:

    def setParameters(self, param_list:list, set_control_params=False):

    def __init__(self, *args, **kwargs):
    """
        return sizeof(SharedParameters)
    """
    def __sizeof__(self)->int:

    # check in cpp vesion
    @property
    def connected(self):
    @property
    def numControlParams(self):
    @property
    def numWaveParams(self):
    @property
    def gamepadCommand(self):
```

说明：

- SharedParameters 对象只能创建在堆区，由 create 创建，并手动通过 destroy 函数销毁，其中 create 函数中的参数 num_control_params 和 num_wave_params 用于确定控制参数与波形参数个数，这两个函数基本用不到；
- collectParameters 函数用于将 SharedParameters 中的所有参数添加到 ParameterCollection 中，见第4.2.1节中的例程；
- connected 属性用于指明连接共享内存的进程数量，每次连接共享内存时，需手动将该值加一，以保证 PHAWD 能检测到数据写入，见第4.1.1节中的例程；
- numControlParams 和 numWaveParams 存储控制参数个数与波形参数个数，由 PHAWD 自动写入，用户只能读取，不要随意修改；
- gamepadCommand 存储手柄数据；
- 与 C++ 不同在于 Python 不能直接操作 parameters 数组，因此使用 setParameters 函数设置参数，其中其参数 param_list 是一个 Parameter 列表，参数 set_control_params 用于指定设置控制参数还是波形参数，默认为 False 即设置波形参数，见第4.2.1节中的例程；
- getParameters 函数用于获取参数，其中返回值是一个 Parameter 列表，其唯一参数用于指定获取控制参数还是波形参数，默认为 True 即获取控制参数，见第4.2.1节中的例程；
- __sizeof__ 函数用于获取该类型数据占用内存，方便设置 SocketConnect 发送数据的大小。

5.6.3 C++ 使用案例

存放于共享内存中，一般通过如下方式获取该结构体的指针或对象：

```cpp
#include<phawd/phawd.h>
using namespace phawd;
int main(){
    auto shm = SharedMemory<SharedParameters>();
    int numControlParams = 3;
    int numWaveParams = 3;
```

```
    size_t shm_size = sizeof(SharedParameters) +
           (numControlParams + numWaveParams) * sizeof(Parameter);
    shm.attach("shm_name", shm_size);
    auto *sharedParameters = shm.get();
    // or using:
    // auto sharedParameters = shm();
    // printf("%d", sharedParameters.gameCommand.x);
    printf("%d", sharedParameters->gameCommand.x); // get joystick data
    return 0;
}
```

5.7 SocketFromPhawd

通过 TCP/IP 从 PHAWD 发送过来消息对应的数据结构，一般由 SocketConnect 对象直接创建，无需用户创建，用户只需了解如何读取数据即可，具体使用例程见第4.1.2节和第4.2.2节。

5.7.1 C++

```
class SocketFromPhawd {
public:
    size_t numControlParams;
    GamepadCommand gameCommand;
    Parameter parameters[];
private:
    SocketFromPhawd();
    SocketFromPhawd(const SocketFromPhawd &p);
    SocketFromPhawd(SocketFromPhawd &&p) noexcept;
public:
    SocketFromPhawd& operator=(const SocketFromPhawd &p);
    SocketFromPhawd& operator=(SocketFromPhawd &&p) noexcept;

    static SocketFromPhawd* create(int num_params);

    static void destroy(SocketFromPhawd* p);
```

```
    void collectParameters(ParameterCollection *pc);
};
```

相关 API 操作与 SharedParameters 类似，具体可参考第5.6节中 C++ 部分的说明。

5.7.2 Python

```
class SocketFromPhawd():
    def collectParameters(self, parameter_collection:ParameterCollection):

    def create(self, num_params):

    def destroy(self, instance):

    def __init__(self, *args, **kwargs):

    def __sizeof__(self)->int:
    @property
    def numControlParams(self):
    @property
    def parameters(self):
    @property
    def gamepadCommand(self):
```

相关 API 操作与 SharedParameters 类似，具体可参考第5.6节中的 Python 部分说明，此处可直接通过 parameters 直接进行读取，该 list 只包含控制参数，见第4.2.2节例程。

5.8 SocketToPhawd

通过 TCP/IP 发送给 PHAWD 消息对应的数据结构，一般由 SocketConnect 对象直接创建，无需用户创建，用户只需了解如何写入数据即可，具体使用例程见第4.1.2节和第4.2.2节。

5.8.1 C++

```
class SocketToPhawd {
public:
    size_t numWaveParams;
    Parameter parameters[];
private:
    SocketToPhawd();
    SocketToPhawd(const SocketToPhawd &p);
    SocketToPhawd(SocketToPhawd &&p) noexcept;

public:
    SocketToPhawd& operator=(const SocketToPhawd &p);
    SocketToPhawd& operator=(SocketToPhawd &&p) noexcept;

    static SocketToPhawd* create(int num_params);

    static void destroy(SocketToPhawd* p);

    void collectParameters(ParameterCollection *pc);
};
```

相关 API 操作与 SharedParameters 类似，具体可参考第5.6节中 C++ 部分的说明。

5.8.2 Python

```
class SocketToPhawd():
    def collectParameters(self, parameter_collection:ParameterCollection):

    def create(self, num_params):

    def destroy(self, instance):

    def __init__(self, *args, **kwargs):

    def __sizeof__(self)->int:
    @property
    def numWaveParams(self):
    @property
    def parameters(self):
```

相关 API 操作与 SharedParameters 类似，具体可参考第5.6节中的 Python 部分说明，此处可直接通过 parameters 直接进行写入，见第4.2.2节例程。

5.9 SharedMemory

在 C++ 中是一个模板类，用于创建和连接共享内存，但仅支持 SharedParameters 类型。

5.9.1 C++

```cpp
template<typename T>
class SharedMemory{
public:
    SharedMemory() = default;
    ~SharedMemory();

    void createNew(const std::string &name, size_t size,
                   bool allowOverwrite = true);

    void attach(const std::string &name, size_t size);

    void closeNew();

    void detach();

    T *get();

    T &operator()();
};
```

说明：

- createNew 函数用于创建共享内存，其中 name 参数是共享内存名称，参数 size 指定共享内存大小，参数 allowOverwrite 用于确定是否允许覆盖重写内存映射文件；
- attach 函数用于连接共享内存，其中 name 参数是连接共享内存的名称，参数 size 为连接共享内存的大小，见第4.1.1节中的例程；
- closeNew 用于关闭 createNew 函数创建的共享内存；

- detach 用于断开共享内存的连接；
- get 函数用于获取 SharedParameters 指针；
- SharedMemory 仿函数用于获取 SharedParameters 引用。

5.9.2 Python

```
class SharedMemory():
    def attach(self, name: str, size: int):

    def closeNew(self):

    def createNew(self, name:str, size:int, allowOverwrite:bool=True):

    def detach(self):

    def get(self)->SharedParameters:

    def __call__(self)->SharedParameters:

    def __init__(self):
```

说明：

- createNew 函数用于创建共享内存，其中 name 参数是共享内存名称，参数 size 指定共享内存大小，参数 allowOverwrite 用于确定是否允许覆盖重写内存映射文件；
- attach 函数用于连接共享内存，其中 name 参数是连接共享内存的名称，参数 size 为连接共享内存的大小，见第4.1.1节中的例程；
- closeNew 用于关闭 createNew 函数创建的共享内存；
- detach 用于断开共享内存的连接；
- get 函数用于获取 SharedParameters 对象；
- SharedMemory 仿函数用于获取 SharedParameters 对象。

5.10 SocketConnect

在 C++ 中是一个模板类，用于 TCP/IP 连接，发送数据类型为 SocketToPhawd 类型，读取数据类型为 SocketFromPhawd 类型。

5.10.1 C++

```
template<typename SendData, typename ReadData>
class SocketConnect {
public:
    SocketConnect();
    ~SocketConnect();

    void Init(size_t sendSize, size_t readSize, bool is_server = false);

    void connectToServer(const std::string& serverIP, unsigned short port,
                         long int milliseconds = 30);

    void listenToClient(unsigned short port, int listenQueueLength = 2,
                        long int milliseconds = 60);

    int Send(bool verbose = false);

    int Read(bool verbose = false);

    void Close();

    SendData *getSend();
    ReadData *getRead();
};
```

说明：

- Init 函数用于初始化 SocketConnect 对象，包括确定发送与接收消息的大小以及该进程是否作为服务端，其中发送与接收数据大小的设置见第4.1.2节例程；
- connectToServer 函数用于客户端请求连接服务端，其中第一参数是服务端的 IP 地址，第二个参数为端口号，端口号一般设置于 PHAWD 中的机器人名称，第三个参数为等待时间，超过这个时间没连接上自动取消，见第4.1.2节中的例程；

- listenToClient 函数用于服务端监听客户端的连接，第一个参数为监听端口，第二个参数为监听队列长度，第三个参数为等待时间，超过一定时间没有客户端请求连接则自动取消；
- Send 用于发送数据，若发送成功则返回大于 0 的值；
- Read 函数读取数据，若读取成功则返回大于 0 的值；
- Close 函数用于关闭连接；
- getSend 函数用于获取待发送数据的 SocketToPhawd 指针,以设置发送数据,见第4.1.2节例程；
- getRead 函数用于获取读取数据的 SocketFromPhawd 指针，以获取读取到的数据，见第4.1.2节例程。

5.10.2 Python

```
class SocketConnect():
    def close(self):

    def connectToServer(self, server_ip:str, port:int, milliseconds:=30):

    def getRead(self)->SocketFromPhawd:

    def getSend(self)->SocketToPhawd:

    def init(self, send_size:int, read_size:int, is_server=False):

    def listenToClient(self, port:int, listenQueueLength=2,
                       milliseconds=60):

    def read(self, verbose=False)->int:

    def send(self, verbose=False)->int:

    def __init__(self):
```

说明：

- init 函数用于初始化 SocketConnect 对象，包括确定发送与接收消息的大小以及该进程是否作为服务端，其中发送与接收数据大小的设置见第4.2.2节例程；
- connectToServer 函数用于客户端请求连接服务端，其中第一参数是服务端的 IP 地址，第二个参数为端口号，端口号一般设置于 PHAWD 中的机器人名称，第三个参数为等待时间，超过这个时间没连接上自动取消，见第4.2.2节中的例程；
- listenToClient 函数用于服务端监听客户端的连接，第一个参数为监听端口，第二个参数为监听队列长度，第三个参数为等待时间，超过一定时间没有客户端请求连接则自动取消；
- send 用于发送数据，若发送成功则返回大于 0 的值；
- read 函数读取数据，若读取成功则返回大于 0 的值；
- close 函数用于关闭连接；
- getSend 函数用于获取待发送数据的 SocketToPhawd 对象，以设置发送数据，见第4.2.2节例程；
- getRead 函数用于获取读取数据的 SocketFromPhawd 对象，以获取读取到的数据，见第4.2.2节例程。

6 声明

PHAWD 已申请软件著作权，受国家版权局的版权保护，请勿用于商业目的 (虽然是一个很小的软件)，并遵循 GPL v3 协议。

7 联系作者

你好，我是何祜宁，感谢你使用 PHAWD，这是我在 22 年春节假期独立完成的一个很小的软件，只是最近才整理好开源 (重度拖延症)。从自学 Qt5 到它的雏形花了两个月左右，后来自己使用过程中断断续续的修过一些 bug，才得到现在这个比较稳定的版本，加上作者很菜，难免还会存在一些没有注意到的 bug。如果你真的觉得 PHAWD 对你有用，希望你能加入改进 PHAWD，在 链接 中提出你遇到的 bug 和改进想法，或者直接添加作者 QQ: 957278968 交流。但作者马上要成为一名社畜了，因此关于 PHAWD 的事情可能处理的比较慢，还请谅解。

8 待改进事项

1、只有一个波形显示界面，而不是四个或者更多，因为作者希望 PHAWD 能尽量少占用 CPU，以将电脑算力全用在机器人控制程序中，在作者的初期测试中发现如果不使用显卡进行绘制波形，波形稍微多一点作者电脑 CPU 占用就很高 (等我挣钱换电脑)，而如果使用显卡对多个波形界面进行绘制时，各类绘图库如 matplotlibcpp、QCustomPlot 以及 QChart 等都存在一些 bug，如果你有更好的解决方案请务必告诉我。

2、使用 Boost 库中的共享内存，毕竟 Boost 库可以共享 std::vector 还是很不错的，但是考虑到这是一个小软件，不想为此下一个庞大的 Boost 库，也不方便想要改进该软件的朋友配置开发环境。主要原因还是作者懒得写在一堆 vector 中找波形对应数据索引的算法了，实际上大部分机器人控制参数完全可以由多个 VEC3 的参数组成。

3、界面丑，过于原始，不过 Ubuntu 下还是能接受的。

4、添加串口通信，作者一开始就考虑到了，但苦于时间以及暂时没这方面的需求，因为串口波形显示的上位机软件已经够多了。

5、支持 Mac 系统。