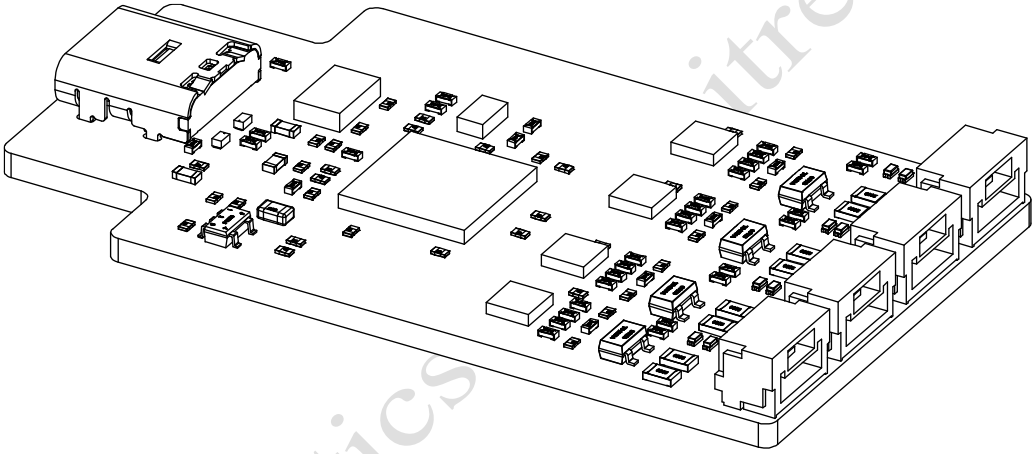
4 路 485 转 USB 模块

使用手册 V1.0



Unitree

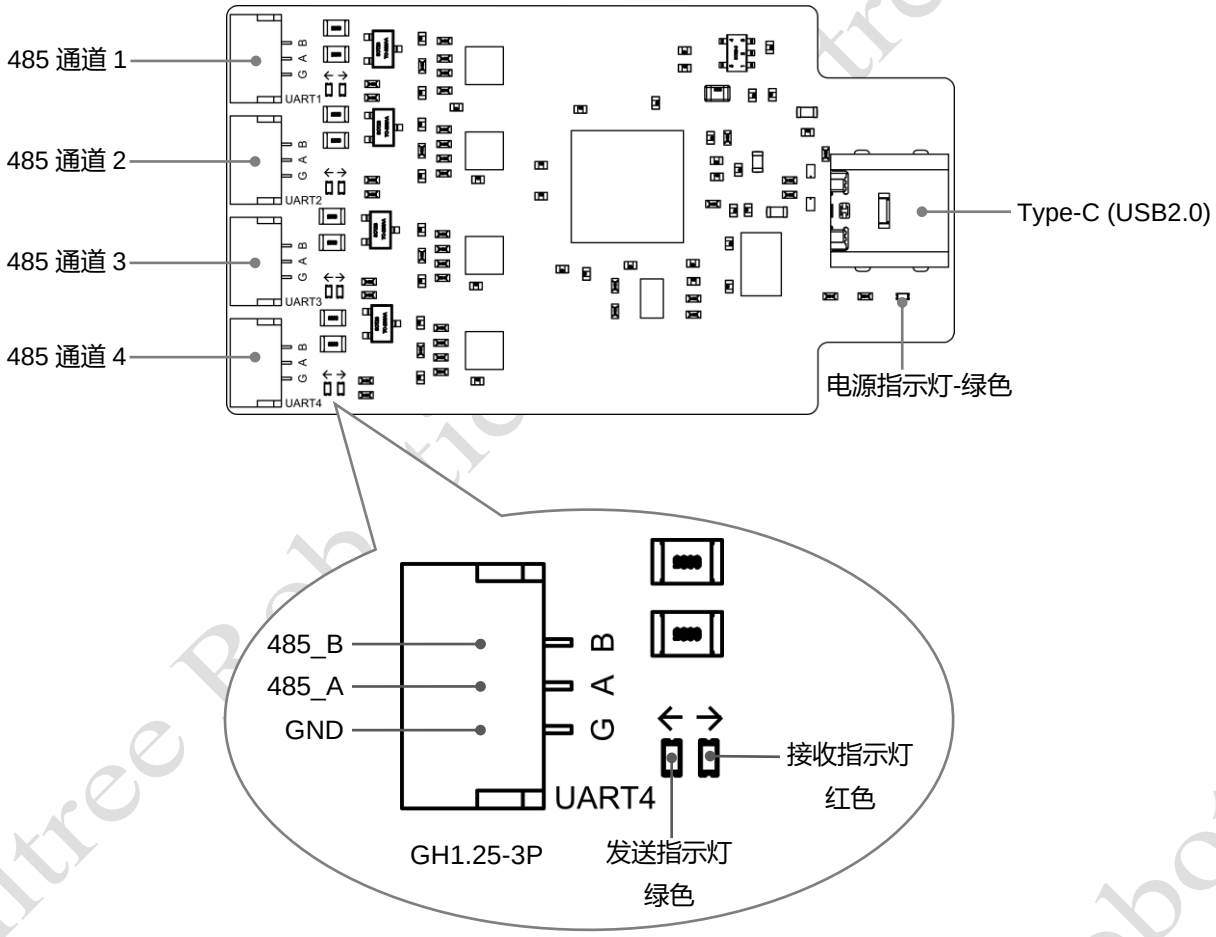
4路485转USB模块

简介

4 路 485 转 USB 模块是一款高性能、多功能的通信转换器，支持高传输速率和多种操作系统，最大传输速率达 115200 bps，确保快速可靠的数据传输，采用 USB 接口，无需额外配置即可被计算机识别并使用，每个 485 通道都配备了电气隔离保护电路，提高系统的稳定性和可靠性，适用于各种需要多路 485 通信的场合。

接口说明

USB 转 4 通道 485 模块 485 接口定义以及指示灯如下图所示。

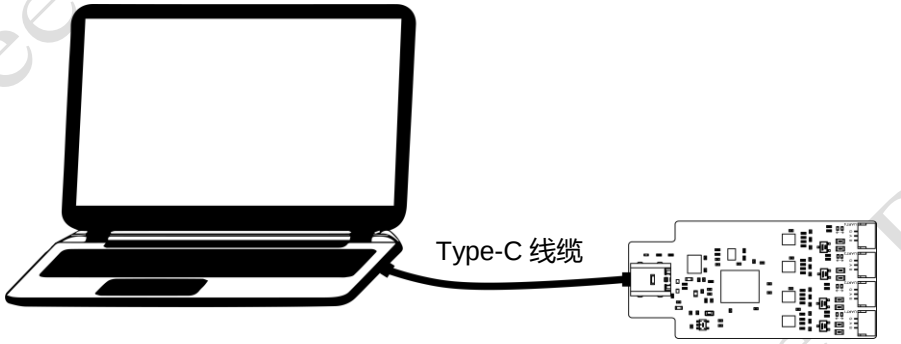


● 485 接口为 5V 标准，为保证通信质量请确保 USB 接口供电稳定！

调试连接

(1) 硬件连接

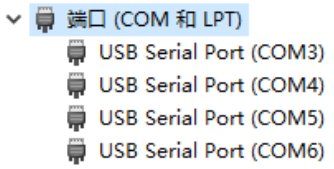
- **连接 USB 端口：**将 USB 数据线的一端插入电脑的 USB 接口，另一端插入设备的 USB 接口。
- **连接 485 设备：**使用 GH1.25-3P 通讯线将设备的 4 个 485 端口分别连接到目标 485 设备。



(2) 软件安装

打开电脑设备管理器，若未出现四个串口（端口号可能有所不同），且设备管理器中有未识别设备，

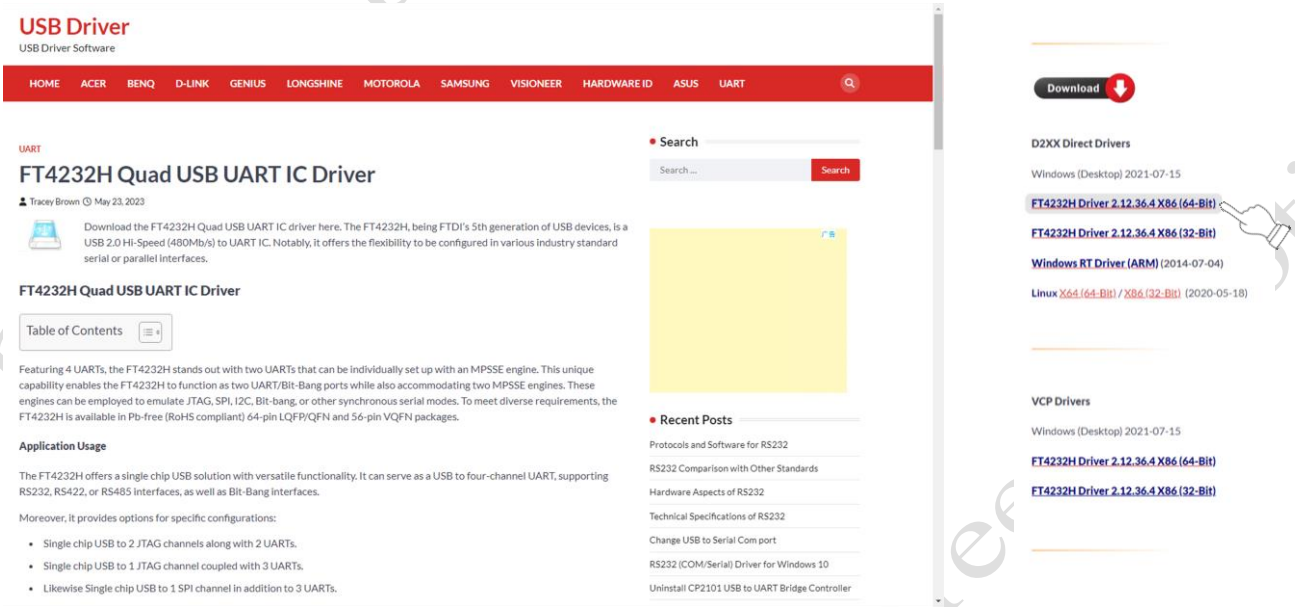
未安装驱动



驱动已安装

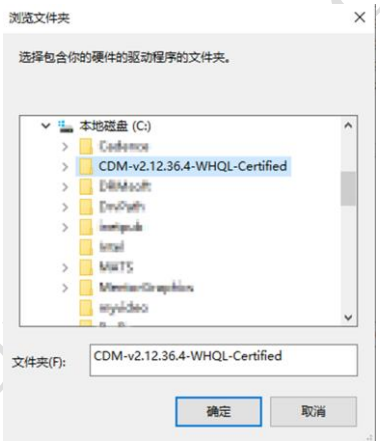
则说明当前电脑并没有安装此设备驱动。请访问：https://www.usb-drivers.org/ft4232h-quad-usb-uart-ic-driver.html 下载最新驱动程序。

滑动到页面底端选择 FT4232H Driver 2.12.36.4 X86 (64-Bit) 文件下载，下载后记住解压位置。



驱动程序下载成功后，在设备管理器中选择未知的设备右键单击-更新驱动程序

浏览我的电脑以查找驱动程序，选择解压的文件夹路径后点击确认。



点击下一步，成功安装驱动程序。

- 四个未知设备需要安装四次驱动，安装完成后重新插拔设备即可，
- 在“设备管理器”-“端口 (COM 和 LPT)”确认设备已被正确识别。

阻塞式电机通信例程

例程概述

本例程展示了如何利用 USB 转 4 路 485 模块与 12 个电机进行阻塞式通信。该系统支持三种运行模式：

1. **速度模式**：电机输出端以约 1 转/秒的速度运行。
2. **力矩模式**：电机输出端产生 0.25Nm 的力矩。如果电机初始静止，可尝试逆时针旋转输出轴，一旦克服启动力矩，电机将开始转动。
3. **停止模式**：电机停止运行（零力矩状态）。

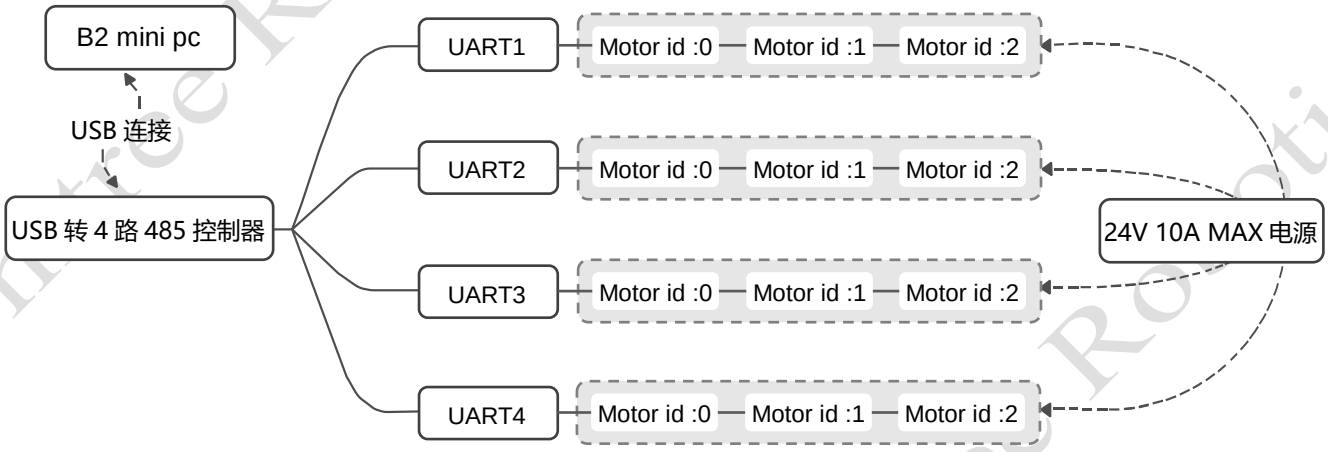
本例程旨在演示多通道电机控制系统的基本功能，包括参数设置、数据采集和实时监控。

硬件要求

为了成功运行本例程，您需要准备以下硬件设备：

设备	数量	说明
USB 转 4 路 485 模块	1	用于连接计算机和电机
GO-M8010-6 电机	12	被控制的电机
XT30（2+2）转 GH1.25-3P	4	用于电源和信号连接
XT30(2+2) 双头母头	12	用于电机连接
Ubuntu 系统计算机	1	用于运行控制程序
电源	1	24V，最大电流≥10A

硬件连接



● 注意：确保所有连接牢固可靠，并在通电前仔细检查接线是否正确，以避免可能的设备损坏。

安全注意事项

1. 在操作过程中，请遵守所有相关的安全规程和指南。
2. 确保工作区域通风良好，远离易燃易爆物品。
3. 操作高速旋转的电机时，请保持安全距离，避免接触转动部件。
4. 如遇异常情况，请立即切断电源并排查故障。

软件环境配置

本例程在 Ubuntu 操作系统下运行。推荐您的系统满足以下要求：

- Ubuntu 20.04 LTS 或更高版本
- GCC 9.3.0 或更高版本
- CMake 3.11 或更高版本

> - **需要打上 rt 实时性补丁**，具体操作请查看“附录”章节。

例程运行

(1) 准备工作

1. 确保所有硬件连接正确，并给系统供电。
2. 将 USB 转 4 路 485 模块插入计算机的 USB 端口。
3. 打开终端，运行以下命令查看设备识别情况：

```
ls /dev/ttyUSB*
```

您应该能看到四个 ttyUSB 设备，对应模块的四个通道：

- UART1 对应 /dev/ttyUSB0
- UART2 对应 /dev/ttyUSB1
- UART3 对应 /dev/ttyUSB2
- UART4 对应 /dev/ttyUSB3

4. 如果没有看到设备，请检查连接或尝试重新插拔 USB 模块。

(2) 编译程序

如果您还没有编译程序，请按以下步骤操作：

1. 进入项目根目录：

   cd path/to/project

2. 创建并进入 build 目录：

```
mkdir build && cd build
```

3. 运行 CMake 并编译：

```
cmake ..
make
```

(3) 运行例程

1. 在 build 目录中，根据需要选择运行模式，执行相应的命令：

- **停止模式：**

```
sudo ./block_comm_test stop
```

- **力矩模式**：

```
sudo ./block_comm_test tor
```

- **速度模式：**

```
sudo ./block_comm_test speed
```

- 注意：使用 sudo 运行程序以确保有足够的权限访问 USB 设备。

2. 程序开始运行后，您将看到实时更新的电机状态信息

(4) 运行结果解释

程序运行后，每秒都会输出类似下图的日志信息：

```
Channel 0, Motor 0 - Output Torque: 0, Output Speed: 0, Output Position: 248.783, Temp: 35, Error: 0
Channel 0, Motor 1 - Output Torque: 0, Output Speed: -0.0155094, Output Position: 97.3103, Temp: 32, Error: 0
Channel 0, Motor 2 - Output Torque: -0.0494531, Output Speed: -0.00387736, Output Position: 135.035, Temp: 33, Error: 0
Channel 1, Motor 0 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 1, Motor 1 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 1, Motor 2 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 2, Motor 0 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 2, Motor 1 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 2, Motor 2 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 3, Motor 0 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 3, Motor 1 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel 3, Motor 2 - Output Torque: 0, Output Speed: 0, Output Position: 0, Temp: 0, Error: 0
Channel | Motor | Sent | Received | Lost | Loss Rate
--------|-------|------|----------|------|----------
      0 |     0 |  334 |      334 |    0 |     0.00%
      0 |     1 |  334 |      333 |    1 |     0.30%
      0 |     2 |  333 |      334 |    0 |     0.00%
      1 |     0 |    3 |        0 |    3 |   100.00%
      1 |     1 |    3 |        0 |    3 |   100.00%
      1 |     2 |    3 |        0 |    3 |   100.00%
      2 |     0 |    3 |        0 |    3 |   100.00%
      2 |     1 |    3 |        0 |    3 |   100.00%
      2 |     2 |    3 |        0 |    3 |   100.00%
      3 |     0 |    3 |        0 |    3 |   100.00%
      3 |     1 |    3 |        0 |    3 |   100.00%
      3 |     2 |    3 |        0 |    3 |   100.00%
```


日志包含以下信息：

- 每个通道和电机的当前状态。
- 输出转矩、速度、位置等参数。
- 电机温度和错误代码（如果有）。
- 每秒的数据包发送和接收统计。

说明：

- Output Torque：输出端转矩，单位为 Nm。
- Output Speed：输出端速度，单位为 rad/s。
- Output Position：输出端位置，单位为 rad。
- Temp：电机温度，单位为 ℃。
- Error：错误代码，0 表示无错误。

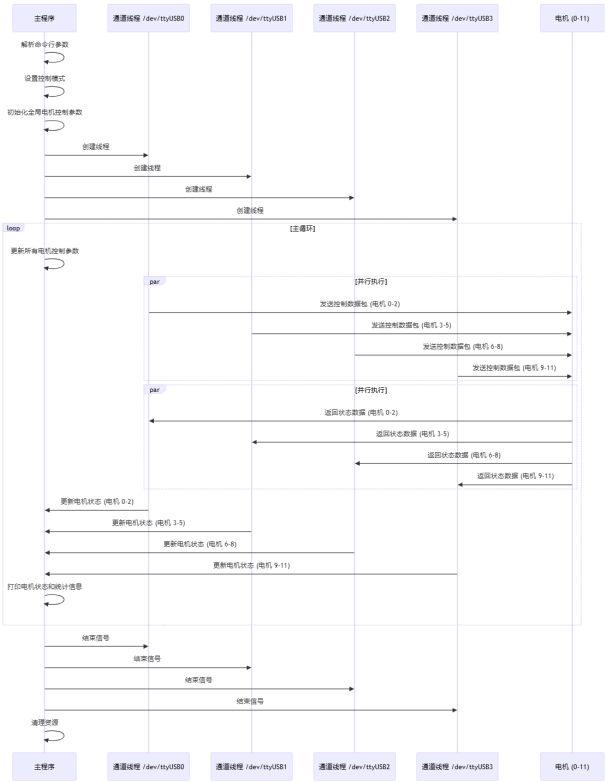
- 注意：图中仅显示了 UART1 通道的三个电机数据，如果您连接了更多电机，将会看到相应的额外信息。

(5) 停止程序

要停止程序运行，请在终端中按 Ctrl+C。程序将会安全地停止所有电机并清理资源。

代码结构

以下是本例程的代码结构框架图：



代码主要分为以下几个部分：

1. 主程序：负责初始化、运行模式选择和 overall 控制。
2. 通道线程：每个通道一个线程，负责与该通道上的电机通信。
3. 电机控制：包括参数设置、状态获取等功能。
4. 数据处理：处理接收到的电机数据，更新状态。
5. 统计信息：记录和显示通信统计数据。

主函数详细解释

主函数（main）是程序的入口点，负责整个系统的初始化、运行和清理。以下是其主要部分的详细解释：

(1) 命令行参数处理

```
if (argc != 2) {
    std::cerr << "Usage: " << argv[0] << " <mode>\n";
    std::cerr << "Modes: stop, tor, speed\n";
    return 1;
}
```

这部分检查命令行参数，确保用户提供了正确的运行模式。

(2) 控制模式设置

```
if (strcmp(argv[1], "stop") == 0) {
    // 停止模式设置
} else if (strcmp(argv[1], "tor") == 0) {
    // 转矩模式设置
} else if (strcmp(argv[1], "speed") == 0) {
    // 速度模式设置
} else {
    // 错误处理
}
```

这部分根据命令行参数设置相应的控制模式。根据用户输入的模式，设置相应的全局控制参数。这些参数将影响所有电机的行为。

(3) 电机初始化

```
for (int i = 0; i < NUM_CHANNELS; ++i) {
    for (int j = 0; j < MOTORS_PER_CHANNEL; ++j) {
        g_motors[i][j].setControlParams(0.0f, 0.0f, 0.0f, 0.0f, 0.0f);
    }
}
```

这部分初始化所有电机对象，并设置默认的控制参数。

(4) 通道线程创建

```
std::vector<std::thread> threads;
for (int i = 0; i < NUM_CHANNELS; ++i) {
    threads.emplace_back(channel_thread, i);
}
```

为每个通道创建一个独立的线程，每个线程负责管理该通道上的三个电机。

(5) 主循环

```
while (g_running) {
    std::this_thread::sleep_for(std::chrono::seconds(1));
    // 更新电机控制参数和获取状态
    for (int i = 0; i < NUM_CHANNELS; ++i) {
        for (int j = 0; j < MOTORS_PER_CHANNEL; ++j) {
            // 更新控制参数
            // 获取和打印电机状态
        }
    }
    // 打印统计信息
    print_statistics();
}
```

主循环每秒执行一次，负责：

更新所有电机的控制参数，考虑减速比进行转换。

获取并打印每个电机的当前状态。

打印整体统计信息。

(6) 资源清理

```
g_running = false;
for (auto& thread : threads) {
    thread.join();
}
```

程序结束时，通过设置 g_running 为 false 来停止所有线程，然后等待它们完成。这个主函数设计允许系统同时控制多个电机，提供了灵活的运行模式选择，并通过多线程实现了高效的并行操作。它还包括了完整的初始化和清理流程，确保系统资源的正确管理。

故障排除

如果在运行例程时遇到问题，请检查以下几点：

硬件连接	(1) 确保所有电缆连接正确且牢固。 (2) 检查电源连接，确保电压稳定在 24V。
软件环境	(1) 验证 Ubuntu 系统版本是否符合要求 (2) 检查是否安装了所有必要的依赖库
USB 设备识别	(1) 使用 lsusb 命令检查 USB 设备是否被系统识别 (2) 如果设备未被识别，尝试在不同的 USB 端口上插拔设备
权限问题	确保使用 sudo 运行程序
编译问题	(1) 如果编译失败，检查 CMake 和 GCC 版本是否满足要求 (2) 查看编译错误信息，解决可能的依赖问题
运行时错误	(1) 确保没有其他程序占用了 USB 转 485 设备 (2) 如果按照以上步骤检查后问题仍然存在，请收集以下信息并联系技术支持： ● 完整的错误日志 ● 系统信息（使用 uname -a 命令获取） ● 设备连接的详细描述 ● 尝试解决问题的步骤

结论

本例程演示了如何使用 USB 转 4 路 485 模块控制多个电机，实现了基本的速度控制、力矩控制和停止功能。通过这个例程，用户可以了解多通道电机控制系统的基本原理和实现方法。

在实际应用中，可以基于此例程进行扩展，增加更复杂的控制算法、更详细的错误处理、以及与其他系统的集成。这个基础框架为开发更高级的机器人控制系统或工业自动化应用提供了良好的起点。

通过不断改进和扩展这个基础例程，开发者可以构建出功能更加强大和灵活的电机控制系统。

附录

- GO-M8010-6电机数据手册
- Ubuntu22.04 编译实时内核安装 preempt_rt 实时补丁

修订历史

版本	日期	修改内容
1.0	2024-9-26	初始版本

Unitree 技术支持

Unitree Support

https://www.unitree.com

内容如有更新，恕不另外通知

您可以在 Unitree 官方网站获取更多文档

https://www.unitree.com/download

If you have any questions or suggestions about the manual, please contact us at the following

E-mail address: support@unitree.cc

微信扫一扫，关注 Unitree 公众号