

PHAWD

User Manual

Henning He

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Overview	1
2	Installation and Compilation	2
2.1	Windows	2
2.2	Linux	2
2.3	Installation of Python Package	3
3	Interface and Functionality	3
3.1	UI Introduction	3
3.2	Features and Settings	6
3.2.1	Robot Name Setting	6
3.2.2	Control Parameter Settings	6
3.2.3	Read and Save of Control Parameters	7
3.2.4	Number of Waveform Parametres	7
3.2.5	Data Write Detection and Terminal Hint	9
3.2.6	Waveform Display	9
3.2.7	Joystick Test	12
4	Demos For Robot Program	12
4.1	C++	13
4.1.1	SharedMemory	14
4.1.2	TCP/IP	16
4.2	Python	18
4.2.1	SharedMemory	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	26
5.4.1	C++	26
5.4.2	Python	27
5.5	GamepadCommand	28
5.5.1	C++	28
5.5.2	Python	28
5.6	SharedParameters	30
5.6.1	C++	30
5.6.2	Python	31
5.6.3	Usage in C++	33

5.7	SocketFromPhawd	33
5.7.1	C++	33
5.7.2	Python	34
5.8	SocketToPhawd	35
5.8.1	C++	35
5.8.2	Python	35
5.9	SharedMemory	36
5.9.1	C++	36
5.9.2	Python	37
5.10	SocketConnect	38
5.10.1	C++	38
5.10.2	Python	39
6	Acknowledgement and License	40
7	Contact	40
8	Improvement	40

1 Introduction

1.1 Motivation

PHAWD, targeting the field of robot simulation and control, is developed with the purpose of simplifying the process of debugging robot control parameters. It enables control engineers to observe the impact of different control parameters on the robot's motion state in real time, avoiding the need for constant parameter modifications and recompilation of the control program. This improves the efficiency of parameter debugging and helps control algorithm engineers record optimal parameters for future testing.

1.2 Overview

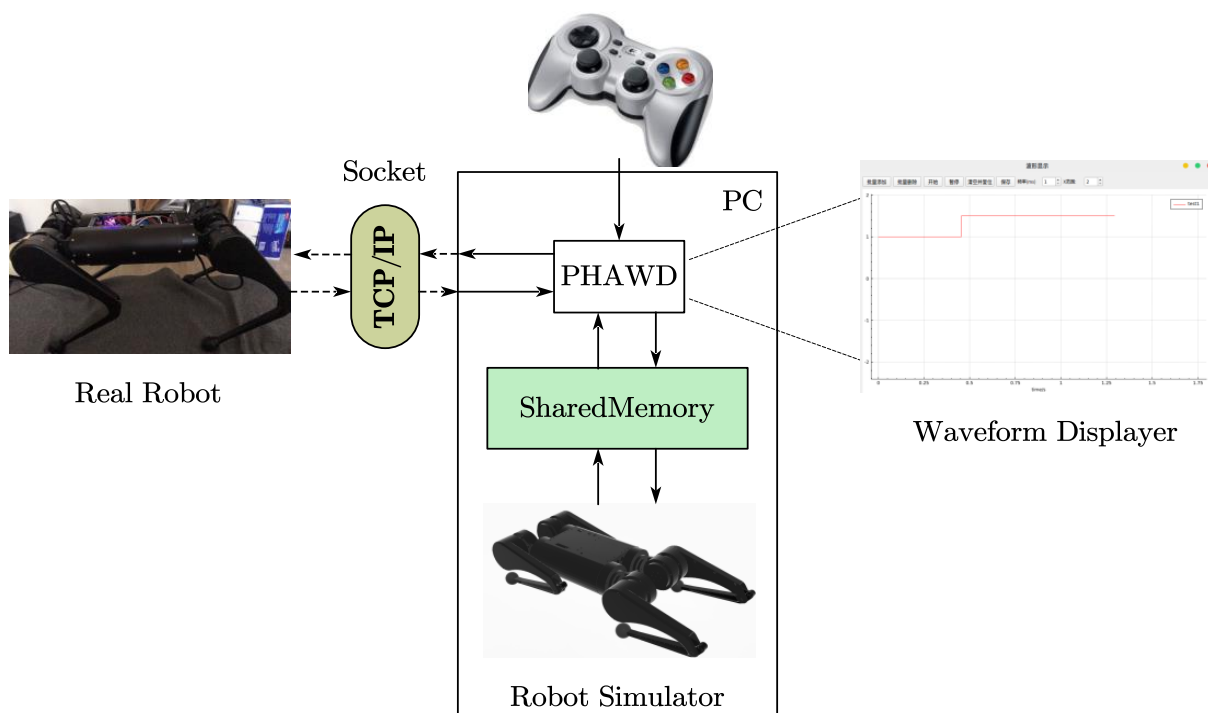


Figure 1.1 PHAWD Overall Schematic

PHAWD is an upper computer software that exchanges data with robot simulation software and physical robot controllers. As shown in Figure1.1, it is user-friendly, compact, and stable. PHAWD provides two communication methods for users to choose: shared memory and socket.

The shared memory method is commonly used in robot simulation. In this method, PHAWD allocates a block of shared memory and stores all control parameter data and joystick data in it. The robot control program continuously reads updated control parameters from the shared memory and can also write robot states, such as Euler angles, to the shared memory. When PHAWD detects a data write event, users can choose to plot the desired waveforms.

The socket communication method is generally used for debugging physical robots. The robot controller and PHAWD are connected via TCP/IP. Once the parameters are set and confirmed, PHAWD enters the listening state. When PHAWD establishes a connection with the robot, it only sends data to the robot when there are updates in parameter values or joystick input. At the same time, PHAWD continuously reads status information sent by the robot control program for waveform display. It does not block even if no new data is received.


Users can save the optimal collection of parameters to a YAML file, PHAWD can also directly load parameter configurations from a YAML file for convenient testing. Additionally, PHAWD supports joystick testing by sending joystick control data to the robot control program. This helps compensate for limitations such as certain robot simulators not supporting joystick and complexities in reading joystick inputs from physical robot control boards.

In the waveform display feature, users can choose to add multiple data sources, adding one to four curves at a time. Users can also delete or clear unwanted waveforms as needed. The sampling frequency of the waveform can be set by the user. Users can use the left mouse button to select and zoom in on a specific area of the waveform, use the right mouse button to drag and pan the waveform, and use the mouse scroll wheel for overall scaling.

2 Installation and Compilation

2.1 Windows

☑ Installation:

Click the  [link](#) to download "phawd_setup_*.exe" and follow the steps to install it. **Please note that it is recommended to avoid installing it on the system disk to prevent permission issues when creating shared memory.** If you prefer to install it on the system disk, please run PHAWD with administrator privileges. The default installation comes with an English interface. If you prefer a Chinese interface, you can download "phawd_windows_zhCN.exe" from the link, rename the file to "phawd.exe", and replace the file with the same name in the "bin" folder of the PHAWD installation directory.


☑ Compilation:

Please make sure to modify the Qt path in the "CMakeLists.txt" file to match your Qt installation directory. If you are using the Qt5 bundled MinGW compiler, the configuration is similar to the one described below. I won't elaborate on it here.

```
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

☑ Installation on Ubuntu:

Click on the  [link](#) to download "phawd_*.amd64.deb", and follow the steps to install it. The default installation will be in English. If you prefer a Chinese interface, you can download "phawd_linux_zhCN" from the link and rename the file to "phawd". Replace the existing file

with the same name in the bin folder of the PHAWD installation directory. On Linux, this folder is typically located at "/usr/local/bin".

☑ **Compilation:**

Please make sure to modify the Qt path in the "CMakeLists.txt" file to match your Qt installation directory.

```
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 Installation of Python Package

Please refer to the [🔗 link](#) for installation instructions:

```
pip install phawd
```

To install from source code:

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

3 Interface and Functionality

3.1 UI Introduction

The initial interface of PHAWD is shown in Figure 3.1. It consists of three tabs in the top left corner: "Parameters", "Waveform", and "About". Clicking on the "About" tab will switch to the About page, which contains information about the author of PHAWD and other relevant details, as shown in Figure 3.2. Clicking on the "Waveform" tab will open a new window for waveform plotting, as illustrated in Figure 3.3.

Clicking on the "Parameters" tab will reveal two options: "Parameter Settings" and "Joystick Test", as shown in Figure 3.4. Clicking on "Parameter Settings" will take you back to the initial interface, while selecting "Joystick Test" will navigate to the joystick test page, where the joystick connection status and current button states will be displayed.

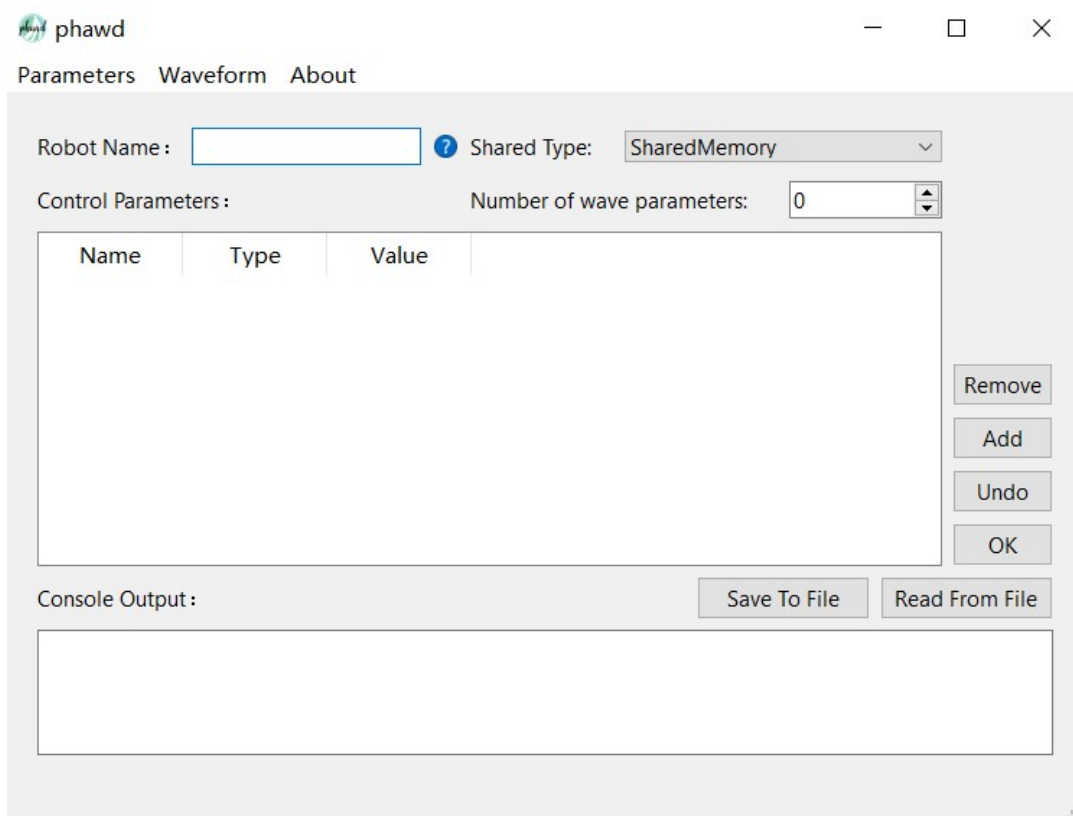


Figure 3.1 PHAWD Initial UI

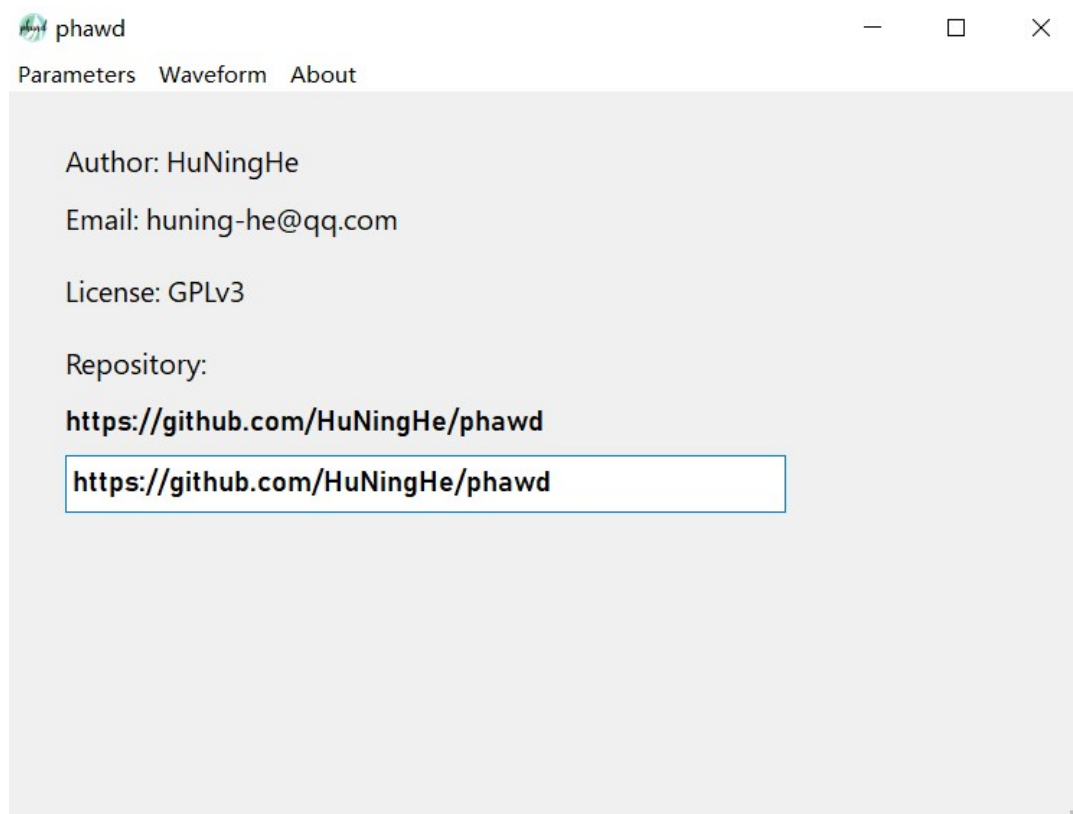


Figure 3.2 About UI

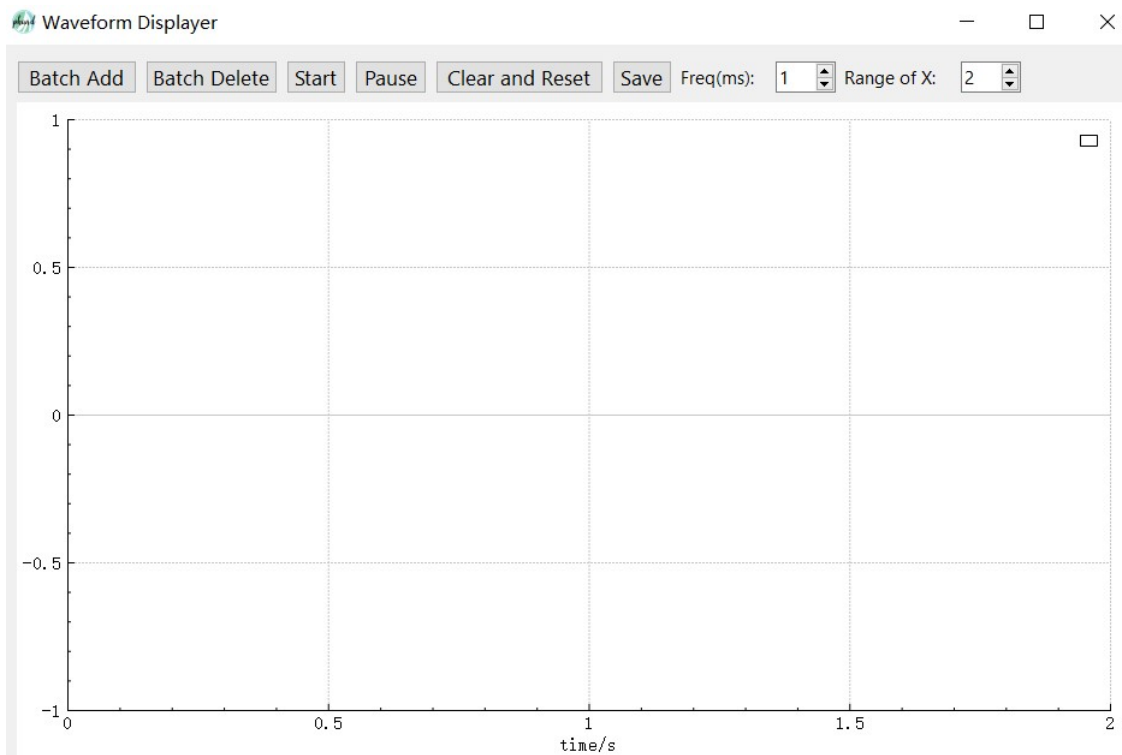


Figure 3.3 Waveform UI

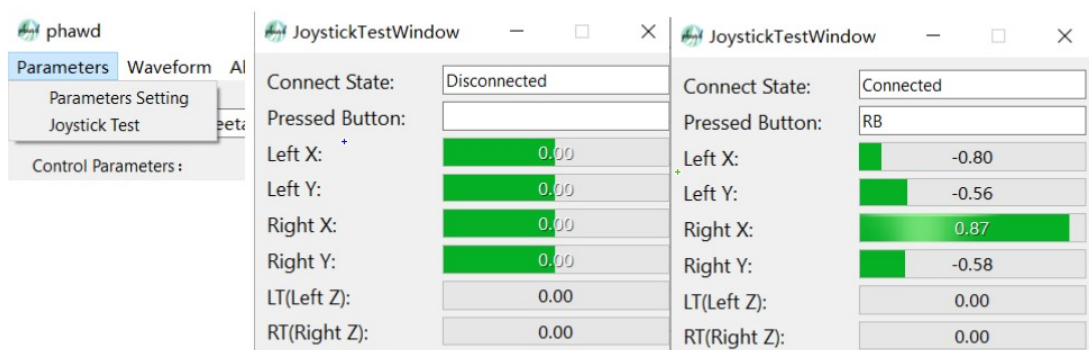


Figure 3.4 Joystick Test Window

3.2 Features and Settings

This section primarily focuses on the parameter-related functionality depicted in Figure 3.1 and the waveform display-related features illustrated in Figure 3.3. It covers the following aspects: setting the robot name, control parameter configuration (adding, deleting, clearing, save to file, and load from file), setting the number of waveform parameters, switching between sharing types, terminal prompts, and the functionality of waveform display and joystick testing.

3.2.1 Robot Name Setting

The robot name should be set based on the different sharing types. When the user is using the default sharing mode, which is shared memory, the robot name can have a maximum of 16 characters and must start with a letter. It can include alphanumeric characters and underscores, such as "MiniCheetah", "Cheetah3", "BD_Spot123", and so on. This name represents the shared memory mapping file name, which is typically generated in the "/dev/shm" folder for Ubuntu systems and in the same directory as the "phawd.exe" file for Windows 10 system.

If the user chooses Socket communication, the robot name can only be a non-zero integer of up to six digits as it represents the port number used for Socket communication. Once the port number is set, the robot control program can find PHAWD and establish Socket communication. The robot name has been validated using regular expressions, and if the input is not compliant, the user needs to check the input. **If no robot name is set, the confirmation button will be disabled, and it will not be possible to create shared memory or enter the port listening state.**

3.2.2 Control Parameter Settings

A parameter consists of three pieces of information: "parameter name", "parameter type", and "parameter value". The parameter name can have a maximum length of 16 characters and follows the naming rules set for robot names. Double-clicking on a cell in the parameter type column allows you to choose from five types: "FLOAT", "DOUBLE", "S64", "VEC3_FLOAT", and "VEC3_DOUBLE", as shown in the following figure 3.5.

"FLOAT" represents single-precision floating-point number, "DOUBLE" represents double-precision floating-point number, "S64" represents 64-bit signed integer, "VEC3_FLOAT" represents a three-dimensional array of "FLOAT", and "VEC3_DOUBLE" represents a three-dimensional array of "DOUBLE". **Please note that if you are using Python to write the robot controller, "FLOAT" and "VEC3_FLOAT" types may not be available.**

The parameter value can be enclosed in square brackets "[" and "]" or can be without square brackets. For VEC3 array types, the three numbers in the array must be separated by commas, such as [1.1, 2.2, 3.3], and there can be any number of spaces between the commas and the numbers. **If the parameter value or parameter name is not filled in according to the specification, the confirmation will fail. If the confirmation is successful and the new parameter value is not formatted correctly, the new value will not be written to shared memory or sent over the network.**

To delete or add parameters, you can click the "Add/Delete" button or right-click inside the table to open a context menu, as shown in the following figure 3.6. The context menu will have five options: "Add Row", "Remove Row", "Clear All", "Load From File", and "Save to File". The "Add Row" function is the same as the "Add" button, which adds a new row appended the currently selected row, thus adding a new parameter. The "Remove

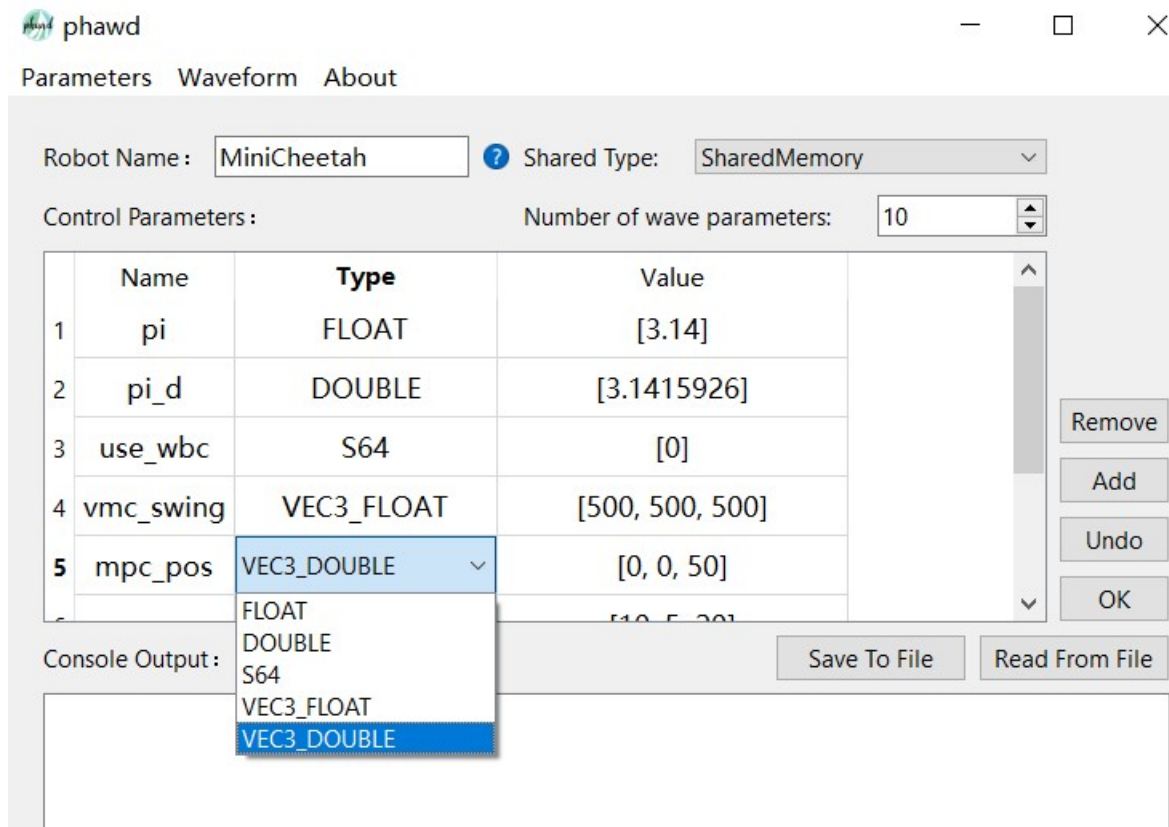


Figure 3.5 Parameter Type Setting

Row" function is the same as the "Delete" button, which deletes the currently selected row. "Clear All" clears all the parameters.

Once the robot name, sharing type, number of waveform parameters, and all parameter settings have been completed, you can click the "OK" button to create shared memory or start listening on the specified network port. **Once confirmed, you cannot modify parameter types, parameter names, shared memory names, communication methods, or Socket communication ports on the initial interface. You can only click "Undo" to reset the settings. At this point, you are only allowed to click "Undo" or "Save to File".** Make sure to stop the robot program before clicking "Undo".

3.2.3 Read and Save of Control Parameters

As shown in Figure 3.6, the right-click menu also includes options to save parameters to a file and load parameters from a file, which correspond to the respective buttons. After setting the parameters, you can save them to a YAML file as shown in Figure 3.7. Users can refer to the YAML file generated by PHAWD or write their own YAML file. If no parameters are added before saving, PHAWD will generate a template YAML file. You can also refer to the YAML file in Figure 3.7 for writing. Afterwards, you can directly load control parameters from the YAML file, which improves efficiency.

3.2.4 Number of Waveform Parametres

The number of waveform parameters needs to be informed to PHAWD in advance in order to allocate sufficient memory for storing waveform data. It can be set according to the user's needs, with a maximum value of 29999. It is important to note that if no control parameters

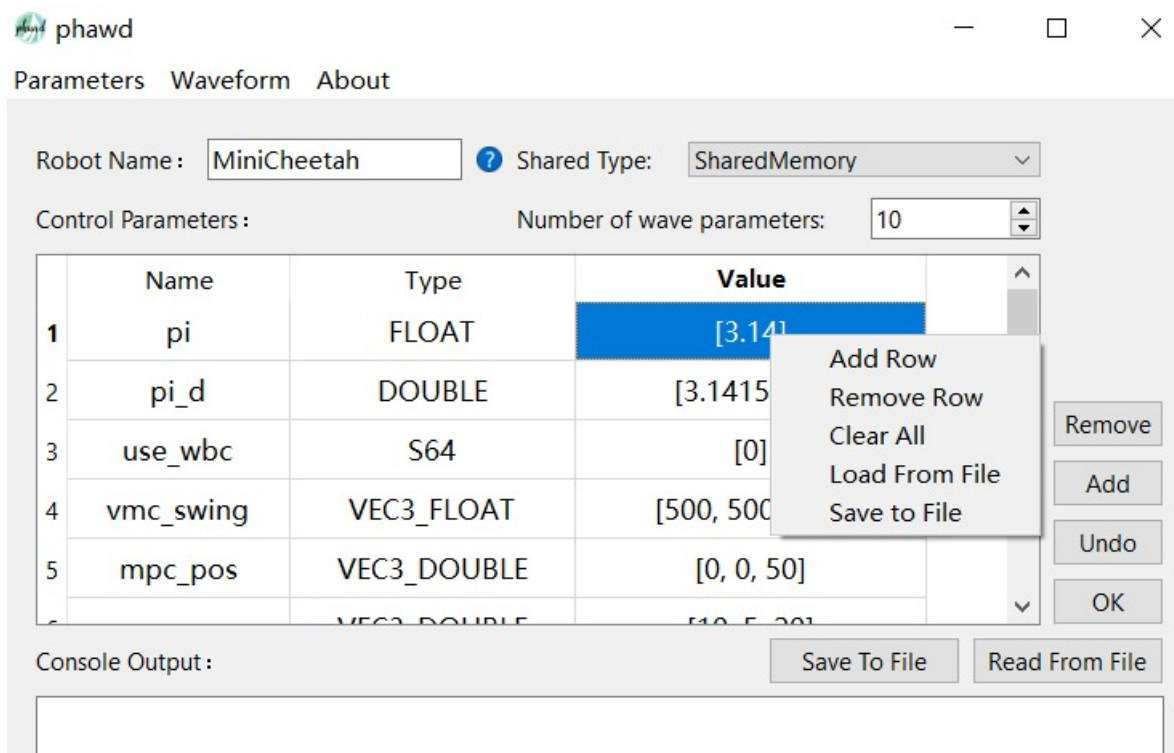


Figure 3.6 Parameter Context Menu

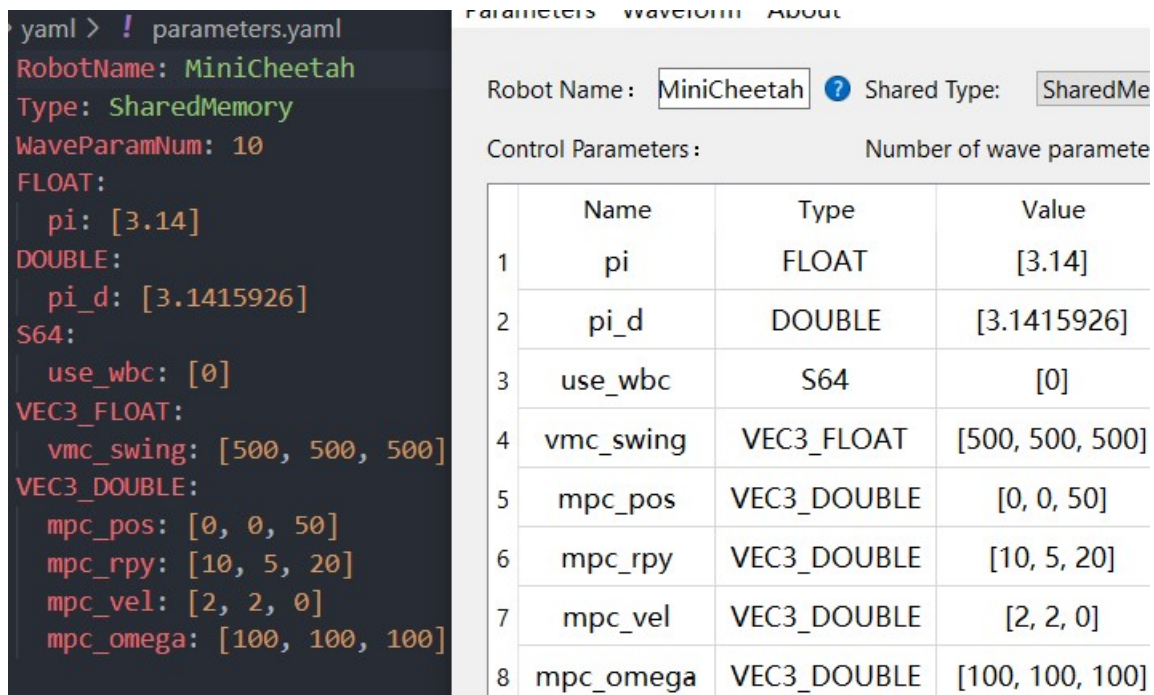


Figure 3.7 Parameter Configuration YAML File

are added and the number of waveform parameters is also zero, the confirmation button for starting communication cannot be clicked. Alternatively, you can choose not to set control parameters and only set number of waveform parameters, using PHAWD as an oscilloscope. Additionally, when setting the parameters, it is advisable to consider the size of your computer's memory to avoid insufficient memory.

3.2.5 Data Write Detection and Terminal Hint

The data writing detection thread in PHAWD monitors the shared memory for the flag set by the robot control program. Once the flag is set to 1, the thread automatically exits and passes all waveform parameter names to the waveform display window. Subsequently, users can select the corresponding data source in the waveform display window to visualize the waveforms. When data is detected, a prompt will be printed in the terminal as shown in Figure 3.8.

The terminal will output various prompts, such as the number of parameters and the memory usage after allocating shared memory. For example, when loading parameters from a file, there may be prompts in black text indicating that some parameter types are missing or the robot name is not set.

If the robot name is not set or no parameters are added before clicking confirm or saving to a file, or if there is a permission issue preventing the creation of shared memory, or if there is an existing shared memory mapping file with the same name, a red warning will be displayed. In such cases, users need to follow the instructions provided in the red warning prompts. For example, they may need to delete the mapping file with the same name in the `"/dev/shm"` directory or resolve conflicts with system-occupied socket ports.

Right-clicking inside the terminal provides options such as clearing the terminal, copying text, and selecting all, as shown in Figure 3.8.

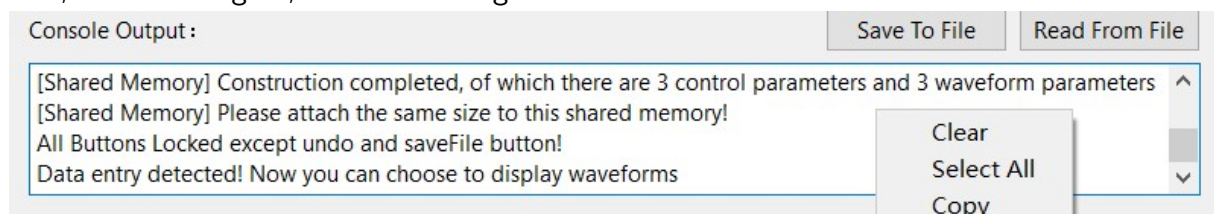


Figure 3.8 Terminal Output Prompt

3.2.6 Waveform Display

After detecting data written by the robot control program, PHAWD allows you to add the desired waveforms. The waveform displayer supports right-click and drag to pan, left-click to box-select and zoom in on a specific region, and mouse scroll to zoom out or zoom in on the entire waveform graph. The effects are shown in Figure 3.9 and Figure 3.10. Clicking the "Save" button allows you to save the current display as a `".png"` image. The horizontal axis range can be freely set. The "Clear" button will reset and clear all curves, and the next plot will start from time 0.

When the user clicks on "Batch Add", they can choose to add 0 to 4 different curves. For VEC3 array parameters, `"-x/-y/-z"` will be automatically appended to the parameter name in sequence, as shown in Figure 3.11. When clicking on "Batch Remove", the user can choose to remove 0 to 4 different curves, as shown in Figure 3.12. When adding curves, the user can choose the thickness, style, and color of the curves. The thickness ranges from



Figure 3.9 Box Selection to Zoom

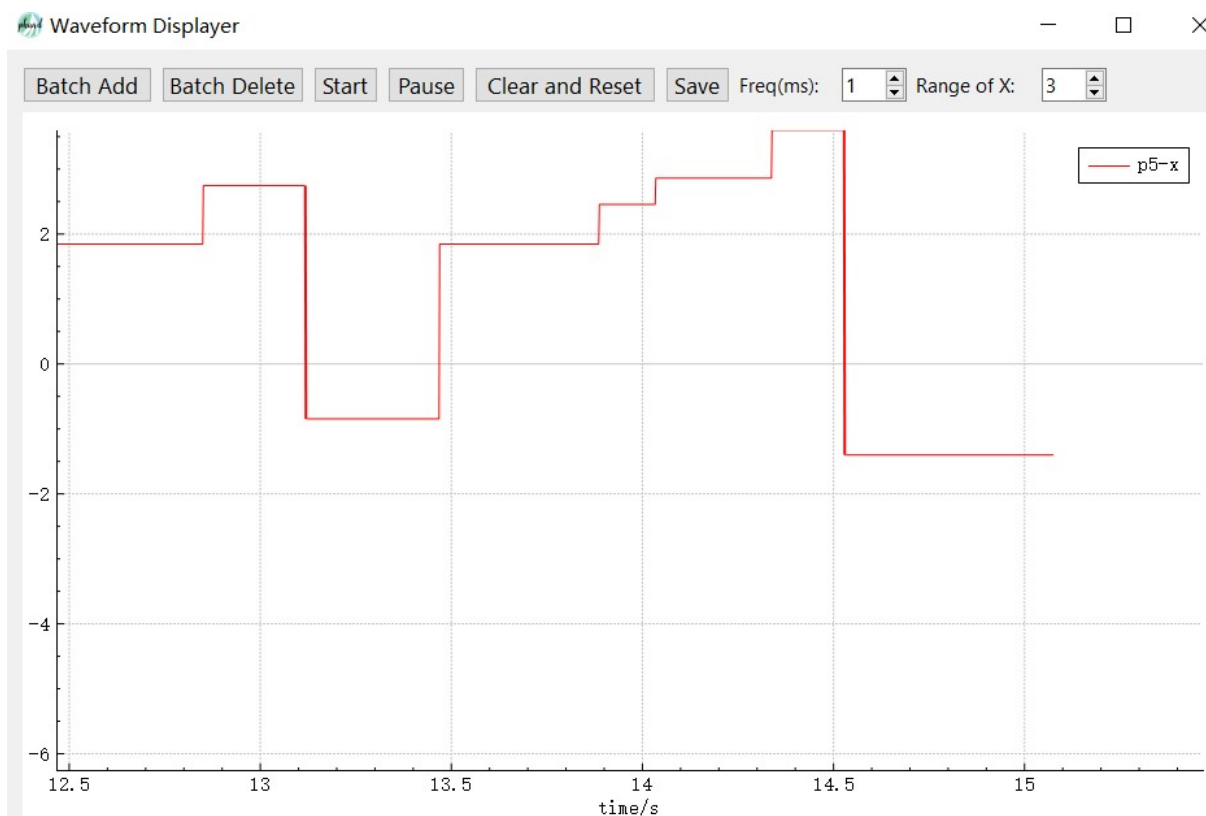


Figure 3.10 Right Click Drag to Pan

1 to 5. The available styles are "SolidLine", "DashLine", "DotLine", "DashDotLine", "DashDotDotLine", as shown in Figure 3.13.

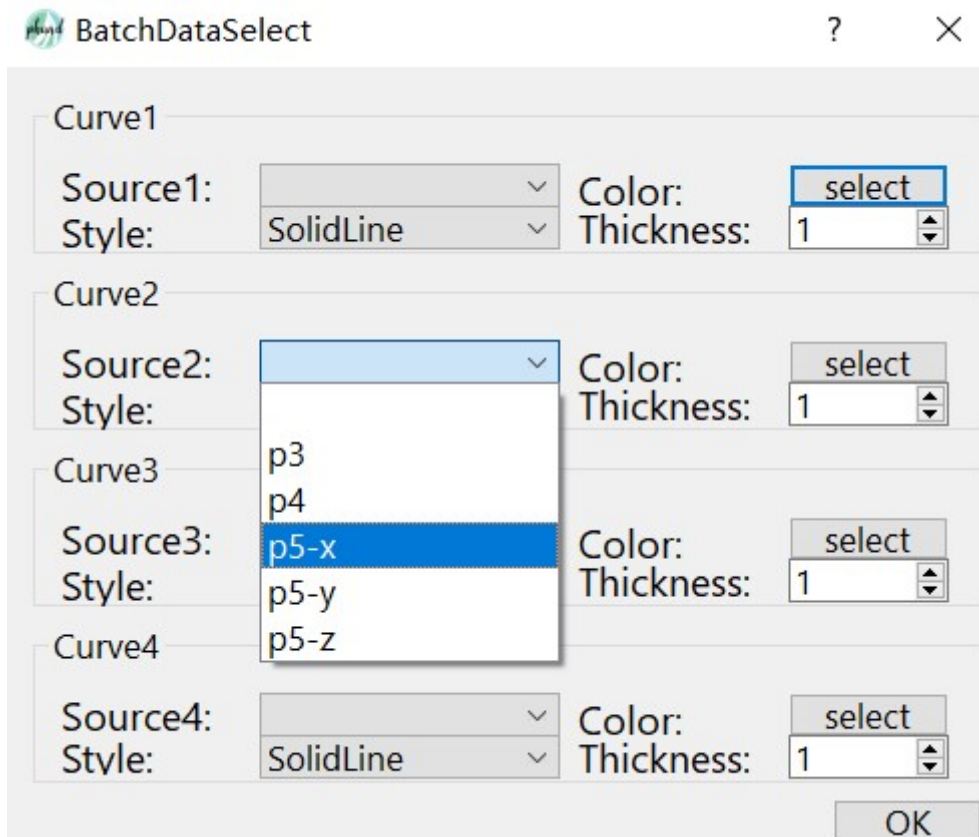


Figure 3.11 Batch Add Schematic

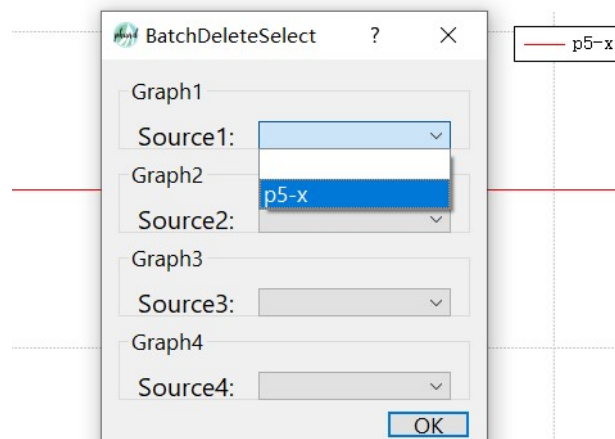


Figure 3.12 Batch Delete Schematic

Clicking on the "Select" button in Figure 3.11 will take you to the curve color selection interface, as shown in Figure 3.14. **A curve can only be added when both the color and data source have been selected.** You can only add one curve at a time. However, if you accidentally opened the window and don't want to add a curve, simply close the window (the same applies to removing curves). Clicking on "Start" will begin sampling data and drawing the waveform at the specified frequency. Modifying the frequency only takes effect after the waveform has started drawing. Clicking on "Pause" will pause the curve drawing.

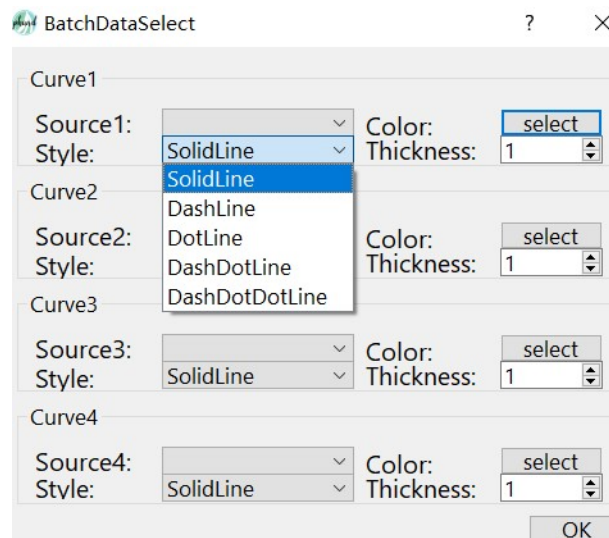


Figure 3.13 Curve Style Selection

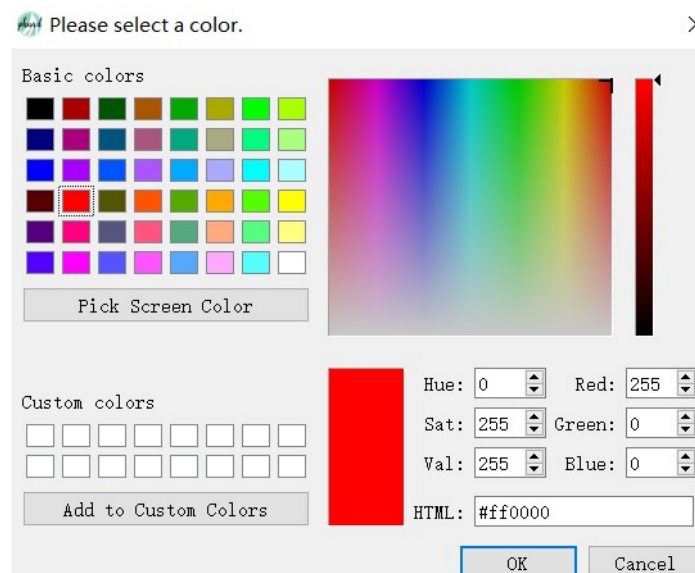


Figure 3.14 Curve Color Selection

3.2.7 Joystick Test

The author uses a Logitech F710 gamepad, which should be set to X mode. The "Mode" light on the gamepad should not be lit. If PHAWD is unable to detect your gamepad, you can try plugging it into a different USB port. If the issue persists, you can use the built-in gamepad testing functionality in Windows to verify that the gamepad is properly connected to the computer.

4 Demos For Robot Program

In the provided [link](#), you can find the latest sample codes, including examples for We-bots and MuJoCo. The "examples" directory in the PHAWD installation directory contains the following sample codes. Here, I will introduce two basic examples that include robot control programs using shared memory and TCP/IP network communication. The gamepad data

can be read from the "SharedParameters" structure in shared memory or from the "Socket-FromPhawd" structure. However, I won't demonstrate it here, but you can refer to Section 5.6.3 for more details.

4.1 C++

First, read the following yaml file in PHAWD:

```
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]
```

Both examples use this parameter configuration. When using the Socket, you need to modify the "Type" field to "Socket" and the "RobotName" field to "5230". Both examples share the same CMakeLists.txt as follows. Please modify the "CMAKE_PREFIX_PATH" to your installation path.

```
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)
```

It should be noted that when using the phawd static library on Windows, the project must add the PHAWD_STATIC macro. The main.cpp corresponding to shared memory is as follows:

4.1.1 SharedMemory

```
#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;
}

```

Steps:

- Read the aforementioned YAML file using PHAWD and click the "OK" button;
- Compile this C++ project, you can use Visual Studio or MinGW on Windows, and g++ on Linux. After compiling the project, run the program;
- Open the waveform displayer interface of the PHAWD and add the corresponding five curves;
- Modifying any parameter value in the PHAWD parameter settings interface will cause the corresponding curve to change accordingly.

4.1.2 TCP/IP

This example runs on the same computer where PHAWD is installed, so it uses the local IP address 127.0.0.1. Users can run it on other devices (Ubuntu/Windows) by setting the appropriate IP address and copying the corresponding dynamic library(phawd.dll or libphawd.so) to the directory where the executable file is located. The corresponding main.cpp is as follows:

```

#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;
}

```

Steps:

- Modify the YAML file mentioned above, use PHAWD to read it and click the "OK" button;
- Compile this C++ project, you can use Visual Studio or MinGW on Windows, and g++ on Linux. After compiling the project, run the program;
- Open the waveform displayer interface of the PHAWD and add the corresponding five curves;
- Modifying any parameter value in the PHAWD parameter settings interface will cause the corresponding curve to change accordingly;
- The terminal keeps outputting the parameter values sent by PHAWD.

4.2 Python

PHAWD's Python package does not support "FLOAT" and "VEC3.FLOAT" type parameters.

Similar to the C++ example, use PHAWD to read the following yaml file:

```

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 SharedMemory

The example program is as follows:

```
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

```

Steps:

- Read the aforementioned YAML file using PHAWD and click the "OK" button;
- Run this Python script;
- Open the waveform displayer interface of the PHAWD and add the corresponding 3 curves;
- Modifying any parameter value in the PHAWD parameter settings interface will cause the corresponding curve to change accordingly;
- The terminal keeps outputting the parameter values sent by PHAWD.

4.2.2 TCP/IP

This example runs on the same computer where PHAWD is installed, so it uses the local IP address 127.0.0.1. Users can run it on other devices (Ubuntu/Windows) by setting the appropriate IP address. Python script is as follows:

```

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())
```

Steps:

- Modify the YAML file mentioned above and use PHAWD to read it and click the "OK" button;
- Run this Python script;

- Open the waveform displayer interface of the PHAWD and add the corresponding 3 curves;
- Modifying any parameter value in the PHAWD parameter settings interface will cause the corresponding curve to change accordingly;
- The terminal keeps outputting the parameter values sent by PHAWD.

5 API Reference

This section contains an introduction to the PHAWD API, including the description of relevant function parameters and their purposes. You can download the latest documentation from the [Doc](#) for detailed information.

5.1 ParameterKind

The parameter types management in PHAWD includes various types that can be used to define parameters. However, please note that in Python, the "FLOAT" and "VEC3_FLOAT" types are not supported. The remaining types are consistent with those available in C++.

```
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
```

The functions "getParameterKindFromString" and "ParameterKindToString" are specific to C++. In C++, "getParameterKindFromString" is used to convert a string representation of a parameter type, such as "DOUBLE," to its corresponding ParameterKind enum value, like ParameterKind::DOUBLE. The ParameterKind object cannot be printed by cout, so you need do this conversion. Similarly, "ParameterKindToString" is used to convert a ParameterKind enum value, such as ParameterKind::DOUBLE, to its string representation, like "DOUBLE". In Python, you can directly print the ParameterKind enum value using print. There is no need to use these specific functions for string conversion.

```

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

```

5.2 ParameterValue

Ensure that each parameter consumes the same memory.

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 are assigned and read directly as properties in Python, and the corresponding setters are no longer listed.

```

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

```

5.3 Parameter

This class contains parameter information. Parameter Stored in the flexible arrays "Shared-Parameters", "SocketFromPhawd", and "SocketToPhawd" in Section 5.6, Section 5.7, and Section 5.8.

5.3.1 C++

```
class Parameter {
public:
    // default 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

Usage refers to Section 4.2.1.

```

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++

Usage refers to Section 4.1.1.

```

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

Usage refers to Section 4.2.1.

```
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

Can be directly used as a property for assignment and reading operations, and the corresponding setter is no longer listed.

```

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

The data structure stored in shared memory, generally created directly by the "Shared-Memory" object, without user creation, users only need to understand how to read and write data to shared memory through SharedParameters.

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);
};
```

- The SharedParameters object can only be created in the heap using the "create" function and must be manually destroyed using the "destroy" function. The parameters num_control_params and num_wave_params in the "create" function are used to determine the number of control parameters and waveform parameters, but they are generally not necessary;
- The "collectParameters" function is used to add all the parameters from the SharedParameters object to the ParameterCollection, as shown in the example in Section 4.2.1;
- The "connected" attribute indicates the number of processes connected to the shared memory. Each time a connection is made to the shared memory, this value needs to be manually incremented to ensure that PHAWD can detect data writing, as shown in the example in Section 4.1.1;
- The "numControlParams" and "numWaveParams" variables store the number of control parameters and waveform parameters, respectively. These values are automatically written by PHAWD and can only be read by the user. Do not modify them arbitrarily;
- The "parameters" array can be directly accessed to read and write parameters;
- The "gameCommand" variable stores the gamepad data.

5.6.2 Python

Refer to examples in Section 4.2.1.

```
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 version
    @property
    def connected(self):
    @property
    def numControlParams(self):
    @property
    def numWaveParams(self):
    @property
    def gamepadCommand(self):

```

- The SharedParameters object can only be created in the heap using the "create" function and must be manually destroyed using the "destroy" function. The parameters num_control_params and num_wave_params in the "create" function are used to determine the number of control parameters and waveform parameters, but they are generally not necessary;
- The "collectParameters" function is used to add all the parameters from the SharedParameters object to the ParameterCollection, as shown in the example in Section 4.2.1;
- The "connected" attribute indicates the number of processes connected to the shared memory. Each time a connection is made to the shared memory, this value needs to be manually incremented to ensure that PHAWD can detect data writing, as shown in the example in Section 4.2.1;
- The "numControlParams" and "numWaveParams" variables store the number of control parameters and waveform parameters, respectively. These values are automatically written by PHAWD and can only be read by the user. Do not modify them arbitrarily;
- The "gamepadCommand" variable stores the gamepad data;
- Unlike C++, Python does not directly manipulate the parameters array. Instead, the "setParameters" function is used to set the parameters. The param_list parameter of this function is a list of Parameter objects. The set_control_params parameter is used to specify whether to set control parameters or waveform parameters. By default, it is set to False, indicating that waveform parameters are being set. This can be seen in the example in Section 4.2.1.
- "getParameters" function is used to get the parameters. Return value is a list of Parameter objects. The get_control_params parameter is used to specify whether to get control parameters or waveform parameters. By default, it is set to True, indicating that control parameters are being got. This can be seen in the example in Section 4.2.1.
- "__sizeof__" function is used to retrieve the memory size occupied by data of "SharedParameters". It is useful for determining the size of data to be sent using SocketConnect.

5.6.3 Usage in C++

The "SharedParameters" structure is typically accessed in shared memory by obtaining a pointer or object using the following approach:

```
#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

The data structure corresponding to the message received from PHAWD via TCP/IP is generally created directly by the SocketConnect object and does not need to be created by the user. Users only need to understand how to read the data. For specific usage examples, please refer to Section 4.1.2 and Section 4.2.2.

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);
};

```

The API operations for the data structure related to "SocketConnect" are similar to those of the "SharedParameters" class. For specific details, please refer to the C++ section of Section 5.6 for reference.

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):

```

The API operations for the data structure related to "SocketConnect" are similar to those of the "SharedParameters" class. For specific details, please refer to the C++ section of Section 5.6 for reference.

In this case, you can directly read the parameters through the parameters list, which contains only the control parameters. Please refer to the example in Section 4.2.2 for more details.

5.8 SocketToPhawd

The data structure corresponding to the message sent to PHAWD via TCP/IP is typically created directly by the "SocketConnect" object and does not need to be created by the user. Users only need to understand how to write data into the structure. Please refer to the examples in Section 4.1.2 and Section 4.2.2 for more details.

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);
};
```

The API operations for the data structure related to "SocketConnect" are similar to those of the "SharedParameters" class. For specific details, please refer to the C++ section of Section 5.6 for reference.

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):

```

The API operations for the data structure related to "SocketConnect" are similar to those of the "SharedParameters" class. For specific details, please refer to the python section of Section 5.6 for reference. In this case, you can directly write data into the "parameters" list. Please refer to the example in Section 4.2.2 for more details.

5.9 SharedMemory

A template class for creating and connecting shared memory in C++, but only the "SharedParameters" type is supported.

5.9.1 C++

```

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()();
};

```

- "createNew" is used to create shared memory, "name" specifies shared memory's name,

"size" specifies shared memory size, "allowOverwrite" is used to determine whether overwriting of the rewritten memory-mapped file is allowed;

- The "attach" is used to connect to shared memory. The "name" parameter specifies the name of the shared memory to connect to, and the "size" parameter specifies the size of the shared memory to connect to, refer to example in Section 4.1.1;
- "closeNew" is used to close shared memory created by createNew;
- "detach" is used to detach from shared memory;
- "get" is used to get pointer of "SharedParameters";
- The SharedMemory functor is used to obtain a reference to SharedParameters.

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):
```

- "createNew" is used to create shared memory, "name" specifies shared memory's name, "size" specifies shared memory size, "allowOverwrite" is used to determine whether overwriting of the rewritten memory-mapped file is allowed;
- The "attach" is used to connect to shared memory. The "name" parameter specifies the name of the shared memory to connect to, and the "size" parameter specifies the size of the shared memory to connect to, refer to example in Section 4.2.1;
- "closeNew" is used to close shared memory created by createNew;
- "detach" is used to detach from shared memory;
- "get" is used to get object of "SharedParameters"
- The SharedMemory functor is used to obtain a object to "SharedParameters".

5.10 SocketConnect

In C++, the SharedMemory functor is a template class used for TCP/IP connection. It is used to send data of type SocketToPhawd and receive data of type SocketFromPhawd.

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();
};
```

- The "Init" is used to initialize a "SocketConnect" object, including determining the size of the sent and received messages, as well as specifying whether the process acts as a server. The settings for the size of the sent and received data can be found in the example provided in Section 4.1.2;
- The "connectToServer" is used for a client to request a connection to a server. The first parameter is the IP address of the server, the second parameter is the port number, which is usually set as the robot name in PHAWD. The third parameter is the timeout duration, after which the connection attempt is automatically canceled if no connection is established. Please refer to the example provided in Section 4.1.2 for more details;

- The "listenToClient" is used by the server to listen for client connections. The first parameter is the listening port, the second parameter is the length of the listening queue, and the third parameter is the waiting time. If there is no client connection request within a certain time, the function automatically cancels the listening;
- "Send" is used to send data. If the data is sent successfully, it returns a value greater than 0;
- "Read" is used to read data. If the data is read successfully, it returns a value greater than 0;
- "Close" is used to close socket connection;
- "getSend" is used to get the pointer of SocketToPhawd, to set up sending data, Please refer to the example provided in Section 4.1.2 for more details;
- "getRead" is used to get the pointer of SocketFromPhawd, to get reading data, Please refer to the example provided in Section 4.1.2 for more details.

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):
```


- The "init" is used to initialize a "SocketConnect" object, including determining the size of the sent and received messages, as well as specifying whether the process acts as a server. The settings for the size of the sent and received data can be found in the example provided in Section 4.1.2;

- The "connectToServer" is used for a client to request a connection to a server. The first parameter is the IP address of the server, the second parameter is the port number, which is usually set as the robot name in PHAWD. The third parameter is the timeout duration, after which the connection attempt is automatically canceled if no connection is established. Please refer to the example provided in Section 4.1.2 for more details;
- The "listenToClient" is used by the server to listen for client connections. The first parameter is the listening port, the second parameter is the length of the listening queue, and the third parameter is the waiting time. If there is no client connection request within a certain time, the function automatically cancels the listening;
- "send" is used to send data. If the data is sent successfully, it returns a value greater than 0;
- "read" is used to read data. If the data is read successfully, it returns a value greater than 0;
- "close" is used to close socket connection;
- "getSend" is used to get the object of SocketToPhawd, to set up sending data, Please refer to the example provided in Section 4.1.2 for more details;
- "getRead" is used to get the object of SocketFromPhawd, to get reading data, Please refer to the example provided in Section 4.1.2 for more details.

6 Acknowledgement and License

PHAWD is distributed with [GNU General Public License v3.0](#).

7 Contact

Hello, I'm Henning He. Thank you for using PHAWD. This is a small software project that I independently completed during the Spring Festival holiday in 2022. It took me about two months from self-learning Qt5 to its initial development. Later, during my own usage, I intermittently fixed some bugs, resulting in the relatively stable version that it is now. However, as the author, I'm not very skilled, so there may still be some bugs that I haven't noticed. If you find PHAWD useful and would like to contribute to its improvement, please submit any bugs you encounter or improvement ideas through the  [link](#) provided or directly contact me via wechat: "HenningHe" for communication.

8 Improvement

1. There is only one waveform display interface, rather than four or more, because the author wants PHAWD to minimize CPU usage and allocate the computer's computational power to the robot control program. In the author's initial testing, it was found that without using the GPU for waveform rendering, the CPU usage would be high even with slightly more waveforms (I'll upgrade my computer when I have enough money). However, when using the GPU to render multiple waveform interfaces, various libraries such as "matplotlibcpp",

"QCustomPlot", and "QChart" have some bugs. If you have a better solution, please let me know.

2. Using the shared memory feature of the Boost library, as it is convenient for sharing `std::vector`. However, considering that PHAWD is a small software project, I didn't want to introduce the large Boost library as a dependency, which could make it inconvenient for those who want to improve or use the software. The main reason is that I didn't want to write the algorithm to search for waveform data indices in a bunch of vectors. In practice, most robot control parameters can be represented by multiple VEC3 parameters.

3. The interface is basic and may not be visually appealing.

4. Serial communication support was considered from the beginning, but due to time constraints and the lack of immediate need, it was not implemented. There are already plenty of upper-level software applications available for serial waveform display.

5. PHAWD currently supports Windows and Linux, but Mac system support is something that I have considered and may be added in the future.