L30机械手 API 使用指南

目录

安装说明

系统要求

- Python 3.6 或更高版本
- Windows/Linux/MacOS 系统

安装步骤

1. 获取安装包：
    - 从提供的 USB 设备或网盘下载以下任一安装包：
        - `l30_hand_api-1.0.0-py3-none-any.whl`（推荐，安装更快）
        - `l30_hand_api-1.0.0.tar.gz`
    - 或者从技术支持处获取最新版本的安装包
2. 安装依赖：

```
# 安装pyserial库
pip install pyserial
```

3. 安装L30机械手API：

```
# 进入安装包所在目录
cd /所在目录

# 使用wheel包安装（推荐）
pip install l30_hand_api-1.0.0-py3-none-any.whl

# 或者使用源码包安装
pip install l30_hand_api-1.0.0.tar.gz
```

4. 验证安装：

```
# 运行Python，输入以下代码
from l30_hand_api import L30HandAPI
hand = L30HandAPI()
print("安装成功！")
```

可能遇到的安装问题

1. 权限问题
   - Linux系统可能需要使用sudo：

     ```
     sudo pip install l30_hand_api-1.0.0-py3-none-any.whl
     # 或
     sudo pip install l30_hand_api-1.0.0.tar.gz
     ```

   - 或者使用 `--user` 选项安装到用户目录：

     ```
     pip install --user l30_hand_api-1.0.0-py3-none-any.whl
     # 或
     pip install --user l30_hand_api-1.0.0.tar.gz
     ```

2. 串口访问权限（Linux系统）
   - 添加当前用户到dialout组：

     ```
     sudo usermod -a -G dialout $USER
     ```

   - 重新登录后生效
3. 卸载和重新安装
   - 如需重新安装，先卸载旧版本：

     ```
     pip uninstall l30-hand-api
     ```

   - 然后重新安装新版本

基本使用流程

0.注意事项：

严格按照demo.py程序流程使用，先使用limit_angle限制手关节运动范围，再使用joint_to_motor_angle将关节角度转换为电机角度，最后使用set_motor_position。若使用其他方法导致机械手损坏，概不负责。 API使用方法以demo.py程序为准 例如：

```
joint_angle = float(value)
# 先限制关节角度范围
limited_joint_angle = self.hand_api.limit_angle(motor_id, joint_angle)
# 将关节角度转换为电机角度
motor_angle = self.hand_api.joint_to_motor_angle(motor_id,
limited_joint_angle)

# 更新输入框显示（显示关节角度）
self.motors[motor_id]['entry'].delete(0, tk.END)
self.motors[motor_id]['entry'].insert(0, f"{limited_joint_angle:.1f}")

# 设置电机位置
if not self.hand_api.set_motor_position(motor_id, motor_angle):
    messagebox.showerror("错误", f"设置电机 {motor_id} 位置失败")


### 1. 导入并初始化
```python
from l30_hand_api import L30HandAPI

# 创建API实例
hand = L30HandAPI()
```

2. 连接到机械手

```
# 获取可用串口列表
available_ports = hand.get_available_ports()
print("可用串口:", available_ports)

# 连接到指定串口
success, message = hand.connect_motor("/dev/ttyUSB0")  # Windows下使用类
似"COM3"
if success:
    print("连接成功！")
else:
    print(f"连接失败: {message}")
```

3. 基本操作
```

```
# 启用电机出力
hand.set_all_torque(True)

# 设置运动速度（0-1000）
hand.set_all_velocity_sync(200)

# 打开手掌（所有手指伸直）
hand.open_hand()

# 读取当前位置
positions = hand.read_current_positions()
print("当前位置:", positions)

# 断开连接
hand.disconnect_motor()
```

API详细说明

类：L30HandAPI

初始化

```
hand = L30HandAPI()
```

创建API实例，不需要参数。

连接管理方法

1. `get_available_ports()`
   - 功能：获取系统中可用的串口列表
   - 返回：串口名称列表
   - 示例：

     ```
     ports = hand.get_available_ports()
     print("可用串口:", ports)
     ```

2. `connect_motor(port: str)`
   - 功能：连接到指定串口
   - 参数：
     - port: 串口名称（如"/dev/ttyUSB0"或"COM3"）
   - 返回：(成功状态, 消息)
   - 示例：

     ```
     success, msg = hand.connect_motor("/dev/ttyUSB0")
     ```

3. `disconnect_motor()`

- 功能：断开当前连接
- 返回：是否成功断开
- 示例：

```
hand.disconnect_motor()
```

手关节控制方法

1. `set_all_torque(enable: bool)`
   - 功能：设置所有手指关节的出力状态
   - 参数：
     - enable: True表示启用，False表示禁用
   - 返回：是否成功
   - 示例：

```
hand.set_all_torque(True)  # 启用所有关节出力
```

2. `toggle_all_torque()`
   - 功能：切换所有手指关节的出力状态
   - 返回：(是否成功, 新状态)
   - 示例：

```
success, new_state = hand.toggle_all_torque()
```

3. `set_all_velocity_sync(velocity: int)`
   - 功能：同步设置所有手指关节的运动速度
   - 参数：
     - velocity: 速度值（0-1000，建议200-500）
   - 说明：速度设置过大可能导致手指运动不稳定
   - 示例：

```
hand.set_all_velocity_sync(200)  # 设置适中的运动速度
```

4. 手指关节控制

```python
# 从API导入关节映射函数
from l30_hand_api.motor_mapping import get_table_columns,
get_motor_id, get_joint_info

# 获取所有关节名称和对应的电机ID
joint_names = get_table_columns()[1:]  # 跳过序号列

# 关节名称和ID的对应关系:
关节列表 = [
    "食指指根",    # motor_id: 1, 范围: 0-90度
    "食指指节",    # motor_id: 2, 范围: 0-90度
    "拇指指根",    # motor_id: 3, 范围: 0-90度
    "拇指指节",    # motor_id: 4, 范围: 0-90度
    "中指指根",    # motor_id: 5, 范围: 0-90度
    "中指指节",    # motor_id: 6, 范围: 0-90度
    "无名指指根",  # motor_id: 7, 范围: 0-90度
    "无名指指节",  # motor_id: 8, 范围: 0-90度
    "小指指根",    # motor_id: 9, 范围: 0-90度
    "小指指节",    # motor_id: 10, 范围: 0-90度
]

# 获取关节信息
def get_joint_details(joint_name):
    """获取关节的详细信息"""
    motor_id = get_motor_id(joint_name)
    info = get_joint_info(joint_name)
    return motor_id, info
```

5. `set_motor_position(joint_id: int, angle: float)`
   - 功能：设置单个手指关节的角度
   - 参数：
     - joint_id: 关节ID（使用 get_motor_id 获取）
     - angle: 目标角度（会自动限制在关节的有效范围内）
   - 返回：是否成功
   - 示例：

```python
# 使用关节名称获取电机ID
motor_id = get_motor_id("食指指根")
# 设置关节角度
hand.set_motor_position(motor_id, 45)

# 获取关节信息并设置
motor_id, info = get_joint_details("拇指指根")
hand.set_motor_position(motor_id, 90)
```

6. `set_motors_position_sync(joint_positions: dict)`
   - 功能：同步设置多个手指关节的角度
   - 参数：
     - joint_positions: 关节ID和角度的字典
   - 说明：

- 同步设置可以实现更流畅的动作
- 角度值会自动限制在有效范围内
- 建议使用 get_motor_id 获取关节ID

- 示例：

```
# 使用关节名称构建位置字典
positions = {
    get_motor_id("食指指根"): 45,
    get_motor_id("拇指指根"): 90,
    get_motor_id("中指指根"): 30
}
hand.set_motors_position_sync(positions)

# 或者使用预定义的电机ID
positions = {
    1: 45,  # 食指指根
    3: 90,  # 拇指指根
    5: 30   # 中指指根
}
hand.set_motors_position_sync(positions)
```

7. 角度限制和安全检查

```
def limit_angle(self, motor_id: int, angle: float) -> float:
    """
    将角度限制在关节的有效范围内
    所有关节的角度范围都是0-90度
    """
    if motor_id in self.motor_limits:
        min_angle = self.motor_limits[motor_id].get('min', 0)
        max_angle = self.motor_limits[motor_id].get('max', 90)
        return max(min_angle, min(angle, max_angle))
    return angle
```

8. open_hand()

- 功能：执行手掌完全张开动作
- 说明：所有手指伸直（关节角度设为0度）
- 返回：是否成功
- 示例：

```
hand.open_hand()  # 张开手掌
```

状态读取方法

1. read_current_positions()
   - 功能：读取所有电机的当前位置
   - 返回：电机ID和位置的字典
   - 示例：

```
positions = hand.read_current_positions()
print("当前位置:", positions)
```

预设动作方法

1. `open_hand()`
   - 功能：执行手掌完全张开动作
   - 返回：是否成功
   - 示例：

```
hand.open_hand()
```

校准相关方法

1. `calibrate_hand(password: str)`
   - 功能：执行手部校准
   - 参数：
     - password: 校准密码
   - 返回：(是否成功, 消息)
   - 示例：

```
success, msg = hand.calibrate_hand("your_password")
```

常见问题解答

Q1: 连接失败怎么办?

A1: 请检查：

- 串口名称是否正确
- 串口是否被其他程序占用
- 机械手是否已通电

Q2: 电机不运动怎么办?

A2: 请检查：

- 是否已启用电机出力（set_all_torque(True)）
- 运动速度是否设置过小
- 目标位置是否在有效范围内
- 是否已完成校准

Q3: 如何正确断开连接?

A3: 建议按以下步骤操作：

1. 停止所有运动
2. 关闭电机出力
3. 调用disconnect_motor()方法

示例代码

基本控制示例

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

## GUI应用示例

本节将详细介绍如何使用 L30HandAPI 构建一个完整的图形界面应用程序，完全基于
`demo.py` 的实现。

### 1. 导入必要的模块
```python
import os
import sys
import time
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import simpledialog
import threading
import serial.tools
import serial.tools.list_ports
from ttkbootstrap import Style  # 使用ttkbootstrap美化界面
import csv
from tkinter import filedialog

# 从安装的包中导入API和必要的函数
from l30_hand_api import L30HandAPI
from l30_hand_api.motor_mapping import get_table_columns, get_motor_id,
get_joint_info

# Dynamixel电机控制表地址定义
ADDR_OPERATING_MODE = 11      # 操作模式地址
ADDR_TORQUE_ENABLE = 64       # 扭矩使能地址
ADDR_GOAL_POSITION = 116      # 目标位置地址
ADDR_PRESENT_POSITION = 132   # 当前位置地址
ADDR_MOVING = 122            # 运动状态地址
ADDR_PROFILE_VELOCITY = 112   # 速度控制地址

# 协议版本和通信参数设置
PROTOCOL_VERSION = 2.0        # Dynamixel协议2.0版本
BAUDRATE = 1000000           # 串口波特率1M
DEFAULT_VELOCITY = 255       # 默认速度（最大值）
MAX_VELOCITY = 1000          # 最大速度值
DEFAULT_INTERVAL = 0.2      # 默认动作间隔时间（秒）

# 电机控制参数
```
```

```
# 电机控制参数
TORQUE_ENABLE = 1          # 扭矩使能
TORQUE_DISABLE = 0         # 扭矩禁用
DXL_MINIMUM_POSITION_VALUE = 0      # 最小位置值
DXL_MAXIMUM_POSITION_VALUE = 4095  # 最大位置值（12位分辨率）
POSITION_CONTROL_MODE = 3          # 位置控制模式

class MultiMotorControlGUI:
    """
    实现了机械手控制界面的所有功能，包括：
    - 界面布局
    - 用户交互
    - 动作序列管理
    - 数据保存和加载
    """

    def __init__(self, root):
        """
        初始化控制界面
        Args:
            root: tkinter主窗口对象
        """
        self.root = root
        self.root.title("L30_demo")

        # 初始化API实例
        self.hand_api = L30HandAPI()
        self.hand_api.gui = self  # 设置GUI引用

        # 初始化其他属性
        self.is_connected = False
        self.sequence_running = False      # 序列运行状态标志
        self.stop_flag = False
        self.loop_count_label = None      # 循环计数标签

        # 电机控制组件存储
        self.motors = {}  # 存储电机控制组件
        # 初始化所有电机的控制组件
        for motor_id in range(1, 18):
            self.motors[motor_id] = {
                'label': None,             # 电机标签
                'slider': None,            # 滑块控件
                'entry': None,             # 输入框控件
                'current_pos': None        # 当前位置标签
            }

        # 界面控件
        self.velocity_label = None  # 速度显示标签
        self.velocity_slider = None  # 速度滑块
        self.port_combobox = None   # 串口选择下拉框

        # 设置界面布局
        self.setup_gui()

        # 初始化串口列表
        self.refresh_ports()
```

```
        # 检查校准文件
        if not os.path.exists('calibration.csv'):
            messagebox.showerror("错误", "未找到校准文件！请先进行手部校准。")
            self.disable_operation_controls()
```

主要功能实现示例

1. 连接管理

```
def refresh_ports(self):
    """刷新可用串口列表"""
    ports = [port.device for port in serial.tools.list_ports.comports()]
    self.port_combobox['values'] = ports
    if ports:
        self.port_combobox.set(ports[0])

def toggle_connection(self):
    """切换连接状态"""
    if not self.is_connected:
        self.connect_motor()
    else:
        self.disconnect_motor()

def connect_motor(self):
    """连接到机械手"""
    if not self.port_combobox.get():
        messagebox.showerror("错误", "请选择串口")
        return

    success, message =
self.hand_api.connect_motor(self.port_combobox.get())
    if success:
        self.is_connected = True
        self.connect_btn.configure(text="断开")
        self.port_combobox.state(['disabled'])
        self.enable_operation_controls()
        messagebox.showinfo("成功", "连接成功！")
    else:
        messagebox.showerror("错误", f"连接失败: {message}")

def disconnect_motor(self):
    """断开连接"""
    if self.hand_api.disconnect_motor():
        self.is_connected = False
        self.connect_btn.configure(text="连接")
        self.port_combobox.state(['!disabled'])
        self.disable_operation_controls()
```

2. 电机控制

```
    def toggle_all_torque(self):
        """切换所有电机的使能状态"""
        if not self.is_connected:
            return

        success, new_state = self.hand_api.toggle_all_torque()
        if success:
            if new_state:
                self.torque_btn.configure(text="关闭电机出力")
                self.enable_operation_controls()
            else:
                self.torque_btn.configure(text="开启电机出力")
                self.disable_operation_controls()

    def on_slider_change(self, motor_id, value):
        """处理滑块值变化事件"""
        if not self.is_connected:
            return

        try:
            joint_angle = float(value)
            # 先限制关节角度范围
            limited_joint_angle = self.hand_api.limit_angle(motor_id,
joint_angle)
            # 将关节角度转换为电机角度
            motor_angle = self.hand_api.joint_to_motor_angle(motor_id,
limited_joint_angle)

            # 更新输入框显示（显示关节角度）
            self.motors[motor_id]['entry'].delete(0, tk.END)
            self.motors[motor_id]['entry'].insert(0, f"
{limited_joint_angle:.1f}")

            # 设置电机位置
            if not self.hand_api.set_motor_position(motor_id, motor_angle):
                messagebox.showerror("错误", f"设置电机 {motor_id} 位置失败")

        except ValueError:
            pass

    def on_entry_change(self, motor_id):
        """处理输入框值变化事件"""
        if not self.is_connected:
            return

        try:
            value = self.motors[motor_id]['entry'].get().strip()
            joint_angle = float(value) if value else 0.0

            # 先限制关节角度范围
            limited_joint_angle = self.hand_api.limit_angle(motor_id,
joint_angle)
            # 将关节角度转换为电机角度
            motor_angle = self.hand_api.joint_to_motor_angle(motor_id,
limited_joint_angle)
```

```
            # 更新滑块
            self.motors[motor_id]['slider'].set(limited_joint_angle)

            # 设置电机位置
            if not self.hand_api.set_motor_position(motor_id, motor_angle):
                messagebox.showerror("错误", f"设置电机 {motor_id} 位置失败")

        except ValueError:
            messagebox.showerror("错误", "请输入有效的数字")
            # 重置为当前位置
            self.read_current_positions()
```

3. 动作序列管理

```
def read_current_positions(self):
    """读取所有启用电机的当前位置"""
    if not self.is_connected:
        messagebox.showerror("错误", "请先连接串口")
        return

    # 读取当前位置
    positions = self.hand_api.read_current_positions()
    if not positions:
        messagebox.showerror("错误", "读取电机位置失败")
        return

    # 更新界面显示
    for motor_id, motor_angle in positions.items():
        if motor_id in self.motors:
            # 转换为关节角度
            joint_angle = self.hand_api.motor_to_joint_angle(motor_id,
motor_angle)
            # 更新滑块和输入框的值，保留一位小数
            self.motors[motor_id]['slider'].set(joint_angle)
            self.motors[motor_id]['entry'].delete(0, tk.END)
            self.motors[motor_id]['entry'].insert(0, f"{joint_angle:.1f}")
            # 更新当前位置显示
            self.motors[motor_id]['current_pos'].configure(text=f"当前位置:
{joint_angle:.1f}°")
            print(f"电机 {motor_id} - 电机角度: {motor_angle:.1f}°, 关节角度:
{joint_angle:.1f}°")
```

4. 数据保存和加载

```
def save_current_positions(self):
    """保存当前所有关节角度"""
    if not self.is_connected:
        messagebox.showerror("错误", "请先连接串口")
        return

    # 获取当前位置
    positions = self.hand_api.save_current_positions()
    if not positions:
        messagebox.showerror("错误", "读取电机位置失败")
        return

    # 生成新的序号
    items = self.tree.get_children()
    new_index = len(items) + 1

    # 准备表格数据
    values = [str(new_index)]  # 序号列
    for col in self.table_columns[1:]:  # 跳过序号列
        motor_id = get_motor_id(col)
        if motor_id in positions:
            values.append(f"{positions[motor_id]:.1f}")
        else:
            values.append("0.0")

    # 插入新行
    self.tree.insert('', 'end', values=values)

    # 更新界面显示
    for motor_id, joint_angle in positions.items():
        if motor_id in self.motors:
            # 更新滑块和输入框的值，保留一位小数
            self.motors[motor_id]['slider'].set(joint_angle)
            self.motors[motor_id]['entry'].delete(0, tk.END)
            self.motors[motor_id]['entry'].insert(0, f"{joint_angle:.1f}")
            # 更新当前位置显示
            self.motors[motor_id]['current_pos'].configure(text=f"当前位置:
{joint_angle:.1f}°")
            print(f"电机 {motor_id} - 关节角度: {joint_angle:.1f}°")
```

5. 更新电机UI

```
def update_motor_ui(self, motor_id, joint_angle):
    """更新电机UI显示"""
    if motor_id in self.motors:
        # 更新滑块和输入框的值，保留一位小数
        self.motors[motor_id]['slider'].set(joint_angle)
        self.motors[motor_id]['entry'].delete(0, tk.END)
        self.motors[motor_id]['entry'].insert(0, f"{joint_angle:.1f}")
        # 更新当前位置显示
        self.motors[motor_id]['current_pos'].configure(text=f"当前位置:
{joint_angle:.1f}°")
```

6. 主程序入口

```
def main():
    root = tk.Tk()
    style = Style(theme='cosmo')  # 使用ttkbootstrap主题
    app = MultiMotorControlGUI(root)

    # 设置窗口关闭处理
    def on_closing():
        if app.is_connected:
            app.disconnect_motor()
        root.destroy()

    root.protocol("WM_DELETE_WINDOW", on_closing)
    root.mainloop()

if __name__ == "__main__":
    main()
```

这个GUI应用程序提供了以下主要功能：

1. 串口连接管理
    - 自动检测可用串口
    - 连接/断开控制
    - 电机使能控制
2. 电机位置控制
    - 滑块和输入框双重控制
    - 实时位置显示
    - 全局速度调节
3. 动作序列管理
    - 保存当前位置
    - 编辑序列数据
    - 运行单个/多个动作
    - 循环执行控制
4. 数据管理
    - 保存/加载CSV格式的动作数据
    - 表格数据编辑
    - 删除/清空功能
5. 预设功能
    - 五指张开
    - 手部校准
    - 急停功能
6. 错误处理

- 连接异常处理
- 数据验证
- 运行时异常处理

7. 资源管理
    - 正确的连接关闭
    - 线程安全控制
    - 界面状态管理

注意事项

1. 安全第一
    - 首次使用时速度设置不要太快
    - 注意观察机械手运动情况
    - 如有异常及时关闭电机出力
2. 资源管理
    - 使用完毕后要正确断开连接
    - 建议使用try-finally结构确保正确清理
3. 错误处理
    - 检查所有API调用的返回值
    - 合理处理异常情况
    - 保持日志记录
4. 性能优化
    - 使用同步控制方法（带_sync后缀）控制多个电机
    - 避免频繁读取位置信息
    - 合理设置运动速度

技术支持

如果遇到问题：

1. 查看错误信息和日志
2. 参考本文档的常见问题解答
3. 检查示例代码
4. 联系技术支持