

DexHand2 SDK 二次开发手册

(DexHand2 Pro)



北京橡木果机器人科技有限公司

Acorn Robotics

手册版本：v1.1

软件版本：DexHand2-SDK-v1.1

修改时间：2025 年 02 月 14 日

联系方式：support@acornrobotics.com

目录

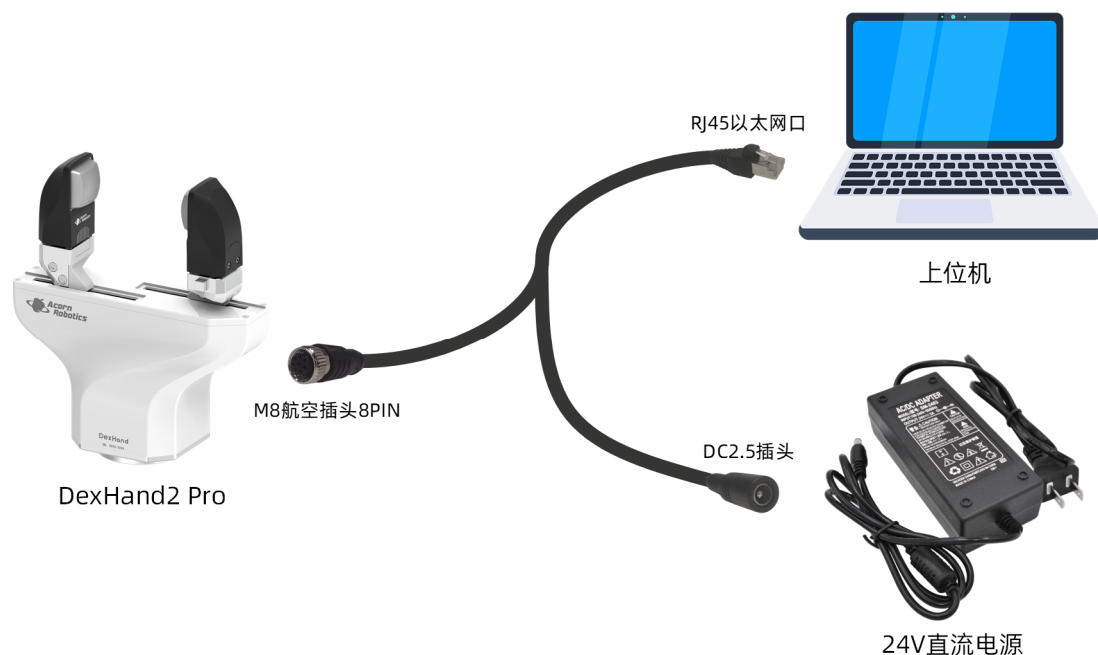
| | | |
|--------|----------------------------------|----|
| 1 | DexHand 软件介绍与使用准备..... | 1 |
| 2 | DexHand-SDK 文件目录 | 2 |
| 3 | 基于 DexHand-C 语言 SDK 的二次开发 | 3 |
| 3.1 | Windows 平台的环境配置和编译 | 3 |
| 3.2 | Linux 平台的环境配置和编译 | 3 |
| 3.3 | CDexHandClient 的 API 说明..... | 3 |
| 3.3.1 | 关于上下文 | 3 |
| 3.3.2 | 关于返回值 | 4 |
| 3.3.3 | 关于默认参数及参数范围 | 4 |
| 3.3.4 | 关于系统信号 | 4 |
| 3.3.5 | 关于 API 线程安全..... | 4 |
| 3.3.6 | 关于内存安全 | 5 |
| 3.3.7 | 数据类型定义 | 5 |
| 3.4 | CDexHandClient 的 API 介绍..... | 5 |
| 3.4.1 | 创建客户端 | 6 |
| 3.4.2 | 启动与停止服务 | 6 |
| 3.4.3 | 获取与释放控制权限 | 7 |
| 3.4.4 | 停止机械手 | 8 |
| 3.4.5 | 位置校准与力校准..... | 8 |
| 3.4.6 | 位置控制..... | 9 |
| 3.4.7 | 抓取力控制..... | 10 |
| 3.4.8 | DexHand 本体信息读取..... | 12 |
| 3.4.9 | Tac3D 信息读取 | 13 |
| 3.4.10 | 错误消除 | 16 |
| 3.4.11 | 日志设置 | 16 |
| 3.5 | CDexHandClient 的例程说明..... | 16 |
| 3.5.1 | 客户端例程 client.c | 16 |
| 3.5.2 | Tac3D 信息读取例程 tac3d.cpp..... | 19 |
| 4 | 基于 DexHand-Python SDK 的二次开发..... | 25 |
| 4.1 | 环境配置 | 25 |
| 4.1.1 | 安装 dexhand_client 包 | 25 |
| 4.1.2 | 检验 dexhand_client 包是否安装正确..... | 25 |
| 4.1.3 | 使用 conda 的情况..... | 26 |
| 4.2 | PyDexHand 的 API 介绍..... | 26 |

| | | |
|--------|--------------------------------------|----|
| 4.2.1 | 创建客户端 | 26 |
| 4.2.2 | 启动与停止服务 | 27 |
| 4.2.3 | 获取与释放控制权限 | 28 |
| 4.2.4 | 停止机械手 | 29 |
| 4.2.5 | 位置校准与力校准 | 29 |
| 4.2.6 | 位置控制 | 30 |
| 4.2.7 | 抓取力控制 | 31 |
| 4.2.8 | DexHand 本体信息读取 | 32 |
| 4.2.9 | Tac3D 信息读取 | 34 |
| 4.2.10 | 错误消除 | 36 |
| 4.3 | PyDexHand 的例程说明 | 36 |
| 4.3.1 | 开启服务端例程 activate_service | 37 |
| 4.3.2 | 获取机械手信息例程 get_info | 37 |
| 4.3.3 | 机械手力控例程 grasp_force_control | 38 |
| 4.3.4 | 机械手运动例程 move_dexhand | 39 |
| 4.3.5 | 机械手与 Tac3D 联合使用例程 handandtac3d | 39 |
| 5 | 问题与解答 FAQ | 45 |
| 5.1 | 机械手报错无法执行指令? | 45 |
| 5.2 | 启动机械手后将指尖传感器拆下? | 45 |
| 5.3 | 机械手输出力是否存在上限? | 45 |
| 5.4 | 机械手在进行抓取力控制时出现了震荡? | 45 |
| 5.5 | 机械手在进行位置校准后提示电流过大? | 46 |
| 5.6 | 机械手在进行力校准后读数偏离零点严重? | 46 |

1 DexHand 软件介绍与使用准备

本使用手册适用于 DexHand2 Pro 系列机械手。在本使用手册下文中所述的“机械手”、“DexHand”均特指 DexHand2 Pro 系列的机械手。

DexHand 的底层代码运行于机械手内置的控制器中，按照使用手册要求为机械手供电后，DexHand 的主程序开始运行。此时上位机的客户端可通过 UDP 网络协议与机械手控制器进行通双向通讯。若搭配 Tac3D 传感器进行使用，则 DexHand 机械手控制器将对传感器数据进行重新打包并发送至上位机。



如图所示，使用前将机械手的 RJ45 网线接口与上位机相连，并将 24V 电源通过 DC 接口与机械手相连。机械手将于通电后 10s 内完成启动，启动后用户即可开始使用。

在使用前需要根据机械手的 IP 地址调整上位机的 IP。初始状态下，机械手的 IP 地址为 192.168.2.100，端口为 60031，子网掩码为 255.255.255.0，此状态下需要将上位机的 IP 调整位于 192.168.2.x 网段(x 可以是 0 到 255 的任意值，但不能等于 100，因为 100 已经被机械手占用)。

为方便用户使用，DexHand 提供了 SDK 并开放部分数据及控制接口。SDK 包含 Python 版本和 C 版本，支持 Windows 系统和 Linux 系统。目前，本 SDK 仅在 Windows10 23H2 及以上版本进行过测试，旧版 Windows 系统可能存在不兼容或性能下降。本 SDK 仅在 Ubuntu 20.04、Arch Linux 进行过测试，其余版本的 Linux 系统下的兼容性不保证。

⚠ 注意：上电后请等待 10s 以上时间，待机械手内部控制器启动完毕，再进行 SDK 的调用。

2 DexHand-SDK 文件目录

解压 DexHand-SDK-v1.1.zip 后得到的 DexHand-SDK-v1.1 文件夹中包含以下子文件夹：

1. CDexHandClient

此文件夹中为 DexHand 的 C 语言 API 接口。具体包含如下目录：

- windows_x86_32 文件夹：Windows 系统下 X86 架构 32 位库文件 CDexHandClient.dll、编译器连接文件 DexHandClient.lib 和示例代码的编译结果 dh_client.exe。
- windows_x86_64 文件夹：Windows 系统下 X86 架构 64 位库文件 CDexHandClient.dll、编译器连接文件 DexHandClient.lib 和示例代码的编译结果 dh_client.exe。
- linux_arm_64 文件夹：Linux 系统下 ARM 架构 64 位库文件 CDexHandClient.a 和示例代码的编译结果 dh_client。
- linux_x86_32 文件夹：Linux 系统下 X86 架构 32 位库文件 CDexHandClient.a 和示例代码的编译结果 dh_client。
- linux_x86_64 文件夹：Linux 系统下 X86 架构 64 位库文件 CDexHandClient.a 和示例代码的编译结果 dh_client。
- conf 文件夹：SDK 使用的配置文件，与 Python 版一致。
- example 文件夹：示例代码 client.c。
- include 文件夹：库的 C 语言头文件 dexhand_client.h。
- licenses 文件夹：库外部依赖的许可证文件。

⚠ 注意：在使用 C 语言的 SDK 进行二次开发时，注意要将库文件(.dll,.a)与 config 文件夹放在可执行文件所在目录下。

2. pyDexHandClient

此文件夹中为 DexHand 的 Python API 接口。具体包含如下目录：

- dexhand_client 文件夹：其中包含了 dexhand_client 包的核心内容，即接口的具体实现，以及 SDK 使用的配置文件，与 C 语言版本一致。
- examples 文件夹：其中包含了部分可以使用的例程。
- setup.py：用于安装 dexhand_client 包的 Python 脚本。

3 基于 DexHand-C 语言 SDK 的二次开发

本章节将介绍基于 C 语言编写的 DexHand SDK 的 API 接口的使用方法。

3.1 Windows 平台的环境配置和编译

本 SDK 提供 C 语言 API, 保证了最大的 ABI 兼容性, 常见的 C 语言编译器均可使用。目前, 本库在 Windows 系统下的 MSYS2 环境(MinGW-w64 GCC)和 Visio Studio 2017/2022 环境下通过编译及使用测试, 理论上还支持基于 LLVM 的编译器。

由于库文件为静态编译, 因此不需要第三方库的依赖, 也无需添加额外的编译参数。因此用户可以选择任何方便使用的 C 语言开发环境, 将对应的库文件 CDexHandClient.dll、DexHandClient.lib 和头文件 dexhand_client.h 加入项目中即可。

以 MinGW-w64 GCC 为例, 假设<lib_dir>为库文件所在文件夹, <inc_dir>为头文件所在的文件夹, <src_file>为用户的源码文件或示例代码文件, <exe_file>为输出的可执行文件, 则编译指令如下:

```
gcc <src_file> -I<inc_dir> -L<lib_dir> -lDexHandClient -o <exe_file>
```

3.2 Linux 平台的环境配置和编译

本 SDK 提供 C 语言 API, 保证了最大的 ABI 兼容性, 常见的 C 语言编译器均可使用。目前, 本库在 Linux 系统中的 GCC 和 Clang (基于 LLVM) 编译器环境下通过编译及使用测试。

由于库文件为静态编译, 因此不需要第三方库的依赖, 也无需添加额外的编译参数。因此用户可以选择任何方便使用的 C 语言开发环境, 将对应的库文件 CDexHandClient.so 和头文件 dexhand.h 加入项目中即可。

以 GCC 为例, 假设<lib_dir>为库文件所在文件夹, <inc_dir>为头文件所在的文件夹, <src_file>为用户的源码文件或示例代码文件, <exe_file>为输出的可执行文件, 则编译指令如下:

```
gcc <src_file> -I<inc_dir> <lib_dir>/CDexHandClient.a -o <exe_file>
```

Clang 与 GCC 用法几乎一致, 编译指令如下:

```
clang <src_file> -I<inc_dir> <lib_dir>/CDexHandClient.a -o <exe_file>
```

3.3 CDexHandClient 的 API 说明

3.3.1 关于上下文

CDexHandClient 的 API 为 C 语言, 因此使用方式与 C++等具有类 (class) 概念的语言不同。用户在使用前, 需要初始化一个客户端上下文 (context), 并在使用结束后进行释放。每一个功能函数均需要提供上下文作为参数, 类似成员函数中的 self 概念。一个上下文对应

一台 DexHand，用户可以在同一个进程中使用多个上下文以控制多台 DexHand。

上下文的释放在用户 main 函数结束后自动进行，因此没有提供相关的 API。

3.3.2 关于返回值

默认情况下，CDexHandClient 的函数返回值为有符号整数。若函数正常结束则返回 0；若函数出现警告但不影响继续运行，则返回正数；若函数出现必须停止程序的错误，则返回负数。

3.3.3 关于默认参数及参数范围

C 语言不支持默认参数，因此采用特殊值表示该参数使用默认值。对于浮点类型，使用 NAN 作为参数时可自动转换为默认值；对于 DH_BOOL 类型，使用 DH_DEFAULT 作为参数时可自动转换为默认值。

C 语言的 API 与 Python 语言的 API 使用一致的默认参数，参数的范围检查则在 DexHand 硬件内检查。具体参数的范围请参考 API 介绍。

3.3.4 关于系统信号

考虑到客户端会占用 DexHand 硬件的资源，因此在收到系统信号（例如用户按下 Ctrl-C）时，CDexHandClient 会拦截该信号，自动释放占用的硬件资源并释放上下文。CDexHandClient 拦截的系统信号如下：

- Windows 系统：CTRL_C_EVENT、CTRL_BREAK_EVENT 和 CTRL_CLOSE_EVENT；
- Linux 系统：SIGHUP、SIGINT 和 SIGTERM。

这些信号的通常触发方式如下：

- 在终端中按下 Ctrl-C（Windows 中包含 Ctrl-Break）；
- 直接关闭终端窗口；
- 通过 kill 或任务管理器关闭程序进程。

用户如有拦截上述系统信号的需求，可以在创建客户端时设定回调函数（term_callback，详见 3.4.1），该函数与上下文关联，在释放上下文前被调用。

用户如有在 Linux 系统后台运行的需求，可以使用 nohup 指令阻止系统发送 SIGHUP。

注意：在收到系统信号后，CDexHandClient 会立即返回全部阻塞函数并阻止用户继续调用全部 API 函数，因此回调函数中也无法进行 API 的调用，也不提供上下文的指针。

3.3.5 关于 API 线程安全

CDexHandClient 虽然在设计时考虑到了线程安全问题，但并未经历充分的验证，目前关于线程安全的说明如下：

- 不同上下文之间的 API 调用保证线程安全，用户可以将每个上下文运行在不同线程中；
- 同一个上下文的 API 调用不保证线程安全，不建议用户将同一个上下文的 API 调用分开在不同线程中；

- 日志输出保证线程安全，不会出现不完整的日志条目；
- 触发系统信号时，全部上下文的 API 会立即返回并被阻止调用，全部上下文会被依次释放，没有线程安全风险。

3.3.6 关于内存安全

CDexHandClient 在开发阶段进行了较为充分的内存检查，但因为用户持有上下文的指针，因此严禁用户在初始化上下文后更改指针的内容。

3.3.7 数据类型定义

由于标准 C 语言没有提供 bool 类型，因此定义了 DH_BOOL 枚举类型作为布尔类型。

- 布尔类型枚举：DH_BOOL
 - DH_FALSE：假；
 - DH_TRUE：真；
 - DH_DEFAULT：用于在 API 调用时由 CDexHandClient 指定默认值。
- 日志等级枚举：DH_LL
 - DH_LL_NONE：不输出；
 - DH_LL_ERROR：仅输出错误；
 - DH_LL_WARN：输出错误和警告；
 - DH_LL_INFO：输出错误、警告和信息；
 - DH_LL_DBG：输出错误、警告、信息和调试信息；
 - DH_LL_VERB：输出全部信息，日志量极大，不建议使用。

3.4 CDexHandClient 的 API 介绍

DexHand 采用服务端-客户端(server-client)架构描述用户与 DexHand 的交互行为，并使用 UDP 协议实现服务端和客户端之间的通讯。DexHand 本体作为服务端，周期性地发送 DexHand 的状态信息，并响应客户端发送的指令；用户可以创建一个或多个客户端，向服务端发送指令请求，或接收 DexHand 的状态信息。

在交换 DexHand 状态信息和处理 DexHand 指令信息时，DexHand 使用了不同的通讯方式。服务端使用组播（Multicast）方式发送状态信息，意味着组播地址下所有的客户端均可以接收 DexHand 的状态信息。不同的是，客户端和服务端之间的指令使用单播（Unicast）方式发送，意味着只有发送请求的客户端能够接收到服务端发送的回复报文。

需要特别说明的是，为保证系统运行的安全性，服务端在运行时将依照“先到先得”的原则与一个客户端绑定，并只执行该客户端发送的指令请求，拒绝其它客户端的指令请求。也就是说，当一个客户端获得了 DexHand 的控制权限后，除非主动释放控制权限，否则其它客户端将无法控制 DexHand。但在这种情况下，其他客户端仍然能接收 DexHand 的状态信息。

另外，在无另外说明的情况下，客户端发送的所有命令都需要等待服务端回复消息确认没有丢包。部分命令需要等待服务端回复信息确认任务完成或失败，将在下文特别标出。

3.4.1 创建客户端

dexhand_ctx 类是 DexHand 的上下文指针，可以使用以下代码创建并初始化一个上下文指针 ctx（指针的名称是任意的，但在之后的说明中我们均使用 ctx 表示一个上下文指针）：

```
#include "dexhand_client.h"
dexhand_ctx *ctx;
dh_dexhand_client(&ctx, "192.168.2.100", 60031, NULL, NULL);
```

其中，ip 和 port 为 DexHand 服务端的 ip 地址和端口，上面的代码中采用了 DexHand 的出厂默认值。dh_dexhand_client 函数定义如下：

```
int32_t dh_dexhand_client(dexhand_ctx **ctx, const char *ip, uint16_t port,
const term_callback term_cb, const data_callback hand_cb);
```

初始化后可使用上下文 ctx 实现 DexHand 的控制和信息读取，具体接口见下文各小节。初始化函数的参数如下表，其中回调函数的使用方法见前文 3.3.4 和后文 3.4.8。

表 3.1 dh_dexhand_client 的初始化参数

| 参数名称 | 数据类型 | 默认值 | 说明 |
|---------|---------------|-----|--------------|
| ctx | dexhand_ctx** | 必填 | 机械手客户端上下文 |
| ip | const char* | 必填 | 服务端 ip |
| port | uint16_t | 必填 | 服务端端口 |
| term_cb | term_callback | 必填 | 处理系统信号的回调函数 |
| hand_cb | data_callback | 必填 | 处理机械手数据的回调函数 |

3.4.2 启动与停止服务

在 DexHand 接通电源后，服务端已经自动启动，但此时尚未开始提供服务。具体而言，DexHand 尚未开始发送状态信息，也不能接收运动指令。为此，需要客户端使用 dh_start_server 指令请求服务端开启服务。函数定义如下：

```
int32_t dh_start_server(dexhand_ctx *ctx);
```

服务开启后，服务端会周期性发送 DexHand 的数据，并准备接收客户端的任务请求。服务顺利开启后便会持续运行，与客户端是否存在无关。


 注意：服务端开启服务需要等待大约 7s 左右，因此除非打算为 DexHand 关机，否则一般不需要在程序运行中关闭服务端的服务，让其保持开启状态即可；当服务端服务已经处于开启状态时，再次运行 start_sevrer 指令不会重新等待。

表 3.2 dh_start_server 函数说明

| 函数说明 | 尝试开启服务端服务 | | |
|----------|--------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 |

与开启服务对应的, 可以使用 dh_stop_server 指令请求服务端停止服务。

```
int32_t dh_stop_server(dexhand_ctx *ctx);
```

调用 dh_stop_server 时, 会停止机械手当前的运动任务, 并释放 DexHand 的控制权, 不再被某一个客户端占据。随后, 停止 DexHand 的所有运算, 不再发送 DexHand 的数据。

表 3.3 dh_stop_server 函数说明

| 函数说明 | 尝试关闭服务端服务 | | |
|----------|--------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 |

3.4.3 获取与释放控制权限

尽管开启服务端服务后, 任意一个客户端均能够读取 DexHand 的状态信息, 但为了保障安全, 任何时候只有一个客户端能够控制 DexHand。当客户端申请并获得 DexHand 的控制权限后, 服务端便只会响应该客户端的指令, 直到该客户端释放 DexHand 的控制权限。

当 DexHand 的控制权限没有被任何客户端占据时, 客户端可以使用 dh_acquire_hand 函数获取 DexHand 的控制权。在调用该指令后, 若 DexHand 尚未进行过初始化, 则会进行一次初始化, 即校准位置零点(dh_set_home)和力零点(dh_calibrate_force_zero), 机械手会执行回零动作, 此过程中机械手不可受到来自外部环境的干扰, 否则可能会造成校零失败。对零点校正函数的介绍见 3.4.5 小节

```
int32_t dh_acquire_hand(dexhand_ctx *ctx);
```

表 3.4 dh_acquire_hand 函数说明

| 函数说明 | 尝试获取 DexHand 控制权限 | | |
|----------|-------------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 |

当客户端暂时不需要使用 DexHand 时, 应该使用 dh_release_hand 函数释放

DexHand 的控制权。

```
int32_t dh_release_hand(dexhand_ctx *ctx);
```

表 3.5 dh_release_hand 函数说明

| 函数说明 | 尝试释放 DexHand 控制权限 | | |
|----------|-------------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 |

⚠ 在程序正常或意外（例如通过键盘输入 Ctrl+C 中断程序运行、在 IDE 内调试时点击终止按钮）退出时，机械手会自动释放控制权限，再次调用程序时需要重新获取，因此建议在所有程序的开头（启动服务器指令后），默认加一行获取控制权限的代码。此外 SDK 中还提供了主动释放控制权限的函数：dh_release_hand，在程序执行过程中，客户端如果需要中途释放控制权限，则可调用该函数。

3.4.4 停止机械手

dh_halt 函数可以取消 DexHand 所有待执行的任务，放弃正在执行的任务，并关闭 DexHand 的电机。

```
int32_t dh_halt(dexhand_ctx *ctx);
```

表 3.6 dh_halt 函数说明

| 函数说明 | 尝试停止 DexHand | | |
|----------|--------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 |

⚠ 此外，当程序收到系统信号时，dh_halt 函数和 dh_release_hand 函数会自动被执行，DexHand 会即刻停止。利用这一机制，如果需要让 DexHand 急停，可以通过使用 Ctrl+C 触发系统信号来实现。

3.4.5 位置校准与力校准

DexHand 本体数据中包括当前的位置和抓取力，若这些数据的零点出现偏差，可以使用以下函数校准。

dh_set_home 通过使 DexHand 张开到最大位置，校准夹爪的位置零点。

```
int32_t dh_set_home(dexhand_ctx *ctx, double goal_speed);
```

表 3.7 dh_set_home 函数说明

| 函数说明 | | 尝试校准 DexHand 的位置零点 | | | |
|------------|--------------|--------------------|-----|-----|-------------------|
| 是否等待任务完成 | | 是 | 返回值 | | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| ctx | dexhand_ctx* | 必填 | / | / | 机械手客户端上下文 |
| goal_speed | double | 4.0 | 8.0 | 1.0 | 回零速度, 单位 mm/s |

dh_calibrate_force_zero 将 DexHand 当前接触力的测量值设置为力零点, 在调用此函数时, 需保证手指未受到任何外界干扰, 否则会导致抓取力控制出错。

```
int32_t dh_calibrate_force_zero(dexhand_ctx *ctx);
```

表 3.8 dh_calibrate_force_zero 函数说明

| 函数说明 | | 尝试校准 DexHand 的力零点 | | |
|----------|--------------|-------------------|-----------|-------------------|
| 是否等待任务完成 | | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认 | 说明 | |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 | |

在调用权限获取指令 dh_acquire_hand 时, 若 DexHand 尚未进行过初始化, 则会自动调用位置零点和力零点校准函数各一次。

3.4.6 位置控制

CDexHandClient 提供两个函数进行位置控制, 分别是 dh_pos_goto 和 dh_pos_servo。

dh_pos_goto 函数将使 DexHand 以给定的最大速度和最大加速度自动规划轨迹运动到指定位置。同时, dh_pos_goto 函数需要指定运动过程中的最大力, 当运动过程中检测到接触力大于给定的最大力, DexHand 将以最大驱动能力减速至静止。换言之, 该函数将在“DexHand 到达指定位置”和“接触力大于给定最大力”这两个条件任一满足时退出。

```
int32_t dh_pos_goto(dexhand_ctx *ctx, double goal_pos, double max_speed, double max_acc, double max_f);
```

表 3.9 dh_pos_goto 函数说明

| 函数说明 | | 尝试运动到指定位置 | | | |
|----------|--------------|-----------|------|-----|-------------------|
| 是否等待任务完成 | | 是 | 返回值 | | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| ctx | dexhand_ctx* | 必填 | / | / | 机械手客户端上下文 |
| goal_pos | double | 必填 | 50.0 | 0.0 | 目标位置(mm) |

| | | | | | |
|-----------|--------|------|-------|-----|---------------------------|
| max_speed | double | 16.0 | 110.0 | 0.1 | 最大速度(mm/s) |
| max_acc | double | 20.0 | 500.0 | 0.1 | 最大加速度(mm/s ²) |
| max_f | double | 1.0 | 20.0 | 0.0 | 最大接触力(N) |

dh_pos_servo 函数将使 DexHand 接近最大驱动能力的最大速度 (110mm/s) 和最大加速度(500mm/s²)规划轨迹, 向指定的位置运动。dh_pos_servo 同样需要指定运动过程中的最大接触力, 当运动过程中检测到接触力大于给定的最大力, DexHand 将以最大驱动能力减速至静止。

与 dh_pos_goto 不同的是, dh_pos_servo 函数在确认服务端确认收到后即刻退出, 而不等待 DexHand 完成运动。这意味着 dh_pos_servo 允许用户在 DexHand 尚未到达指定位置时更改目标位置。

```
int32_t dh_pos_servo(dexhand_ctx *ctx, double goal_pos, double max_f);
```

表 3.10 dh_pos_servo 函数说明

| 函数说明 | | 设置位置伺服目标 | | | |
|----------|--------------|----------|------|-----|-------------------|
| 是否等待任务完成 | | 否 | 返回值 | | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| ctx | dexhand_ctx* | 必填 | / | / | 机械手客户端上下文 |
| goal_pos | double | 必填 | 50.0 | 0.0 | 目标位置(mm) |
| max_f | double | 1.0 | 20.0 | 0.0 | 最大接触力(N) |

3.4.7 抓取力控制

抓取力控制要求 DexHand 必须已经接触了物体。因此 DexHandClient 首先提供了 dh_contact 函数控制 DexHand 接触物体。dh_contact 函数将以给定的速度闭合 DexHand 夹爪, 并在接触到物体后施加指定的预载力。需要特别说明的是, 如果预先知道某一个位置一定不会接触上物体, 则通过 dh_contact 函数可以先控制 DexHand 快速运动至该位置, 再进行上述接触过程。

```
int32_t dh_contact(dexhand_ctx *ctx, double contact_speed, double preload_force, double quick_move_speed, double quick_move_pos);
```

表 3.11 dh_contact 函数说明

| 函数说明 | | 尝试接触物体 | | | |
|----------|------|--------|-----|-----|-------------------|
| 是否等待任务完成 | | 是 | 返回值 | | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |

| | | | | | |
|------------------|--------------|-----|------|-----|-------------|
| ctx | dexhand_ctx* | 必填 | / | / | 机械手客户端上下文 |
| contact_speed | double | 8.0 | 25.0 | 0.1 | 接触速度(mm/s) |
| preload_force | double | 1.0 | 20.0 | 0.0 | 最大接触力(N) |
| quick_move_pos | double | NAN | 50.0 | 0.1 | 预运动位置(mm) |
| quick_move_speed | double | NAN | 40.0 | 0.0 | 预运动速度(mm/s) |

注意, quick_move_pos 和 quick_move_speed 可以指定为 NAN。但只要需要指定这两个参数就必须同时指定, 且 quick_move_speed 不应小于 contact_speed。例如下面的例子中, 前两条都是可以接受的指令, 第三条可以运行, 但预运动速度会上调至 contact_speed, 第四、五条则按照第一条处理。

```
dh_contact(ctx, 8.0, 1.0, NAN, NAN)
dh_contact(ctx, 8.0, 1.0, 30.0, 20.0)
dh_contact(ctx, 8.0, 1.0, 4.0, 20.0)//预运动速度上调至 8mm/s
dh_contact(ctx, 8.0, 1.0, 16.0, NAN)//未输入预运动速度, 等效为第一条指令
dh_contact(ctx, 8.0, 1.0, NAN, 20.0)//未输入预运动位置, 等效为第一条指令
```

在此基础上, DexHandClient 提供了两个函数 dh_grasp 和 dh_force_servo 进行抓取力控制, 当客户端调用 dh_grasp 或 dh_force_servo 函数, 但 DexHand 尚未接触物体时, 服务端会按默认值自动调用 dh_contact。

dh_grasp 函数将使用给定时间线性地将抓取力变为给定值。当抓取力稳定在给定值上时 dh_grasp 函数才会返回。

表 3.12 dh_grasp 函数说明

| 函数说明 | | 线性改变抓取力 | | | |
|------------|--------------|---------|------|-----|-------------------|
| 是否等待任务完成 | | 是 | 返回值 | | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| ctx | dexhand_ctx* | 必填 | / | / | 机械手客户端上下文 |
| goal_force | double | 必需 | 20.0 | 0.0 | 目标力(N) |
| load_time | double | 1.0 | / | 0.0 | 加载时间(s) |

dh_force_servo 函数将改变 DexHand 抓取力的目标值, 此时 DexHand 将以最大跟随能力达到这一抓取力。dh_force_servo 函数在确认服务端确认收到后即刻退出, 而不等待 DexHand 达到指定抓取力。这意味着用户可以在 DexHand 尚未达到指定抓取力时改变目标抓取力。

表 3.13 dh_force_servo 函数说明

| 函数说明 | | 设置力伺服目标 | | | |
|----------|--|---------|-----|--|-------------------|
| 是否等待任务完成 | | 是 | 返回值 | | int32_t, 参考 3.3.2 |

| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
|------------|--------------|----|------|-----|-----------|
| ctx | dexhand_ctx* | 必填 | / | / | 机械手客户端上下文 |
| goal_force | double | 必需 | 20.0 | 0.0 | 目标力(N) |

下面是抓取力控制函数的使用示例：

```
dh_grasp(ctx, 5.0, 1.0) // 抓取力到达 5N 时返回
dh_force_servo(ctx, 6.0) // 确认服务端收到后即刻返回
```

3.4.8 DexHand 本体信息读取

只要服务端开始提供服务，且客户端存在，则客户端就会不断接收 DexHand 的本体信息，更新频率为 100Hz。使用 DexHandClient 的成员 hand_info 可以获取最近一帧 DexHand 本体信息。hand_info 本身又包括以下成员：

- frame_cnt: 数据包计数, uint32_t 类型;
- time: 数据包生成的 UNIX 时间 (自 1970 年 1 月 1 日 00:00:00 UTC 起), 单位为 s;
- now_pos: DexHand 当前位置, double 类型, 单位为 mm;
- goal_pos: DexHand 运动目标位置, double 类型, 单位为 mm;
- now_speed: DexHand 当前速度, double 类型, 单位为 mm/s;
- goal_speed: DexHand 目标速度, double 类型, 单位为 mm/s;
- now_current: DexHand 内部电机当前电流, double 类型, 单位为 mA;
- goal_current: DexHand 内部电机当前电流, double 类型, 单位为 mA;
- now_force[2]: DexHand 两个指根处的力传感器测力值, double 类型, 单位 N;
- avg_force: DexHand 两个力传感器的平均测力值, double 类型, 单位 N;
- goal_force: DexHand 当前目标接触力, double 类型, 单位 N;
- imu_acc[3]: DexHand 内部 IMU 所测加速度 (含重力加速度), double 类型, 单位 m/s^2 ;
- imu_gyr[3]: DexHand 内部 IMU 所测的角速度, double 类型, 单位为 $^{\circ}/s$;
- is_contact[2]: 基于力传感器判断手指是否接触物体, DH_BOOL 类型;
- error_flag: 当前 DexHand 是否在错误状态, DH_BOOL 类型;
- now_task: DexHand 正在运行中的任务, char[] 类型;
- recent_task: DexHand 最近 (正在处理或已经结束) 的任务, char[] 类型;
- recent_task_status: DexHand 最近的任务状态, str[] 类型, 分为 RUNNING (正在运行), FINISHED (正常结束) 和 ABORTED (异常结束)。

其中, now_task 和 recent_task 与任务的对应关系如下:

表 3.14 任务字符串与其含义的对照表


| 任务字符串 | 对应含义 |
|------------|-----------------------|
| NORMAL | 无任务，任务结束后回到此状态 |
| STOP | 电机停止（有错误等待处理） |
| CONTACT | 接触物体 |
| SETHOME | 校准位置零点 |
| FORCECALI | 校准力零点 |
| GOTO | 运行至指定位置（由 goto 触发） |
| SETFORCE | 加载至指定抓取力（由 grasp 触发） |
| POSSERVO | 位置伺服（由 pos_servo 触发） |
| FORCESERVO | 力伺服（由 force_servo 触发） |

此外, CDexHandClient 提供了回调函数功能。当 DexHandClient 接收到一帧 DexHand 本体信息后, 会自动执行回调函数内的指令。该回调函数由用户自行编写, 且以上下文的指针作为参数。下面是一个回调函数的实现案例, 它在每次接收到 DexHand 本体信息后输出此刻 DexHand 的位置。

```
void hand_cb(const dexhand_ctx *ctx)
{
    // 获取 DexHand 数据并输出序号及时间
    const hand_info *data = dh_get_hand_data(ctx);
    printf("[%u] %lf\n", data->frame_cnt, data->time);
}
// 中间省略
dh_init_ctx(&ctx, "192.168.2.100", 60031, NULL, hand_cb);
```

3.4.9 Tac3D 信息读取

在 V1.0 的 SDK 版本中, Tac3D 与机械手的使用是完全解耦的, 因此获取 Tac3D 信息的方法可参考 Tac3D 的 SDK 二次开发手册, 下面的示例仅做参考, 请仔细阅读 Tac3D 的 SDK 开发手册后再行使用。

 注意: 在调用启动服务端函数 dh_start_server 时, 会同步启动 Tac3D 传感器, 在启动后请尽量不要将 Tac3D 从机械手上拆除; 倘若在启动传感器后因特殊原因从机械手上拆除了传感器, 则需要首先停止服务端 (dh_stop_server), 并再次启动服务端 (dh_start_server) 才能重新启动 Tac3D 传感器。

首先引用头文件。

```
#include "libTac3D.hpp"
```

声明用于存储接收到的数据的用户变量

```
cv::Mat P, D, F, Fr, Mr; // 用于存储三维形貌、三维变形场、三维分布力、三维合力、三维合力矩数据的矩阵
int frameIndex; // 帧序号
double sendTimestamp, recvTimestamp; // 时间戳
std::string SN; // 传感器 SN
```

编写接收触觉数据帧的回调函数：

```
void Tac3DRecvCallback(Tac3D::Frame &frame, void *param)
{
    cv::Mat *tempMat;
    // float* testParam = (float*)param; // 接收自定义参数（在 Tac3D::Sensor
    初始化时传递）
    SN = frame.SN; // 获得传感器 SN 码，可用于区分触觉信息来源于哪个触觉
    传感器
    frameIndex = frame.index; // 获得帧序号
    sendTimestamp = frame.sendTimestamp; // 获得发送时间戳
    recvTimestamp = frame.recvTimestamp; // 获得接收时间戳

    // 使用 frame.get 函数通过数据名称"3D_Positions"获得 cv::Mat 类型的三
    维形貌的数据指针
    tempMat = frame.get<cv::Mat>("3D_Positions");
    tempMat->copyTo(P); // 务必先将获得的数据拷贝到自己的变量中再使用
    （注意，OpenCV 的 Mat 中数据段和矩阵头是分开存储的，因此需要使用 copyTo
    同时复制矩阵的矩阵头和数据段，而不应当用=符号赋值）

    // 使用 frame.get 函数通过数据名称"3D_Displacements"获得 cv::Mat 类型
    的三维变形场的数据指针
    tempMat = frame.get<cv::Mat>("3D_Displacements");
    tempMat->copyTo(D); // 务必先将获得的数据拷贝到自己的变量中再使用

    // 使用 frame.get 函数通过数据名称"3D_Forces"获得 cv::Mat 类型的三维分
    布力的数据指针
```

```

tempMat = frame.get<cv::Mat>("3D_Forces");
tempMat->copyTo(F); // 务必先将获得的数据拷贝到自己的变量中再使用

// 使用 frame.get 函数通过数据名称"3D_ResultantForce"获得 cv::Mat 类型
的三维合力的数据指针
tempMat = frame.get<cv::Mat>("3D_ResultantForce");
tempMat->copyTo(Fr); // 务必先将获得的数据拷贝到自己的变量中再使用

// 使用 frame.get 函数通过数据名称"3D_ResultantForce"获得 cv::Mat 类型
的三维合力的数据指针
tempMat = frame.get<cv::Mat>("3D_ResultantMoment");
tempMat->copyTo(Mr); // 务必先将获得的数据拷贝到自己的变量中再使用
}

```

⚠ 注意：不应将对机器人系统的控制逻辑或其他耗时的程序放在回调函数中执行。否则可能因触觉数据帧处理不及时导致数据堆积和严重的数据滞后。

在 main 函数中初始化 Tac3D 实例启动数据接收：

```

int main(int argc, char **argv)
{
    float testParam = 100.0;
    Tac3D::Sensor tac3d(Tac3DRecvCallback, 9988, & testParam); // 创建
    Sensor 实例，设置回调函数为上面写好的 Tac3DRecvCallback，设置 UDP 接收端
    口为 9988，设置自定义参数为 testParam
    tac3d.waitForFrame(); // 等待 Tac3D-Desktop 端启动传感器并建立连接

    usleep(1000*1000*5); // 5s
    tac3d.calibrate(SN); // 发送一次校准信号（应确保校准时传感器未与任何物
    体接触！否则会输出错误的数据！）

    // 例程中没有其他需要执行的任务，故使用 while 循环阻止主程序退出
    while (1)
    {
        usleep(1000*1000);
    }
    return 0;
}

```

3.4.10 错误消除

如果 DexHand 抓取力过大, 或 DexHand 电机电流过大, 将会触发 DexHand 的安全保护系统, 使 DexHand 急停并进入错误状态。此时, hand_info 中的 error_flag 将为 True, DexHand 不能执行任何位置控制、力控制、校准指令。

如果确认 DexHand 已经脱离危险, 则可以使用 dh_clear_hand_error 函数清除 DexHand 的错误状态。

```
int32_t dh_clean_hand_error(dexhand_ctx *ctx);
```

表 3.15 dh_clear_hand_error 函数说明

| 函数说明 | 尝试清除 DexHand 错误状态 | | |
|----------|-------------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| ctx | dexhand_ctx* | 必填 | 机械手客户端上下文 |

3.4.11 日志设置

默认情况下 CDexHandClient 不输出任何日志, 若用户需要输出日志, 则可使用 dh_set_log 函数。

```
int32_t dh_set_log(const char *fn, DH_LL log_level);
```

表 3.16 dh_set_log 函数说明

| 函数说明 | 输出 DexHand 日志 | | |
|-----------|--------------------|-----|-------------------|
| 是否等待任务完成 | 是 | 返回值 | int32_t, 参考 3.3.2 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| fn | const char* | 必填 | 日志文件名称 |
| log_level | DH_LL 参考 3.4.11 | 必填 | 日志文件等级 |

其中日志文件名可以设置为 NULL, 此时日志输出文件的设置被忽略。

3.5 CDexHandClient 的例程说明

3.5.1 客户端例程 client.c

首先引用 dexhand_client.h 头文件:

```
#include "dexhand_client.h"
```

引用必要的系统头文件：

```
#include <math.h>
#include <stdio.h>

#ifdef _WIN32      //检测当前系统为 windows 或 Linux
#include <windows.h>

void
sleep_sec(double time)
{
    uint32_t msec = round(time * 1e3);
    Sleep(msec);
}
#else
#include <unistd.h>

void
sleep_sec(double time)
{
    uint32_t usec = round(time * 1e6);
    usleep(usec);
}
#endif
```

回调函数的声明：

```
void hand_cb(const dexhand_ctx *ctx);
void term_cb();
```

主函数：

```
int main(int argc, char *argv[])
{
    // 设置日志等级
```

```
dh_set_log("log.txt", DH_LL_DBG);
// 初始化上下文
dexhand_ctx *ctx;
if (dh_init_ctx(&ctx, "192.168.2.100", 60031, term_cb, hand_cb) < 0) {
    printf("Can not init context, exit\n");
    return -1;
}

// 启动 DexHand 硬件
if (dh_start_server(ctx) < 0) {
    goto stop;
}
// 获取控制权
if (dh_acquire_hand(ctx) < 0) {
    goto stop;
}
// 以默认速度移动手指位置
if (dh_pos_goto(ctx, 30.0, NAN, NAN, NAN) < 0) {
    goto stop;
}
// 力控
if (dh_force_servo(ctx, 2.0) < 0) {
    goto stop;
}
sleep_sec(1.0);
// 释放控制权
if (dh_release_hand(ctx) < 0) {
    goto stop;
}
// 再次获取控制权
if (dh_acquire_hand(ctx) < 0) {
    goto stop;
}
// 手指移动至零点
if (dh_set_home(ctx, NAN) < 0) {
```

```

        goto stop;
    }

stop:
    // 停止 DexHand
    dh_stop_server(ctx);
    // 清理上下文
    dh_release_ctx(&ctx);

    return 0;
}

```

回调函数：

```

void hand_cb(const dexhand_ctx *ctx)
{
    // 获取 DexHand 数据并输出序号及时间
    const hand_info *data = dh_get_hand_data(ctx);
    printf("[%u] %lf\n", data->frame_cnt, data->time);
}

void term_cb()
{
    // 输出信息
    printf("Got signal, exited...\n");
}

```

3.5.2 Tac3D 信息读取例程 tac3d.cpp

首先引用 dexhand_client.h 和 libTac3D.hpp 头文件：

```

#include "dexhand_client.h"

#include "libTac3D.hpp"

```

引用必要的系统头文件：

```
#include <stdio.h>
```

回调函数的声明：

```
void hand_cb(const dexhand_ctx *ctx);

void Tac3DRecvCallback(Tac3D::Frame &frame, void *param);
```

定义存储 Tac3D 数据的结构体：

```
struct Tac3D_info {
    std::string SN;
    uint32_t    frameIndex;
    double      sendTimestamp;
    double      recvTimestamp;
    cv::Mat     P;
    cv::Mat     D;
    cv::Mat     F;
    cv::Mat     Fr;
    cv::Mat     Mr;

    Tac3D_info(std::string _SN)
    {
        SN          = _SN;
        frameIndex = 0xFFFFFFFF;
    }
};
```

定义 Tac3D 数据的全局变量：

```
Tac3D_info tacinfo1("HDL1-0001");
Tac3D_info tacinfo2("HDL1-0002");
// 字典
std::map<std::string, Tac3D_info*> tac_dict;
```


主函数:

```
int main(int argc, char *argv[])
{
    // 设置日志等级
    dh_set_log("log.txt", DH_LL_DBG);
    // 初始化上下文
    dexhand_ctx *ctx;
    if (dh_dexhand_client(&ctx, "192.168.2.100", 60031, NULL, hand_cb) < 0)
    {
        printf("Can not init context, exit\n");
        return -1;
    }
    // 将 Tac3D 数据加入字典
    tac_dict.insert({ tacinfo1.SN, &tacinfo1 });
    tac_dict.insert({ tacinfo2.SN, &tacinfo2 });
    // 启动 Tac3D 客户端
    auto tac3d = Tac3D::Sensor(Tac3DRecvCallback, 9988, NULL);
    // 启动 DexHand 硬件
    if (dh_start_server(ctx) < 0) {
        goto stop;
    }
    // 获取控制权
    if (dh_acquire_hand(ctx) < 0) {
        goto stop;
    }
    // 设置位置零点
    if (dh_set_home(ctx, NAN) < 0) {
        goto stop;
    }
    // 设置力零点
    if (dh_calibrate_force_zero(ctx) < 0) {
        goto stop;
    }
    // 为全部 Tac3D 校准
```

```

    for (auto &pair : tac_dict) {
        tac3d.calibrate(pair.first);
    }
    // 两指接触
    if (dh_contact(ctx, 8.0, 2.0, 15.0, 10.0) < 0) {
        goto stop;
    }
    // 施加接触力
    if (dh_grasp(ctx, 5.0, 5.0) < 0) {
        goto stop;
    }
    // 释放控制权
    if (dh_release_hand(ctx) < 0) {
        goto stop;
    }

stop:
    // 停止 DexHand
    dh_stop_server(ctx);

    return 0;
}

```

回调函数：

```

void hand_cb(const dexhand_ctx *ctx)
{
    // 获取 DexHand 数据
    const hand_info *data = dh_get_hand_data(ctx);
    // 检查是否收到数据
    if (tacinfo1.frameIndex == 0xFFFFFFFF || tacinfo2.frameIndex ==
0xFFFFFFFF) {
        return;
    }

    // 间隔 10 帧输出信息

```

```

        if (data->frame_cnt % 10 == 0) {
            printf("Error:{%s},    nowforce1:    %.3fN    nowforce2:    %.3fN
nowTacFz1: %.3fN nowforce2: %.3fN nowpos: %.3fmm ",
                data->error_flag == DH_TRUE ? "True" : "False",
                data->now_force[0], data->now_force[1],
                tacinfo1.Fr.at<double>(0, 2), tacinfo2.Fr.at<double>(0, 2),
                data->now_pos);
        }
    }

void Tac3DRecvCallback(Tac3D::Frame &frame, void *param)
{
    cv::Mat *tempMat;

    // 获取 SN
    std::string SN = frame.SN;

    // 在字典里检查 SN
    if (tac_dict.find(SN) == tac_dict.end()) {
        return;
    }

    // 根据 SN 存储数据
    auto tacinfo = tac_dict[SN];

    // 获取帧序号
    tacinfo->frameIndex = frame.index;

    // 获取时间戳
    tacinfo->sendTimestamp = frame.sendTimestamp;
    tacinfo->recvTimestamp = frame.recvTimestamp;

    // 获取标志点三维形貌
    tempMat = frame.get<cv::Mat>("3D_Positions");
    tempMat->copyTo(tacinfo->P);
}

```

```
// 获取标志点三维位移场
tempMat = frame.get<cv::Mat>("3D_Displacements");
tempMat->copyTo(tacinfo->D);

// 获取三维分布力
tempMat = frame.get<cv::Mat>("3D_Forces");
tempMat->copyTo(tacinfo->F);

// 获得三维合力
tempMat = frame.get<cv::Mat>("3D_ResultantForce");
tempMat->copyTo(tacinfo->Fr);

// 获得三维合力矩
tempMat = frame.get<cv::Mat>("3D_ResultantMoment");
tempMat->copyTo(tacinfo->Mr);
}
```

4 基于 DexHand-Python SDK 的二次开发

本章节将介绍基于 Python 编写的 DexHand SDK 的 API 接口的使用方法。

4.1 环境配置

4.1.1 安装 dexhand_client 包

以下环境配置适用于 Windows 用户和 Linux 用户。在确保电脑安装了 Python (版本不低于 3.8) 后, 在解压 SDK 文件所得到的“pyDexHandClient”文件夹下打开一个终端(也称作“命令提示符”“命令行窗口”等。在 Windows 下, 这可以通过在文件夹窗口上侧地址栏处输入“cmd”实现; 在 Linux 下, 可以在右键菜单中找到选项; 如果使用 VSCode 等开发工具, 请参阅对应开发工具的使用说明)。输入(注意末尾的点不可忽略):

```
pip install .
```

此时, 如果运行正常, 将会在终端末尾看到“Successfully installed dexhand_client-1.0”字样(1.0 为版本号, 可能不同), 说明安装成功。如果看到除了“dexhand_client-1.0”之外, 还安装了其它的包, 例如“numpy”“colorlog”“colorama”(其后版本号省略), 说明安装过程同时安装了部分运行 dexhand_client 包必要的, 但之前尚未安装的包, 属于正常现象。

同时, 可以看到“pyDexHandClient”文件夹下新生成了两个文件夹: “build”和“dexhand_client.egg-info”。这两个文件夹对 dexhand_client 包的使用无影响。

4.1.2 检验 dexhand_client 包是否正确安装

可以使用以下方法检验 dexhand_client 包是否正确安装: 新建任意一个 Python 脚本(例如“test_install.py”), 在其中输入以下内容:

```
from dexhand_client import TestClient
client = TestClient()
```

随后在 Python 脚本所在文件夹下打开一个终端, 运行这一脚本:

```
python your_script.py
```

其中, 需要将“your_script”替换为脚本的文件名, 例如上面举例的“test_install”。如果包安装正确, 终端会仅输出一行“dexhand-client has been installed successfully.”后正常结束。注意, 在部分系统中, 需要将“python”字段替换为“python3”。

在后续文档中, 如无特殊说明, 我们使用如下方式运行一个 Python 脚本:

- dexhand_client 文件夹: 其中包含了 dexhand_client 包的核心内容, 即接口的具体实现, 以及 SDK 使用的配置文件, 与 C 语言版本一致。
- 在脚本所在文件夹打开一个终端;

- 使用“python <脚本名>.py”的方式运行脚本。

4.1.3 使用 conda 的情况

如果你使用了 conda 等 Python 环境管理工具, 请确保你在安装前正确地启用了你所希望使用的环境, 并在每次使用 dexhand_client 包时启用了它被安装的目标环境。其余操作和上文所述相同。

4.2 PyDexHand 的 API 介绍

DexHand 采用服务端-客户端(server-client)架构描述用户与 DexHand 的交互行为, 并使用 UDP 协议实现服务端和客户端之间的通讯。DexHand 本体作为服务端, 周期性地发送 DexHand 的状态信息, 并响应客户端发送的指令; 用户可以创建一个或多个客户端, 向服务端发送指令请求, 或接收 DexHand 的状态信息。

在交换 DexHand 状态信息和处理 DexHand 指令信息时, DexHand 使用了不同的通讯方式。服务端使用组播(Multicast)方式发送状态信息, 意味着组播地址下所有的客户端均可以接收 DexHand 的状态信息。不同的是, 客户端和服务端之间的指令使用单播(Unicast)方式发送, 意味着只有发送请求的客户端能够接收到服务端发送的回复报文。

需要特别说明的是, 为保证系统运行的安全性, 服务端在运行时将依照“先到先得”的原则与一个客户端绑定, 并只执行该客户端发送的指令请求, 拒绝其它客户端的指令请求。也就是说, 当一个客户端获得了 DexHand 的控制权限后, 除非主动释放控制权限, 否则其它客户端将无法控制 DexHand。但在这种情况下, 其他客户端仍然能接收 DexHand 的状态信息。

另外, 在无另外说明的情况下, 客户端发送的所有命令都需要等待服务端回复消息确认没有丢包。部分命令需要等待服务端回复信息确认任务完成或失败, 将在下文特别标出。

4.2.1 创建客户端

DexHandClient 是 DexHand 的客户端类, 可以使用以下代码导入 DexHandClient 并创建一个客户端实例 client (实例的名称是任意的, 但在之后的说明中我们均使用 client 表示一个 DexHandClient 实例):

```
from dexhand_client import DexHandClient
client = DexHandClient(ip="192.168.2.100", port=60031)
```

其中, ip 和 port 为 DexHand 服务端的 ip 地址和端口, 上面的代码中采用了 DexHand 的出厂默认值。

创建之后可以使用 client 的成员函数实现 DexHand 的控制和信息读取, 具体接口见下文各小节。DexHandClient 类的初始化参数如下表, 其中回调函数的使用方法见后文 4.2.8。

表 4.1 DexHandClient 类的初始化参数

| 参数名称 | 数据类型 | 默认值 | 说明 |
|-------------------|------|-------|---|
| ip | str | 必填 | 服务端 ip |
| port | int | 必填 | 服务端端口 |
| recvCallback_hand | 函数 | None | 机械手信息回调函数 |
| ignore_myself | bool | False | 为 True 时，在终端只 显示 server 发送的消 息，而不显示 client 消息。 |

4.2.2 启动与停止服务

在 DexHand 接通电源后，服务端已经自动启动，但此时尚未开始提供服务。具体而言，DexHand 尚未开始发送状态信息，也不能接收运动指令。为此，需要客户端使用 start_server 指令请求服务端开启服务。具体方法如下：

```
client.start_server()
```

服务开启后，服务端会周期性发送 DexHand 的数据，并准备接收客户端的任务请求。

服务顺利开启后会持续运行，与客户端是否存在无关。因此，只要任意客户端开启了服务后，其它客户端不需要重新开启。在 pyDexHandClient/examples 文件夹中，我们提供了 activate_service.py 脚本，它调用 start_server，尝试开启服务端的服务。

⚠ 注意：服务端开启服务需要等待大约 7s 左右，因此除非打算为 DexHand 关机，否则一般不需要在程序运行中关闭服务端的服务，让其保持开启状态即可；当服务端服务已经处于开启状态时，再次运行 start_sevrer 指令不会重新等待。

表 4.2 start_server 函数说明

| 函数说明 | 尝试开启服务端服务 | | |
|----------|-----------|-----|-------------|
| 是否等待任务完成 | 是 | 返回值 | bool，表示是否成功 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| 无 | / | / | / |

与开启服务对应的，可以使用 stop_server 指令请求服务端停止服务。

```
client.stop_server()
```

调用 stop_server 时，会停止机械手当前的运动任务，并释放 DexHand 的控制权，不再被某一个客户端占据。随后，停止 DexHand 的所有运算，不再发送 DexHand 的数据。

表 4.3 stop_server 函数说明

| 函数说明 | 尝试关闭服务端服务 | | |
|----------|-----------|-----|--------------|
| 是否等待任务完成 | 是 | 返回值 | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| 无 | / | / | / |

4.2.3 获取与释放控制权限

尽管开启服务端服务后, 任意一个客户端均能够读取 DexHand 的状态信息, 但为了保障安全, 任何时候只有一个客户端能够控制 DexHand。当客户端申请并获得 DexHand 的控制权限后, 服务端便只会响应该客户端的指令, 直到该客户端释放 DexHand 的控制权限。

当 DexHand 的控制权限没有被任何客户端占据时, 客户端可以使用 acquire_hand 函数获取 DexHand 的控制权。在调用该指令后, 若 DexHand 尚未进行过初始化, 则会进行一次初始化, 即校准位置零点和力零点。对零点校正函数的介绍见 4.2.5 小节

```
client.acquire_hand()
```

表 4.4 acquire_hand 函数说明

| 函数说明 | 尝试获取 DexHand 控制权限 | | |
|----------|-------------------|-----|--------------|
| 是否等待任务完成 | 是 | 返回值 | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| 无 | / | / | / |

当客户端暂时不需要对 DexHand 进行控制时, 可以使用 release_hand 函数释放 DexHand 的控制权, 以便其他用户控制机械手。

```
client.release_hand()
```

表 4.5 release_hand 函数说明

| 函数说明 | 尝试释放 DexHand 控制权限 | | |
|----------|-------------------|-----|--------------|
| 是否等待任务完成 | 是 | 返回值 | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| 无 | / | / | / |

⚠ 在程序正常或意外（例如通过键盘输入 Ctrl+C 中断程序运行、在 IDE 内调试时点击终止按钮）退出时, 机械手会自动释放控制权限, 再次调用程序时需要重新获取, 因此建议在所有程序的开头（启动服务器指令后）, 默认加一行获取控制权限的代码。此外 SDK 中还提供了主动释放控制权限的函数: release_hand, 在程序执行过程中, 客户端如果需要中途

释放控制权限，则可调用该函数。

4.2.4 停止机械手

halt 函数可以取消 DexHand 所有待执行的任务，放弃正在执行的任务，并关闭 DexHand 的电机。

```
client.halt()
```

表 4.6 halt 函数说明

| 函数说明 | | 尝试停止 DexHand | |
|----------|------|--------------|--------------|
| 是否等待任务完成 | 是 | 返回值 | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| 无 | / | / | / |

⚠ 当一个 Python 脚本出现任意未被捕获的异常，导致程序运行终止时，halt 函数和 release_hand 函数会自动被执行，DexHand 会即刻停止。利用这一机制，如果需要让 DexHand 急停，可以通过使用 Ctrl+C 触发 KeyboardInterrupt 来实现。

4.2.5 位置校准与力校准

DexHand 本体数据中包括当前的位置和抓取力，若这些数据的零点出现偏差，可以使用以下函数校准。

set_home 通过使 DexHand 张开到最大位置，校准夹爪的位置零点。

```
client.set_home(goal_speed=4.0)
```

表 4.7 set_home 函数说明

| 函数说明 | | 尝试校准 DexHand 的位置零点 | | | |
|------------|-------|--------------------|-----|-----|---------------|
| 是否等待任务完成 | | 是 | 返回值 | | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| goal_speed | float | 4.0 | 8.0 | 1.0 | 回零速度, 单位 mm/s |

calibrate_force_zero 将 DexHand 当前接触力的测量值设置为力零点，在调用此函数时，需保证手指未受到任何外界干扰，否则会导致抓取力控制出错。

```
client.calibrate_force_zero()
```

表 4.8 calibrate_force_zero 函数说明

| 函数说明 | 尝试校准 DexHand 的力零点 |
|------|-------------------|
|------|-------------------|

| 是否等待任务完成 | | 是 | 返回值 | | bool, 表示是否成功 |
|----------|------|----|-----|-----|--------------|
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| 无 | / | / | / | / | / |

在调用权限获取指令 `acquire_hand` 时, 若 `DexHand` 尚未进行过初始化, 则会自动调用位置零点和力零点校准函数各一次。

4.2.6 位置控制

`DexHandClient` 提供两个函数进行位置控制, 分别是 `pos_goto` 和 `pos_servo`。

`pos_goto` 函数将使 `DexHand` 以给定的最大速度和最大加速度自动规划轨迹运动到指定位置。同时, `pos_goto` 函数需要指定运动过程中的最大力, 当运动过程中检测到接触力大于给定的最大力, `DexHand` 将以最大驱动能力减速至静止。

该函数将在 `DexHand` 到达指定位置, 接触力大于给定最大力, 或者触发安全保护时退出。例如下面是调用 `pos_goto` 函数的一个例子, `print` 语句在触发 `pos_goto` 退出条件时才会退出。前两种情况下均认为运动成功, 函数返回值为 `True`。

```
client.pos_goto(goal_pos=20.0,max_speed=16.0,max_acc=30.0,max_f=1.0)
print('pos_goto has finished.') # DexHand 结束运动后输出
```

表 4.9 `pos_goto` 函数说明

| 函数说明 | | 尝试运动到指定位置 | | | |
|-----------|-------|-----------|-------|-----|---------------------------|
| 是否等待任务完成 | | 是 | 返回值 | | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| goal_pos | float | 必需 | 50.0 | 0.0 | 目标位置(mm) |
| max_speed | float | 16.0 | 110.0 | 0.1 | 最大速度(mm/s) |
| max_acc | float | 20.0 | 500.0 | 0.1 | 最大加速度(mm/s ²) |
| max_f | float | 1.0 | 20.0 | 0.0 | 最大接触力(N) |

`pos_servo` 函数将使 `DexHand` 接近最大驱动能力的最大速度 (110mm/s) 和最大加速度(500mm/s²)规划轨迹, 向指定的位置运动。`pos_servo` 同样需要指定运动过程中的最大接触力, 当运动过程中检测到接触力大于给定的最大力, `DexHand` 将以最大驱动能力减速至静止。

与 `pos_goto` 不同的是, `pos_servo` 函数在确认服务端确认收到后即刻退出, 而不等待 `DexHand` 完成运动。例如下面是调用 `pos_servo` 函数的一个例子, `print` 语句在客户端发送指令, 并接收到服务端的确认报文后即刻执行。这意味着 `pos_servo` 允许用户在 `DexHand` 尚未到达指定位置时更改目标位置。

```
client.pos_servo(goal_pos=20.0,max_f=1.0)
```

```
print('pos_servo has finished.') # 确认服务端收到指令后输出
```

表 4.10 pos_servo 函数说明

| 函数说明 | | 设置位置伺服目标 | | | |
|----------|-------|----------|------|-----|----------|
| 是否等待任务完成 | | 否 | 返回值 | | 无 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| goal_pos | float | 必需 | 50.0 | 0.0 | 目标位置(mm) |
| max_f | float | 1.0 | 20.0 | 0.0 | 最大接触力(N) |

4.2.7 抓取力控制

抓取力控制要求 DexHand 必须已经接触了物体。因此 DexHandClient 首先提供了 contact 函数控制 DexHand 接触物体。contact 函数将以给定的速度闭合 DexHand 夹爪, 并在接触到物体后施加指定的预载力。需要特别说明的是, 如果预先知道某一个位置一定不会接触上物体, 则通过 contact 函数可以先控制 DexHand 快速运动至该位置, 再进行上述接触过程。

表 4.11 contact 函数说明

| 函数说明 | | 尝试接触物体 | | | |
|------------------|-------|--------|------|-----|-------------|
| 是否等待任务完成 | | 是 | 返回值 | | bool,表示是否成功 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| contact_speed | float | 8.0 | 25.0 | 0.1 | 接触速度(mm/s) |
| preload_force | float | 1.0 | 20.0 | 0.0 | 最大接触力(N) |
| quick_move_speed | float | None | 40.0 | 0.1 | 预运动速度(mm/s) |
| quick_move_pos | float | None | 50.0 | 0.0 | 预运动位置(mm) |

注意, quick_move_pos 和 quick_move_speed 可以不指定。但只要需要指定这两个参数就必须同时指定, 且 quick_move_speed 不应小于 contact_speed。例如下面的例子中, 前两条都是可以接受的指令, 第三条可以运行, 但预运动速度会上调至 contact_speed, 第四、五条则不是可以接受的指令。

```
# 前两个参数分别为 contact_speed 和 preload_force
client.contact(8,1)
client.contact(8,1,quick_move_speed=20.0,quick_move_pos=30.0)
client.contact(8,1,quick_move_speed=4.0,quick_move_pos=20.0)
client.contact(8,1,quick_move_speed=16.0)
client.contact(8,1,quick_move_pos=20.0)
```

在此基础上，DexHandClient 提供了两个函数 grasp 和 force_servo 进行抓取力控制，当客户端调用 grasp 或 force_servo 函数，但 DexHand 尚未接触物体时，服务端会按默认值自动调用 contact。

grasp 函数将使用给定时间线性地将抓取力变为给定值。当抓取力稳定在给定值上时 grasp 函数才会返回。

表 4.12 grasp 函数说明

| 函数说明 | | 线性改变抓取力 | | | |
|------------|-------|---------|------|-----|--------------|
| 是否等待任务完成 | | 是 | 返回值 | | bool, 表示是否成功 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| goal_force | float | 必需 | 20.0 | 0.0 | 目标力(N) |
| load_time | float | 1.0 | / | 0.0 | 加载时间(s) |

force_servo 函数将改变 DexHand 抓取力的目标值，此时 DexHand 将以最大跟随能力达到这一抓取力。force_servo 函数在确认服务端确认收到后即刻退出，而不等待 DexHand 达到指定抓取力。这意味着用户可以在 DexHand 尚未达到指定抓取力时改变目标抓取力。

表 4.13 force_servo 函数说明

| 函数说明 | | 线性改变抓取力 | | | |
|------------|-------|---------|------|-----|--------|
| 是否等待任务完成 | | 否 | 返回值 | | 无 |
| 参数名称 | 数据类型 | 默认 | max | min | 说明 |
| goal_force | float | 必需 | 20.0 | 0.0 | 目标力(N) |

下面是抓取力控制函数的使用示例：

```
# 保证在力控前已经接触上物体!
client.grasp(goal_force=5.0,load_time=1.0) # 抓取力到达 5N 时返回
client.force_servo(goal_force=6.0) # 确认服务端收到后即刻返回
```

4.2.8 DexHand 本体信息读取

只要服务端开始提供服务，且客户端存在，则客户端就会不断接收 DexHand 的本体信息，更新频率为 100Hz。使用 DexHandClient 的成员 hand_info 可以获取最近一帧 DexHand 本体信息。hand_info 本身又包括以下成员：

- now_pos: DexHand 当前位置，float 类型，单位为 mm；
- goal_pos: DexHand 运动目标位置，float 类型，单位为 mm；
- now_speed: DexHand 当前速度，float 类型，单位为 mm/s；
- goal_speed: DexHand 目标速度，float 类型，单位为 mm/s；

- now_current: DexHand 内部电机当前电流, float 类型, 单位为 mA;
- goal_current: DexHand 内部电机当前电流, float 类型, 单位为 mA;
- now_force: DexHand 两个指根处的力传感器测力值, 是长度为 2 的列表, 单位 N;
- avg_force: DexHand 两个力传感器的平均测力值, float 类型, 单位 N;
- goal_force: DexHand 当前目标接触力, float 类型, 单位 N;
- imu_acc: DexHand 内部 imu 所测加速度(含重力加速度), 是大小为 3 的 ndarray (numpy 库的数组类型), 单位 m/s²
- imu_gyr: DexHand 内部 imu 所测的角速度, 是大小为 3 的 ndarray, 单位为 dps(degree per second)
- is_contact: 基于力传感器判断手指是否接触物体, 是长度为 2 的列表, bool 类型;
- error_flag: 当前 DexHand 是否在错误状态, bool 类型;
- now_task: DexHand 正在运行中的任务, str 类型;
- recent_task: DexHand 最近 (正在处理或已经结束) 的任务, str 类型;
- recent_task_status: DexHand 最近的任务状态, str 类型, 分为 RUNNING (正在运行), FINISHED (正常结束) 和 ABORTED (异常结束)。
- _frame_cnt: DexHandClient 已经收到了多少 DexHand 本体数据帧, int 类型;
- 其中, now_task 和 recent_task 与任务的对应关系如下:

表 4.14 任务字符串与其含义的对照表

| 任务字符串 | 对应含义 |
|------------|------------------------|
| NORMAL | 无任务, 任务结束后回到此状态 |
| STOP | 电机停止 (有错误等待处理) |
| CONTACT | 接触物体 |
| SETHOME | 校准位置零点 |
| FORCECALI | 校准力零点 |
| GOTO | 运行至指定位置 (由 goto 触发) |
| SETFORCE | 加载至指定抓取力 (由 grasp 触发) |
| POSSERVO | 位置伺服 (由 pos_servo 触发) |
| FORCESERVO | 力伺服 (由 force_servo 触发) |


此外, dexhand-client 包提供了回调函数功能。当 DexHandClient 接收到一帧 DexHand 本体信息后, 会自动执行回调函数内的指令。该回调函数由用户自行编写, 且以 DexHandClient 的对象作为参数。例如, 下面是一个回调函数的实现案例, 它在每次接收到 DexHand 本体信息后输出此刻 DexHand 的位置。

```
from dexhand_client import DexHandClient
```

```
def recvCallback(client: DexHandClient):
    print(client.hand_info.now_pos)
client = DexHandClient(
    ip="192.168.2.100",
    port=60031,
    recvCallback_hand=recvCallback
)
```

4.2.9 Tac3D 信息读取

在 V1.0 的 SDK 版本中，Tac3D 与机械手的使用是完全解耦的，因此获取 Tac3D 信息的方法可参考 Tac3D 的 SDK 二次开发手册，下面的示例仅做参考，请仔细阅读 Tac3D 的 SDK 开发手册后再行使用。

 **注意：**在调用启动服务端函数（start_server）时，会同步启动 Tac3D 传感器，在启动后请尽量不要将 Tac3D 从机械手上拆除；倘若在启动传感器后因特殊原因从机械手上拆除了传感器，则需要首先停止服务端（stop_server），并再次启动服务端（start_server）才能重新启动 Tac3D 传感器。

首先，导入 PyTac3D。

```
import PyTac3D
```

进而，编写接收触觉数据帧的回调函数：

```
def Tac3DRecvCallback(frame, param):
    global SN, frameIndex, sendTimestamp, recvTimestamp, P, D, F
    # 获取 SN
    print(param) # 显示自定义参数

    SN = frame['SN']
    print(SN)

    # 获取帧序号
    frameIndex = frame['index']
    print(frameIndex)

    # 获取时间戳
    sendTimestamp = frame['sendTimestamp']
    recvTimestamp = frame['recvTimestamp']
```

```

# 获取标志点三维形貌
# P 矩阵 (numpy.array) 为 400 行 3 列, 400 行分别对应 400 个标志点, 3
列分别为各标志点的 x, y 和 z 坐标
P = frame.get('3D_Positions')

# 获取标志点三维位移场
# D 矩阵为 (numpy.array) 400 行 3 列, 400 行分别对应 400 个标志点, 3
列分别为各标志点的 x, y 和 z 位移
D = frame.get('3D_Displacements')

# 获取三维分布力
# F 矩阵为 (numpy.array) 400 行 3 列, 400 行分别对应 400 个标志点, 3 列
分别为各标志点附近区域所受的 x, y 和 z 方向力
F = frame.get('3D_Forces')

# 获得三维合力
# Fr 矩阵为 1x3 矩阵, 3 列分别为 x, y 和 z 方向合力
Fr = frame.get('3D_ResultantForce')

# 获得三维合力矩
# Mr 矩阵为 1x3 矩阵, 3 列分别为 x, y 和 z 方向合力矩
Mr = frame.get('3D_ResultantMoment')

```

⚠ 注意：不应将对机器人系统的控制逻辑或其他耗时的程序放在回调函数中执行。否则可能因触觉数据帧处理不及时导致数据堆积和严重的数据滞后。

最后，初始化 Tac3D 实例启动数据接收：

```

# 创建 Sensor 实例, 设置回调函数为上面写好的 Tac3DRecvCallback, 设置 UDP
接收端口为 9988, 数据帧缓存队列最大长度为 5
tac3d = PyTac3D.Sensor(recvCallback=Tac3DRecvCallback, port=9988,
maxQSize=5, callbackParam = 'test param')

# 等待 Tac3D-Desktop 端启动传感器并建立连接
tac3d.waitForFrame()
time.sleep(5) # 5s

```

```
# 发送一次校准信号（应确保校准时传感器未与任何物体接触！否则会输出错误的
数据！）
tac3d.calibrate(SN)
time.sleep(5) #5s

# 获取 frame 的另一种方式：通过 getFrame 获取缓存队列中的 frame
frame = tac3d.getFrame()
if not frame is None:
    print(frame['SN'])

# 例程中没有其他需要执行的任务，故使用 while 循环阻止主程序退出
while True:
    time.sleep(1)
```

4.2.10 错误消除

如果 DexHand 抓取力过大，或 DexHand 电机电流过大，将会触发 DexHand 的安全保护系统，使 DexHand 急停并进入错误状态。此时，hand_info 中的 error_flag 将为 True，DexHand 不能执行任何位置控制、力控制、校准指令。

如果确认 DexHand 已经脱离危险，则可以使用 clear_hand_error 函数清除 DexHand 的错误状态。

```
client.clear_hand_error()
```

表 4.15 clear_hand_error 函数说明

| 函数说明 | 尝试清除 DexHand 错误状态 | | |
|----------|-------------------|-----|-------------|
| 是否等待任务完成 | 是 | 返回值 | bool，表示是否成功 |
| 参数名称 | 数据类型 | 默认值 | 说明 |
| 无 | / | / | / |

4.3 PyDexHand 的例程说明

pyDexHandClient/examples 文件夹内提供了 5 个例程，其作用分别如下：

- activate_service.py：演示如何令服务端开启服务；
- get_info.py：演示如何获得 DexHand 本体和 Tac3D 传感器（限触觉版）信息，以及如何使用回调函数；
- grasp_force_control.py：演示如何使用 DexHand 的力控制指令；
- move_dexhand.py：演示如何使用 DexHand 的位置运动指令；

- handandtac3d.py: 演示机械手与 Tac3D 联合使用的案例;

下面将分段解释各个例程的含义。

4.3.1 开启服务端例程 activate_service

该代码从 dexhand_client 包中导入了 DexHandClient 模块，随后声明了一个名为 client 的 DexHandClient 对象。第二行声明对象时，需要传入服务端的 ip 和端口。随后，利用创建的客户端 (client)，就能够通过调用 start_server 开启服务。

开启服务不要求客户端获得服务端的控制权限，且在客户端被删除后，服务端仍然在提供服务。用户可以在为 DexHand 通电后运行这一脚本以激活服务，无需在自己的代码中调用 start_server。

```
from dexhand_client import DexHandClient

client = DexHandClient(ip="192.168.2.100", port=60031)
client.start_server()
```

4.3.2 获取机械手信息例程 get_info

该代码演示如何获得 DexHand 本体和 Tac3D 传感器（限触觉版）信息，以及如何使用回调函数。

首先，导入程序所需的必要模块。

```
from dexhand_client import DexHandClient
from time import sleep
```

随后，定义 DexHand 本体信息的接收回调函数，该回调函数需要传入一个 DexHandClient 类型的变量，表示用户创建的客户端。使用 hand_info 成员调用 DexHand 本体信息。使用其中的 _frame_cnt 判断当前已经接收了多少帧，当帧数为 20 的倍数时，使用其中的 avg_force 输出 DexHand 的当前平均力。由于 DexHand 本体信息的回传频率为 100Hz，在理想情况下，输出频率应该恰为 5Hz。

```
def recv_callback_hand(client: DexHandClient):
    if client.hand_info._frame_cnt % 20 == 0: # 每 20 帧执行一次
        print(f"DexHand: now position = {client.hand_info.avg_force:.3f}")
```

最后，定义 client，将自定义的回调函数作为参数传入。启动服务端服务，并使程序在 120s 后结束。在程序结束前，终端中将会按照回调函数的定义输出信息。

```
if __name__ == "__main__":
    client = DexHandClient(
```

```

        ip="192.168.2.100",
        port=60031,
        recvCallback_hand=recv_callback_hand,
    )
    client.start_server()
    sleep(120)

```

4.3.3 机械手力控例程 grasp_force_control

该代码演示如何获得 DexHand 本体和 Tac3D 传感器（限触觉版）信息，以及如何使用回调函数。

该代码演示如何使用 DexHand 的力控制指令。运行此例程时，请保证 DexHand 能自由运动，且两指之间没有物体阻挡。

该代码使用如下的 DexHand 本体信息回调函数，以 10Hz 的频率输出 DexHand 是否在错误状态、当前平均力，当前位置。并根据当前任务输出不同的目标值：当当前任务与力控制有关时，输出目标力；当前任务与位置控制有关时，输出目标位置。

```

def report_hand_info(client: DexHandClient):
    info = client.hand_info
    if info._frame_cnt % 10 == 0:
        print(
            f"Error:{info.error_flag},    nowforce:    {info.avg_force:.3f}N
nowpos: {info.now_pos:.3f}mm",
            end=" "
        )
        if info.now_task in ["GOTO", "POSSERVO"]:
            print(f"goalpos : {info.goal_pos:.2f}")
        elif info.now_task in ["SETFORCE", "FORCESERVO"]:
            print(f"goalforce : {info.goal_force:.2f}")
        else:
            print()

```

随后，声明 client，传入上述回调函数，并启动服务（如果已经启动则可以删去）。

```

client = DexHandClient(
    ip="192.168.2.100",
    port=60031,
    recvCallback_hand=report_hand_info,

```

```
)
client.start_server()
```

由于需要控制 DexHand，因此需要获取 DexHand 控制权

```
client.acquire_hand()
```

随后，依次执行 set_home, contact, grasp, forceservo。这一系列命令首先校准了力零点，然后使 DexHand 手指先相互接触，预载 2N 力，再用 5s 时间将接触力增大至 10N，最后将目标力设置为 1N，并等待 8s，保证机械手到达新的目标力。

```
client.set_home()
client.calibrate_force_zero()
client.contact(contact_speed=8,preload_force=2,quick_move_speed=15,quick_move_pos=10)
client.grasp(goal_force=10.0,load_time=5.0)
client.force_servo(goal_force=1.0)
sleep(8)
```

最后，需要释放 DexHand 的控制权

```
client.release_hand()
```

4.3.4 机械手运动例程 move_dexhand

该代码演示如何使用 DexHand 的力控制指令。运行此例程时，请保证 DexHand 能自由运动，且两指之间没有物体阻挡。

该代码和 grasp_force_control.py 几乎一致，只是执行的函数不同。在下面这段代码中，依次执行 set_home,pos_goto,pos_servo。这一系列命令首先让机械手回到零位校准其零点，然后使 DexHand 以最大 40mm/s 的速度，20mm/s² 的加速度运行到 30mm 的位置，随后将目标位置设置为 10mm，令 DexHand 以最大驱动能力运动至该位置。在 DexHand 运动过程中，允许的最大接触力为 3N。

```
client.set_home()
client.pos_goto(goal_pos=30,max_speed=40,max_acc=20,max_f=3)
client.pos_servo(goal_pos=10,max_f=3)
sleep(2)
```

4.3.5 机械手与 Tac3D 联合使用例程 handandtac3d

该代码的主要功能是存储 Tac3D 传感器的测量结果，并在机械手执行任务时提供实时反

馈。运行此例程时，请保证 Tac3D 触觉传感器已经被安装到机械手上。下面是代码的详细解释：

该代码所需导入的库文件如下，包含机械手的库文件 `dexhand_client.py`，标准库 `numpy.py`，这两个库在 4.1 节机械手环境配置时已安装，此外还需导入传感器的库文件 `PyTac3D.py`，该文件可以从 Tac3D 的 SDK 软件包中找到。

```
from dexhand_client import DexHandClient
import numpy as np
import PyTac3D
```

而后创建 Tac3D_info 类：

```
# 用于存储 Tac3D 的测量结果
class Tac3D_info:
    def __init__(self, SN):
        self.SN = SN # 传感器 SN
        self.frameIndex = -1 # 帧序号
        self.sendTimestamp = None # 时间戳
        self.recvTimestamp = None # 时间戳
        self.P = np.zeros((400,3)) # 三维形貌测量结果，400 行分别对应 400 个标志点，3 列分别为各标志点的 x, y 和 z 坐标
        self.D = np.zeros((400,3)) # 三维变形场测量结果，400 行分别对应 400 个标志点，3 列分别为各标志点的 x, y 和 z 变形
        self.F = np.zeros((400,3)) # 三维分布力场测量结果，400 行分别对应 400 个标志点，3 列分别为各标志点的 x, y 和 z 受力
        self.Fr = np.zeros((1,3)) # 整个传感器接触面受到的 x,y,z 三维合力
        self.Mr = np.zeros((1,3)) # 整个传感器接触面受到的 x,y,z 三维合力矩
```

这个类用于存储 Tac3D 传感器的测量结果。包括传感器的序列号（SN）、帧序号（frameIndex）、发送和接收时间戳（sendTimestamp, recvTimestamp）、三维形貌测量结果（P）、三维变形场测量结果（D）、三维分布力场测量结果（F）、整个传感器接触面受到的合力（Fr）和合力矩（Mr），在对某一 Tac3D 传感器创建该类时需要输入对应的传感器 SN 号。

进而，编写接收触觉数据帧的回调函数，当传感器启动后，每次返回数据时均会执行该函数，函数通过传感器的序列号（SN）来确定是哪个传感器触发了回调，并从参数中获取对应的 Tac3D_info 实例。然后，函数会更新这个实例的帧序号、时间戳、三维形貌、位移场、分布力场、合力和合力矩：

```

# Tac3D 的回调函数，当传感器启动后，每次返回数据时均会执行该函数
def Tac3DRecvCallback(frame, param):

    # 获取 SN
    SN = frame['SN'] # 由于机械手上安装了两个 Tac3D 传感器，通过 SN 号确定
    究竟是哪个 Tac3D 调用了该回调函数
    tacinfo = param[SN] # 由于机械手上安装了两个 Tac3D 传感器，通过 SN 号
    确定究竟是哪个 Tac3D 调用了该回调函数

    # 获取帧序号
    frameIndex = frame['index']
    tacinfo.frameIndex = frameIndex

    # 获取时间戳
    sendTimestamp = frame['sendTimestamp']
    recvTimestamp = frame['recvTimestamp']
    tacinfo.sendTimestamp = sendTimestamp
    tacinfo.recvTimestamp = recvTimestamp

    # 获取标志点三维形貌
    # P 矩阵（numpy.array）为 400 行 3 列，400 行分别对应 400 个标志点，3
    列分别为各标志点的 x, y 和 z 坐标
    P = frame.get('3D_Positions')
    tacinfo.P = P

    # 获取标志点三维位移场
    # D 矩阵为（numpy.array）400 行 3 列，400 行分别对应 400 个标志点，3
    列分别为各标志点的 x, y 和 z 位移
    D = frame.get('3D_Displacements')
    tacinfo.D = D

    # 获取三维分布力
    # F 矩阵为（numpy.array）400 行 3 列，400 行分别对应 400 个标志点，3 列
    分别为各标志点附近区域所受的 x, y 和 z 方向力
    F = frame.get('3D_Forces')

```

```

tacinfo.F = F

# 获得三维合力
# Fr 矩阵为 1x3 矩阵, 3 列分别为 x, y 和 z 方向合力
Fr = frame.get('3D_ResultantForce')
tacinfo.Fr = Fr

# 获得三维合力矩
# Mr 矩阵为 1x3 矩阵, 3 列分别为 x, y 和 z 方向合力矩
Mr = frame.get('3D_ResultantMoment')
tacinfo.Mr = Mr

```

进而, 编写接收触觉数据帧的回调函数, 当传感器启动后, 每次返回数据时均会执行该函数。当机械手的数据返回并且帧计数器 (`_frame_cnt`) 是 10 的倍数时, 函数会打印出错误标志、当前力、Tac3D 传感器的合力以及机械手的位置。

此外, 如果机械手的任务是移动到特定位置或位置伺服控制, 它会打印目标位置; 如果是设置力或力伺服控制, 它会打印目标力。

```

# 机械手的回调函数, 当机械手启动后, 每次返回数据时均会执行该函数
def HandRecvCallback(client: DexHandClient):
    info = client.hand_info
    if info._frame_cnt % 10 == 0:
        print(
            f"Error:{info.error_flag},  nowforce1:  {info.now_force[0]:.3f}N
nowforce2:  {info.now_force[1]:.3f}N  nowTacFz1:  {tacinfo1.Fr[0][2]:.3f}N
nowTacFz2: {tacinfo2.Fr[0][2]:.3f}N nowpos: {info.now_pos:.3f}mm",
            end=" ",
        )
    if info.now_task in ["GOTO", "POSSERVO"]:
        print(f"goalpos : {info.goal_pos:.2f}")
    elif info.now_task in ["SETFORCE", "FORCESERVO"]:
        print(f"goalforce : {info.goal_force:.2f}")
    else:
        print()

```

最后, 在主程序内, 创建了一个 `DexHandClient` 实例, 用于与机械手通信。创建了两个 `Tac3D_info` 实例, 分别对应两个不同的 Tac3D 传感器。创建了一个字典 `tac_dict`, 将传感

器序列号映射到对应的 Tac3D_info 实例。创建了一个 PyTac3D 传感器实例，设置了回调函数、UDP 接收端口和数据帧缓存队列的最大长度。

完成实例创建后，开始启动机械手客户端，获取机械手控制权限，进行零点校正，发送校准信号，并执行接触和抓取操作。最后，释放机械手控制权限。

```
if __name__ == "__main__":
    # 创建机械手客户端
    client = DexHandClient(ip="192.168.2.100",port=60031,recvCallback=HandRecvCallback)

    # 创建传感器数据存储实例
    tacinfo1 = Tac3D_info("HDL1-0001") # 注意, 'HDL1-0001'仅是举例, 用户使用时请改成 DexHand 机械手上实际的 Tac3D 传感器编号
    tacinfo2 = Tac3D_info("HDL1-0002") # 注意, 'HDL1-0002'仅是举例, 用户使用时请改成 DexHand 机械手上实际的 Tac3D 传感器编号
    tac_dict = {"HDL1-0001":tacinfo1,"HDL1-0002":tacinfo2}

    # 创建传感器实例, 设置回调函数为上面写好的 Tac3DRecvCallback, 设置UDP 接收端口为 9988, 数据帧缓存队列最大长度为 5
    tac3d = PyTac3D.Sensor(recvCallback=Tac3DRecvCallback, port=9988, maxQSize=5, callbackParam = tac_dict)

    # 启动机械手
    client.start_server() # 启动机械手和 Tac3D, 此后 Tac3D 开始进行测量, 每传回一帧数据时就执行一次回调函数 Tac3DRecvCallback

    client.acquire_hand() # 获取机械手控制权限

    # 机械手零点校正
    client.set_home() # 位置零点
    client.calibrate_force_zero() # 力零点 (一维力传感器)

    # 发送一次校准信号 (应确保校准时传感器未与任何物体接触! 否则会输出错误的数据!)
    tac3d.calibrate("HDL1-0001")
    tac3d.calibrate("HDL1-0002")

    # 接触并施加抓取力
    client.contact(contact_speed=8, preload_force=2, quick_move_speed=15, quick_move_pos=10)

    client.grasp(goal_force=5.0, load_time=5.0)

    # 解除机械手控制权限
```

```
client.release_hand()
```

⚠ 注意：在 IDE（例：VS Code）的调试模式下运行此例程时，有可能会报丢包错误，这是因为调试模式下，IDE 会产生大量的额外开销，导致上位机端程序运行频率降低，再加上同时调用了机械手和传感器的缘故，容易导致系统资源不足，使得通讯延迟过高，导致通讯中断。而无论是在非调试模式下，又或是调试模式下不使用 Tac3D 功能，该问题都不会出现，因此建议用户慎用 IDE 的调试模式。

5 问题与解答 FAQ

5.1 机械手报错无法执行指令?

如果在运行程序时机械手没有响应, 请检查机械手所报的错误, 如果是“Try to clear hand error.”, 则说明机械手检测到了错误, 可能是检测到一维力传感器的受力超过了 20N, 又或者是电机电流超过上限, 这会使 DexHand 进入错误状态。此时, hand_info 中的 error_flag 将为 True, DexHand 不能执行任何位置控制、力控制、校准指令。如果确认 DexHand 已经脱离危险, 则可以使用 clear_hand_error 函数清除 DexHand 的错误状态 (参考 3.4.10 或 4.2.10)。


如果是“TimeoutError: DexHand Server may be not available.” 则可能是出现了通讯异常, 此时可以在创建客户端后, 首先停止服务端(stop_server), 再启动服务端(start_server), 即可解决该问题。

5.2 启动机械手后将指尖传感器拆下?

在调用启动服务端函数 (start_server) 时, 会同步启动 Tac3D 传感器, 在启动后请尽量不要将 Tac3D 从机械手上拆除, 倘若在启动传感器后因特殊原因从机械手上拆除了传感器, 则需要首先停止服务端 (stop_server), 并再次启动服务端 (start_server) 才能重新启动 Tac3D 传感器。

5.3 机械手输出力是否存在上限?

为了防止机械手输出力过大导致出现损伤, 在机械手的底层有两个输出上限的检测, 一是检测一维力传感器的 (单侧) 受力是否会超过 20N, 其次是检测电机输出电流是否超过了阈值。这两个上限值均无法由用户修改, 在运行时如果机械手受到了大于 20N 的力, 或者电机输出电流超过了阈值, 将会触发 DexHand 的安全保护系统, 使 DexHand 急停并进入错误状态。此时, hand_info 中的 error_flag 将为 True, DexHand 不能执行任何位置控制、力控制、校准指令。如果确认 DexHand 已经脱离危险, 则可以使用 clear_hand_error 函数清除 DexHand 的错误状态 (参考 3.4.10 或 4.2.10)。

 当程序结束、但机械手未下电时, 机械手底层仍在实时进行输出力检测, 倘若此时操作人员对机械手施加了超过安全域值的外力 (例如用手强行将机械手指掰开), 机械手会自动进入错误状态, 并关闭电机, 表现为机械手手指会松开; 同时当程序再次启动时, 会提示用户机械手曾出现过错误: “Try to clear hand error.”, 但启动服务端(start_server)时会自动清除机械手的错误状态, 因此程序仍可正常运行, 无需用户手动清除错误。

5.4 机械手在进行抓取力控制时出现了震荡?

在绝大多数情况下, 机械手在执行 grasp 或 force_servo 函数时, 均会正常的进行目标力的跟随, 但在极为特殊的情况下, 可能会出现震荡乃至抓取失效的情况, 具体包括如下情况:

1. 机械手在进行力校准时仍处于受力状态，导致校准完的零点出现偏移。
2. 机械手抓取的物体存在强流变特性，例如没有装满的水袋、沙包等。
3. 给机械手的力控指令是阶跃信号，且阶跃值较大，例如令机械手施加 10N 的阶跃力（`dh_grasp(ctx,10,0)`或 `grasp(10,0)`），则机械手也有可能（并非一定）因目标力施加过于快速导致出现震荡。这里给出的建议是，在抓取刚性物体时，可以施加较大的阶跃力，但在抓取柔性较大的物体时，施加力的速度尽量不要超过 **15N/s**。
4. 机械手内部力传感器或传动机构受损。

当出现上述情况时，可按照上述顺序进行排查，并做出相应的调整，如若实际情况与上述情况均不符，还是出现震荡的情况，请联系公司售后人员。

5.5 机械手在进行位置校准后提示电流过大？

如果发现机械手在使用位置校准(`set_home`)后，在回到零点位置后程序并未正常退出，报了电流值过大的错误，则说明机械手内部的接近开关可能出现了问题，请联系公司售后人员进行维修。

5.6 机械手在进行力校准后读数偏离零点严重？

在使用力校准(`calibrate_force_zero`)后，机械手内部的一维力传感器会将自身读数校零，正常情况下校零时应保证机械手处于无接触、不受外力的状态，校零后在无外力作用时，传感器读数应保持在 0.1N 以下（由于存在噪声和惯性，读数不会严格为 0），但如果发现读数远超该值，则说明一维力传感器可能出现了问题，请联系公司售后人员进行维修。



Acorn
Robotics