Dobot Magician 机械臂实验教程

越疆科技

2018 年 南京理工大学

目 录

1 Dobot Magician 机械臂简介 1
1.1 机械臂的机械结构 1
1.2 机械臂的电气系统 3
1.3 机械臂的末端执行器 3
1.4 机械臂的电气参数 11
2 Dobot Studio 的使用 12
2.1 Dobot Studio 简介 12
2.2 Dobot Studio 的安装 13
2.3 机械臂的连接 17
2.3.1 硬件连接 17
2.3.2 软件连接 18
2.4 机械臂的调试 19
2.5 通信接口及配置 21
2.6 机械臂的基本控制 23
3 机械臂的 SDK 概况 25
3.1 SDK 概况 25
3.2 API 函数介绍 25
3.2.1 指令队列控制 25
3.2.2 实时位姿 26
3.2.3 ALARM 功能 26
3.2.4 HOME 功能 26
3.3 使用 API 函数控制机械臂示例 27
4 机械臂的基本控制方法 30
4.1 控制方式概述 30
4.2 队列控制方式函数介绍 30
4.3 队列控制方式示例 32
5 机械臂的信息检测与报警 37
5.1 概述 37
5.2 信息检测与报警函数介绍 39
5.3 信息检测与报警示例 39
6 机械臂的示教方法及实现 43
6.1 概述 43
6.1.1 功能介绍 43
6.1.2 运动模式 43
6.1.3 示教再现高级功能 44
6.2 示教函数介绍 44
6.3 示教模式示例 46
7 机械臂的动作规划与控制 48
7.1 概述 48
7.2 基本函数介绍 48
7.3 单个关节的动作规划与控制 50
7.3.1 Dobot 机械臂 3D 模型的搭建与运动仿真 50

7.3.2 Dobot 机械臂正、逆运动学求解 52
7.4 多关节的动作规划与控制 55
7.5 机械臂动作规划与控制示例 55
8 机械臂的扩展 59
8.1 概述 59
8.1.1 扩展接口简介 59
8.1.2 EIO 接口分布 59
8.1.3 EIO 功能 60
8.2 扩展接口函数介绍 61
8.3 扩展导轨的控制示例 64
8.4 远程控制示例 69
8.4.1 蓝牙功能 69
8.4.2 WiFi 功能 69
9 基于机器视觉的机械臂控制 72
9.1 视觉部件的安装与连接 72
9.2 相机标定及图像处理 78
9.2.1 相机标定 78
9.2.2 颜色特征提取 82
9.3 视觉分拣功能的实现 90
10 多机械臂协作 100
10.1 多机械臂协作系统的构成 100
10.2 多机械臂协作系统的通信架构 100
10.2.1 TCP 连接的建立（三次握手） 101
10.2.2 TCP 连接的释放（四次挥手） 102
10.2.3 PC 端调试助手和机器人通讯 103
10.3 多机械臂协作系统的实现 103
11 机械臂的 LabVIEW 仿真 110
11.1 机械臂的运动学分析与应用 110
11.1.1 实验一：Dobot Magician 3D 模型的搭建 110
11.1.2 实验二：Dobot Magician 3D 模型运动仿真 111
11.1.3 实验三：运动学正、逆解算 112
11.1.4 实验四：机械臂末端运动路径规划 113
11.2 机械臂的动力学分析与应用 114
11.2.1 实验一：步进电机原理与应用 115
11.2.1 实验二：转矩计算与电机选型 116

1 Dobot Magician 机械臂简介

本章主要介绍越疆魔术师（Dobot Magician）机械臂系统的组成和结构，目的是使得读者能够对越疆魔术师机械臂及其接口有一个初步的认识。

1.1 机械臂的机械结构

越疆魔术师机械臂是一个高精度 4 轴桌面智能机械臂，由回转主体、大臂、小臂、臂头、底座等部分组成。机械臂的机械结构分为手臂直线运动结构，回转运动结构，关节型机械臂三个部分。可完成夹取、书写、焊接、分拣、搬运、雕刻，3D 打印等工作，是一种典型的操作型机器人。图 1.1 为 Dobot Magician 机械臂的外观，机械臂各部分结构组成如图 1.2 所示。

图 1.1 Dobot Magician 机械臂

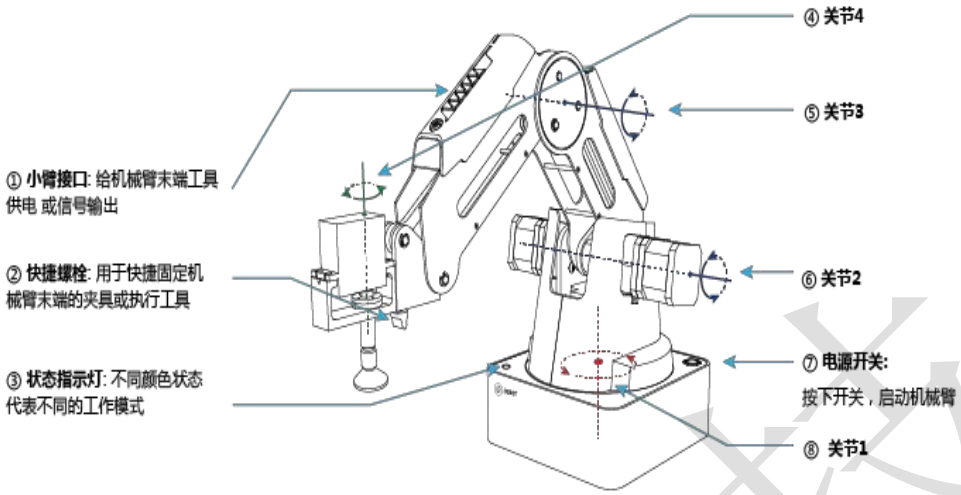

图 1.2　机械臂组成部分

机器臂的运动控制模式分为坐标系控制模式和单轴控制模式。如图 1.3 所示为坐标系控制模式，坐标系原点为大臂、小臂以及底座三个电机三轴的交点，X 轴方向垂直于固定底座向前，Y 轴方向垂直于固定底座向左，Z 轴符合右手定则垂直向上，R 为末端舵机中心相对于坐标原点的姿态，逆时针为正。

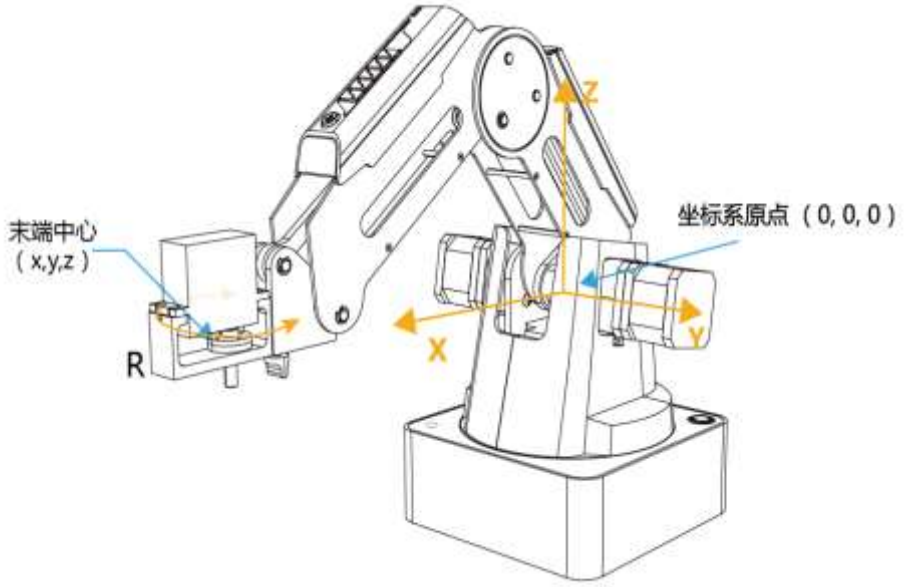

图 1.3　机械臂的坐标系控制示意图

如图 1.4 所示为单轴控制模式，图中的 Joint1、2、3、4 分别对应于底座、大臂、小臂、头部舵机四个独立旋转轴的控制。

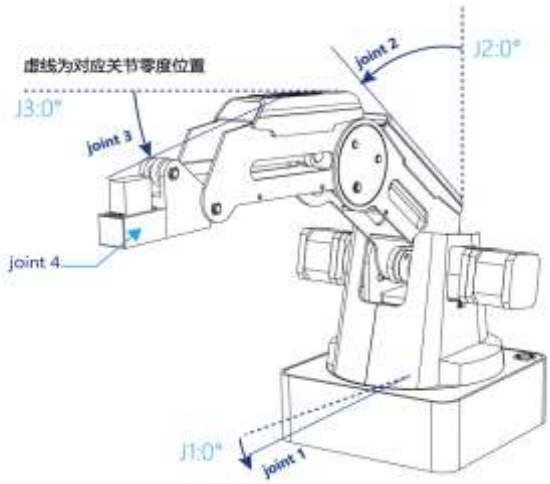

图 1.4　机械臂的单轴控制示意图

如图 1.4 所示，魔术师机器人的本体中心线与底座的正前方中心对应时，关节 1 处于其原点位置，旋转正方向为俯视本体时的关节 1 的逆时针旋转方向；大臂垂直竖立时，关节 2 处于其原点位置，旋转正方向为使大臂朝前朝下运动的方向；小臂处于平行于底座平面时，关节 3 处于其原点位置，旋转正方向为使小臂朝下运动的方向；关节 4 的旋转正方向是俯视本体时关节 4 的逆时针旋转方向。

注 1.1：关节 3 的角度是与水平面的夹角，而不是与关节 2 的相对转角。

1.2　机械臂的电气系统

Dobot Magician 机械臂电气控制系统由四部分组成：交流电机正反转控制电路；交流电机调速控制电路；步进电机控制电路；传感器检测电路。

1.3　机械臂的末端执行器

Dobot Magician 机械臂的末端执行器指连接在操作机手腕的前端，用以执行特定工作任务的工具。机械臂的末端执行器有 3D 打印套件，激光雕刻套件，写字画画套件，吸盘套件以及气动手爪套件。下面分别介绍这几种末端执行器。

（1）3D 打印套件

3D 打印套件包含挤出机，热端，电机线和耗材，材料为 PLA，精度为 0.1mm，如图 1.5 所示。

图 1.5　3D 打印套件

安装步骤如下：

1)　用手压挤出机上面的压杆，将耗材通过滑轮直插到底部通孔，如图 1.6 所示。

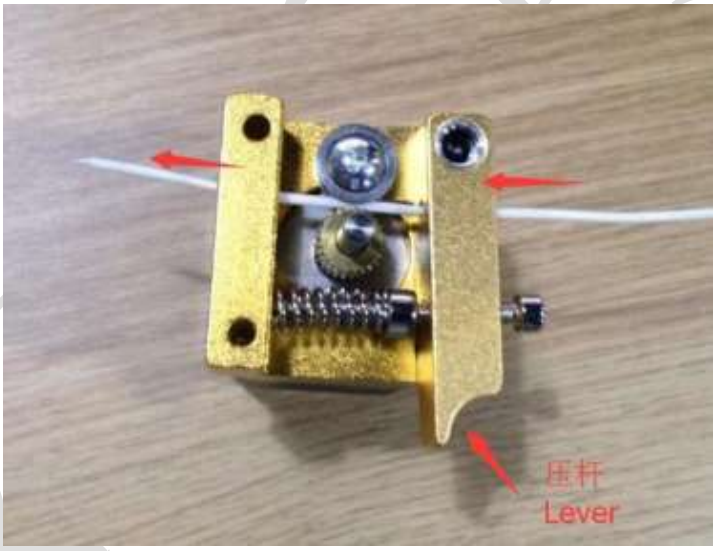

图 1.6　插入耗材

2)　将耗材插入进料管并且插入到加热端的底部，同时将接头拧紧，并将进料管的另一边连接热端，如图 1.7 所示。

注 1.2：进料管与热端的连接一定要插到底，不然会出现出料异常。

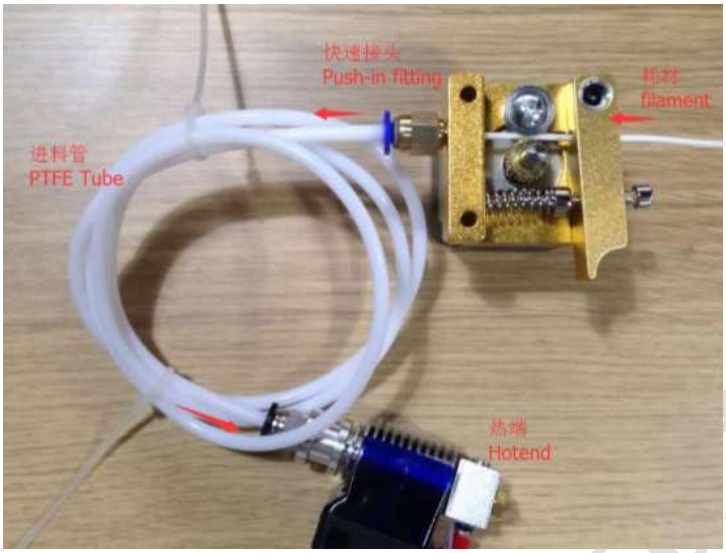

图 1.7　连接挤出机和热端

3) 用蝶形螺母锁紧 3D 加热端于机械臂末端中。

4) 将加热棒的电源线连接到 SW4 接口上，风扇的电源线连接到 GP5 接口上，热敏电阻传感器的连接线接到 ANALOG 接口上，挤出机的连接线接在机械臂底座上的 Stepper1 接口上，如图 1.8 所示。

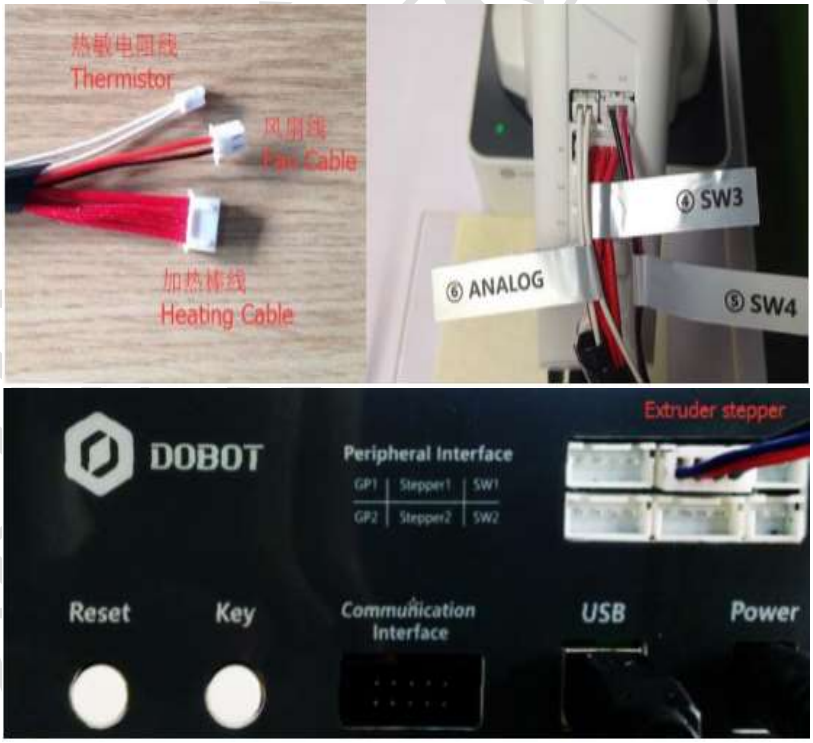

图 1.8　小臂和挤出机接线示意图

5) 3-D 打印套件最终安装效果如图 1.9 所示。

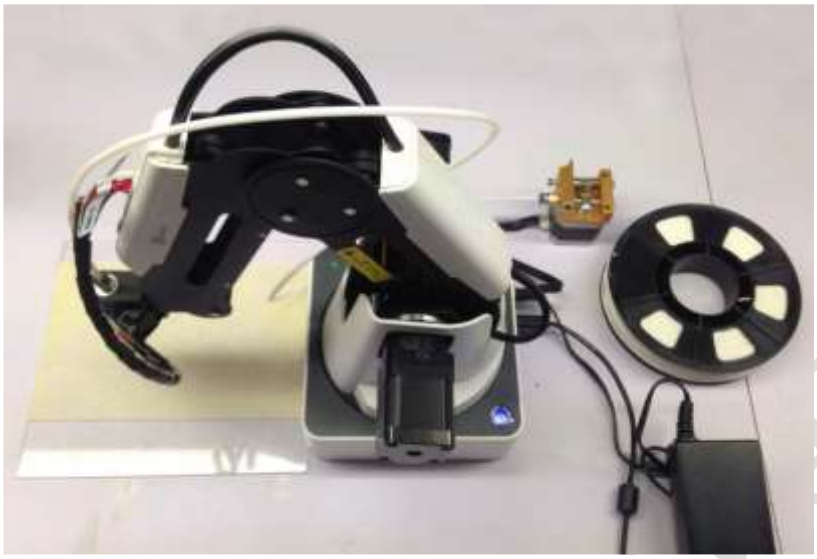

图 1.9　3D 打印套件安装效果

（2）激光雕刻套件

激光雕刻套件包含激光头和夹具，激光功率为 500mw，类型为 405nm 的蓝色激光，由 PWM 调制，电压为 12V，如图 1.10 所示。

图 1.10 激光雕刻套件

安装步骤如下：

1)　将激光套件通过蝶形螺母拧紧在机械臂末端插口中；

2)　将 12V 电源线接在小臂接口 SW4 上，TTL 控制线接在接口 GP5 上，如图 1.11 所示；

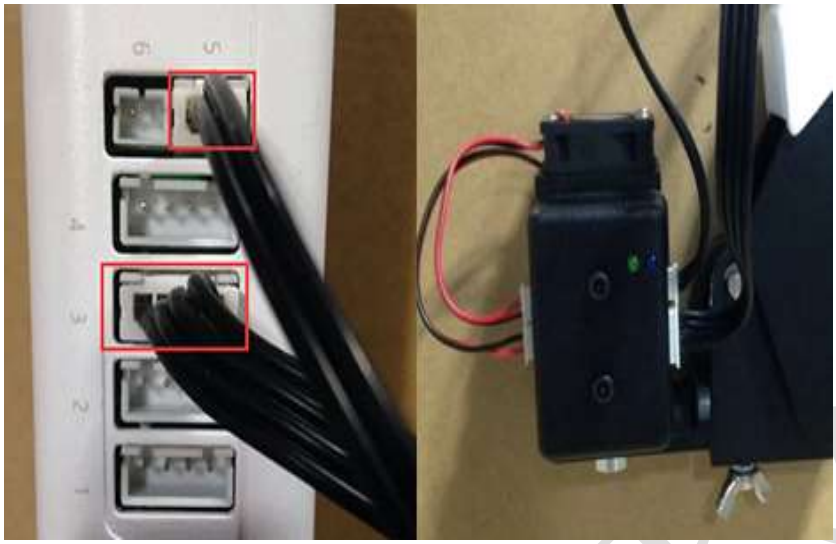

图 1.11　激光雕刻套件安装效果

（3）写字画画套件

写字画画套件包含笔和夹笔器，其中夹笔器的直径为 10mm。

安装步骤如下：

1) 将笔安装在末端夹具中；

2) 将末端夹具插入魔术师机械臂末端安装孔，拧紧蝶形螺母，固定好写字画画末端套件，如图 1.12 所示；

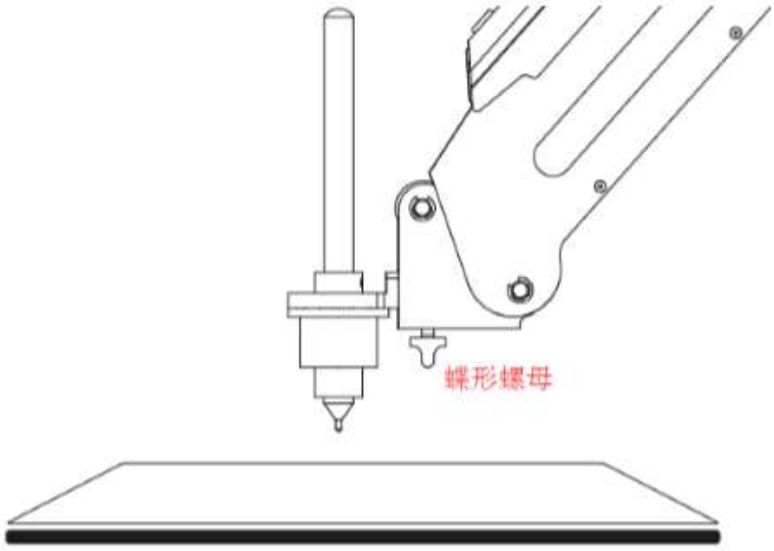

图 1.12　安装写字笔

3) 如果需要更换画笔，请按照以下操作：

i. 松开紧固螺丝，

ii. 更换画笔，

iii. 拧紧螺丝。

如图 1.13 所示。

更换笔，只要拧松夹笔器上面的 4 颗锁附螺钉即可。

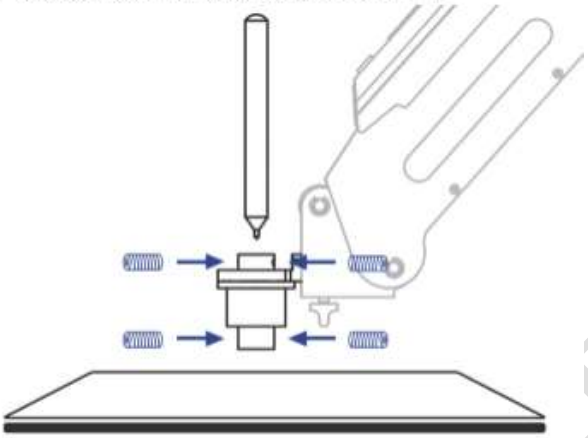

图 1.13　更换写字笔

（4）吸盘套件

吸盘套件包括气泵盒和吸盘，吸盘直径为 20nm，压强为-35kpa，如图 1.14 所示。

图 1.14　吸盘套装

安装步骤如下：

1) 将气泵盒的电源线连接在机械臂底座的 SW1 接口上，信号线接在 GP1 接口上；

2) 将吸盘套件通过蝶形螺母拧紧在机械臂末端插口中，如图 1.15 所示；

3) 将气泵盒的气管连接在吸盘的气管接头上；

4) 将第四轴（Joint4）舵机控制线接在小臂接口 GP3 上即可；

5) 吸盘套件最终安装效果如图 1.15 所示。

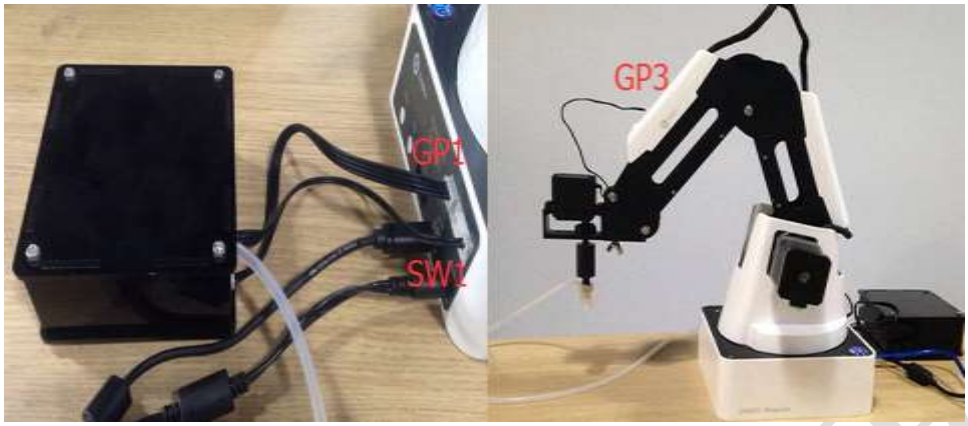

图 1.15 吸盘套件安装效果

（5）气动手爪套件

气动手爪套件包括气泵盒和手爪配件，手爪的张合大小为 27.5mm，驱动方式为气动，力度为 8N，手爪配件如图 1.16 所示。

图 1.16 手爪配件示意图

安装步骤如下：

1) 将气泵盒上的电源线连接在机械臂底座的 SW1 上，末端控制信号线连接到机械臂小臂上的 GP1；
2) 手爪套件需要配合吸盘的气泵套件一起使用，需要先用内六角扳手拧松吸盘上端的固定顶丝，将吸盘从 Joint4 上面拆下来，如图 1.17 所示；

图 1.17　吸盘拆卸示意图

3)　将手爪套件通过联轴器安装在 Joint4 上，如图 1.18 所示。

图 1.18 手爪安装示意图

4)　将气泵盒的气管连接到末端手爪套件的接口上，同时将第四轴舵机控制线插入机械臂小臂上的 GP3 上；

5)　手爪套件最终安装效果如图 1.19 所示。

图 1.19　手爪套件安装效果

1.4 机械臂的电气参数

越疆魔术师机械臂的电气参数如表 1-1 所示。

表 1-1 机械臂的电气参数

参数规格	
轴数	4
最大负载	500g
最大伸展距离	320mm
重复定位精度	0.2mm
通信接口	USB\Wireless
电源电压	100 – 240 V, 50/60Hz
电源输入	12V/7A DC
功率	60W Max
工作温度	-10℃-60℃

越疆科技

2 Dobot Studio 的使用

2.1 Dobot Studio 简介

Dobot Studio 是越疆全系列机械臂产品配套的官方控制软件，其功能界面如图 2.1 所示。

图 2.1　Dobot Studio 主界面

Dobot Studio 支持自由配置机械臂的底层参数，省去繁琐的编程，界面直观并且容易操作，主要包含六大功能模块：

（1）示教再现

用示教的方式操作机械臂实现一系列动作并存点，机械臂便可以重复的去完成刚刚记录的动作。示教再现控制不仅支持传统的点位控制，还支持数字/模拟输入触发、数字输出控制、PWM 输出控制，可满足 70%的应用。

（2）写字&画画

此模块可以控制机械臂写字画画或者激光雕刻，并且可以完成所有预定义大多功能，实现绘画，写字等功能。

（3）Blockly 图形化编程

Dobot Blockly 是为 Dobot Magician 开发的一套图形化编程平台，基于谷歌的开源平台 Google Blockly。通过该平台，用户可以通过拼图的方式进行编程，直观易懂。该平台还整合了机械臂专属的 API，以供用户随时调用。通过该平台，用户通过拼图的方式进行编程，直观易懂，可以实现绝大部分的运动控制与 I/O 输入输出功能，加快应用开发的速度。

（4）脚本控制

编写脚本语言控制机械臂，可以自己定义想要实现的功能，同时 Dobot 社区已经有一些现成的 Demo

程序，仅做简单修改就可以完成功能。

（5）手势控制

基于 leap motion 的手势控制 Dobot Magician 机械臂。

（6）鼠标控制

进入鼠标控制界面，可以看到一个弧形，标有与机械臂坐标轴对应的数据，按下键盘“V”键开始鼠标控制，机械臂跟随鼠标移动；再按下键盘“V”键，停止鼠标控制。

2.2 Dobot Studio 的安装

Dobot Studio 的安装分为两个部分即 Dobot Studio.exe 应用程序的安装和 CH340 驱动包的安装。

安装教程如下：

（1）首先搜索 Dobot 官网，进入“支持”→“下载中心”界面，选择“魔术师机械臂”并下滑至“PC 控制软件”，根据电脑系统版本选择对应的安装软件进行下载。Dobot Studio 下载地址为：https://cn.dobot.cc/downloadcenter.html。

（2）将下载好的压缩文件进行解压，并点击安装 Dobot Studio.exe 应用程序，如图 2.2 所示。

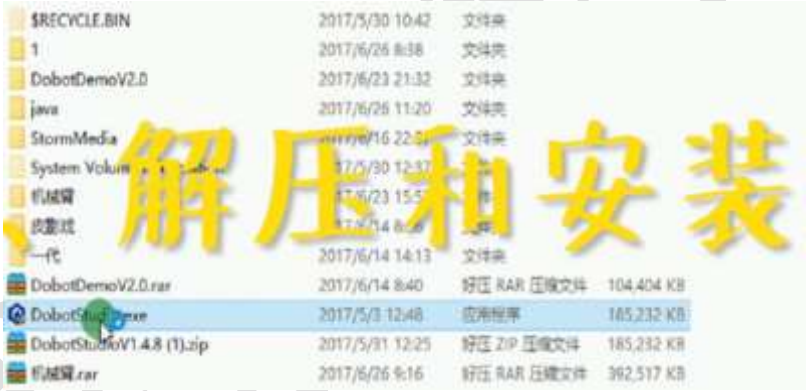

图 2.2　解压安装包

（3）点击安装后会显示如图 2.3 所示的窗口，提示选择 DobotStudio 软件使用语言，这里为了方便理解，选择中文。

图 2.3　语言选择

（4）单击 OK，显示如图 2.4 所示界面，点击“浏览”，选择想要保存的路径，这里默认 C:\Program Files\DobotStudio。

图 2.4　选择保存路径

（5）单击“下一步”，显示如图 2.5 所示界面，选择“创建桌面图标”。

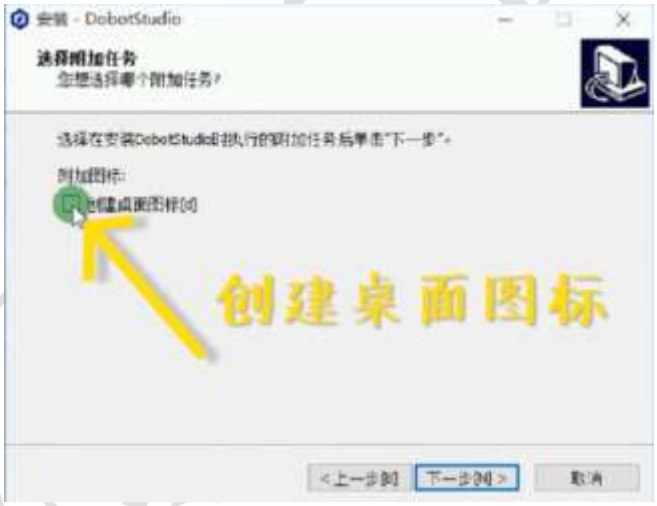

图 2.5　创建桌面图标

（6）单击“下一步”，再点击“安装”，安装进度如图 2.6 所示。

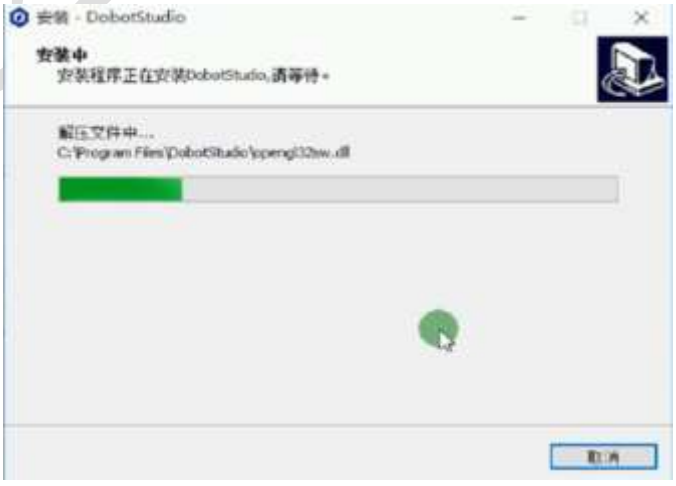

图 2.6　安装进度

（7）安装完成后显示如图 2.7 所示界面，继续安装驱动。

图 2.7　驱动安装界面

（8）驱动如果安装成功，会显示图 2.8 所示界面提示。

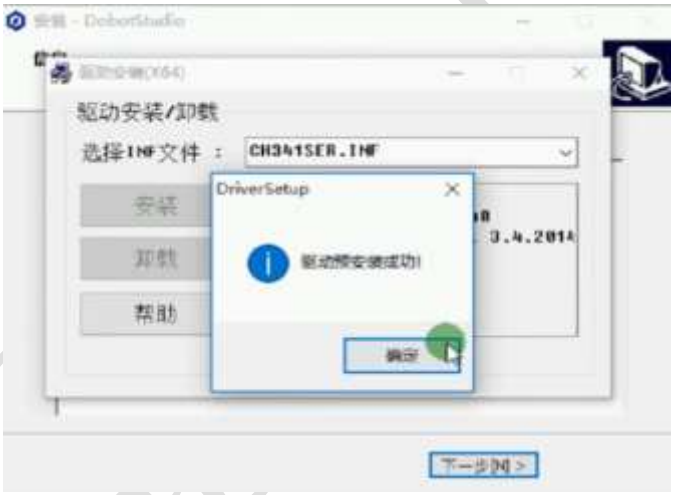

图 2.8　驱动安装成功

（9）完成上述步骤后，显示如图 2.9 所示界面，至此 Dobot Studio 就安装成功了。

图 2.9　安装成功界面

（10）点击桌面图标，运行 Dobot Studio，显示如图 2.10 所示界面，此时用户便可以使用 Dobot Studio 方便快捷的控制机械臂了。

图 2.10　DobotStudio 控制软件

注 2.1：如果打开 Dobot Studio 的时候，报如下错误或其他类似错误：

图 2.11　安装错误提示

请安装 attachment 文件夹下的所有的 vc_redist 安装库，再次打开即可。

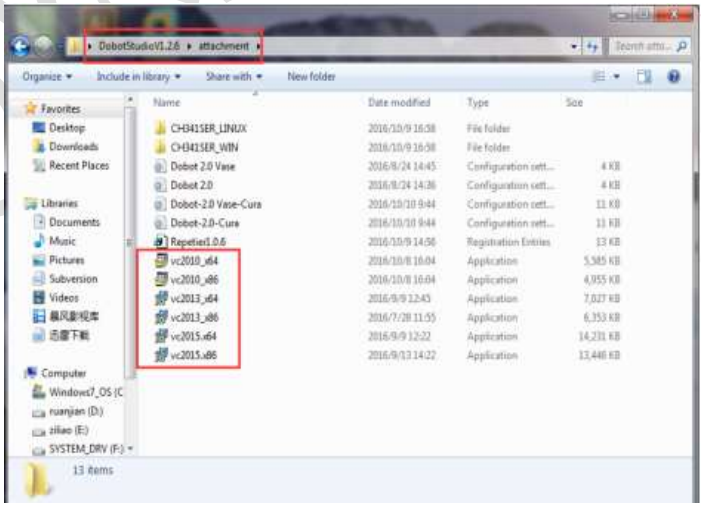

图 2.12　vc_redist 安装库

如果有以下错误并且安装库无法安装，请确保您的系统是原版系统，精简版的可能会导致该错误。解

决方法：安装 VC2015 所需的 netframework4.6.1 组件。

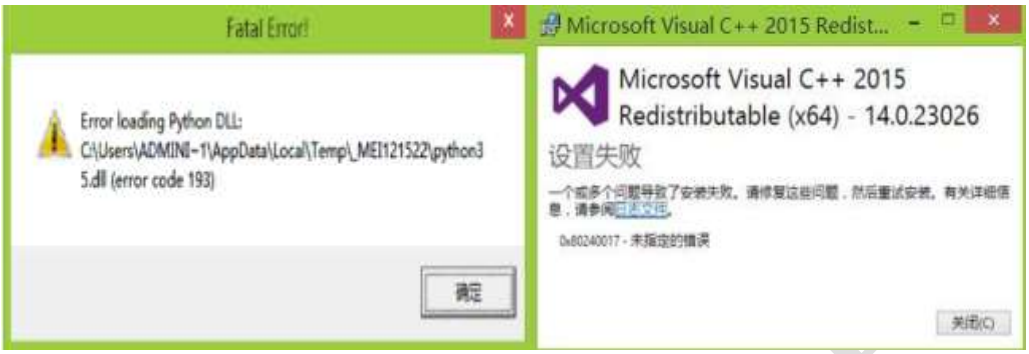

图 2.13　确保系统是原版系统，否则可能导致该错误

2.3　机械臂的连接

2.3.1　硬件连接

将机械臂的电源连接适配器和 USB 线分别连接到电源插座和电脑上，给机械臂末端装上相应的执行器件，提供的末端执行配件有写字画画套件，3D 打印套件，真空气泵套件，手爪和激光套件等，具体安装步骤参考 1.3 章节。

注 2.2：使用前请先参考随箱所附的用户使用手册。

开关机注意事项如下：

（1）开机

用手将机械臂的大小臂放在约 45°的位置并开启主电源，此时所有的电机会锁定。等待约 5 秒后可以听到一声短响，且机械臂的右下方灯由黄色变为绿色后，说明开机正常。如果此时灯为红色，说明机械臂处于限位状态，请确保大小臂在机械臂正常运动范围内。

（2）关机

当机械臂右下方指示灯为绿色时，按下主电源按钮进行关机，机械臂会自动缓慢的收回大小臂到指定位置。此时请注意安全，以防夹手！

（3）如果使用中机械臂的点位读数异常，请按主控盒后面的 Reset 按键，此时机械臂会断开与上位机的连接并复位，然后再重新连接即可。

（4）机械臂底座指示灯具体示意如图 2.14 所示。

蓝色长亮：	表示目前机械臂处于脱机模式	黄色闪烁：	电源电压未供电 或 系统有警告
绿色灯闪烁： 绿色灯常亮：	·机械臂处于正常待机状态 ·已经连接USB，正常工作	红色闪烁：	系统错误

图 2.14　机械臂指示灯的具体示意

2.3.2 软件连接

机械臂与 Dobot Studio 的软件连接，具体步骤如下：

（1）打开 Dobot Studio 软件，记得按下底座上的电源开关，否则当点击界面上“连接”按钮时，Dobot Studio 界面会提示“请连接机械臂”字样，如图 2.15 所示。

图 2.15　机械臂未连接提示

（2）完成步骤（1），连接成功后“连接”按钮会变成“断开连接”字样，同时右侧界面会更新坐标参数，如图 2.16 所示。

图 2.16　Dobot Studio 软件主界面

（3）首次使用机械臂时，可以点击右上角“归零”按钮，使机械臂回到 Home 点位置，当然 Home 点位置也可以在“示教&再现”中自己定义，如图 2.17 所示。

图 2.17　归零设置

（4）点击右上角“设置”按钮，可以对机械臂的各运动参数进行设置，如图 2.18 所示是对再现模式下的各运动参数进行设置。

图 2.18　机械臂参数设置

2.4　机械臂的调试

完成机械臂的软硬件连接后，就可以尝试让机械臂动起来了。这里以“示教&再现功能”为例，实现机械臂搬运一个小物体，具体步骤如下：

（1）打开软件界面，选择示教&再现，主界面如图 2.19 所示。

图 2.19　示教&再现主界面

（2）在示教&再现主界面中找到如下图所示窗口，将末端执行器改为为吸盘（Suction Cup），同时运动模式选择为 MOVJ 模式，如图 2.20 所示。

图 2.20　末端执行器及运动模式选择

（3）将小物体放在机械臂附近，通过按住小臂上的圆形按钮 Unlock key，移动到一定的点时释放，此时主界面会记录一个点，吸盘设置为“释放”。

（4）继续按住机械臂小臂上的圆形按钮，将机械臂移动到小物体上，此时主界面会记录第二个点，吸盘设置为“吸取”。

（5）再按住机械臂小臂上的圆形按钮，将机械臂移动到一定远处时，此时主界面会记录第三个点，吸盘设置为“吸取”。

（6）最后按住机械臂小臂上的圆形按钮，将机械臂向下移动到距离桌面同物体等高处，此时主界面会记录第四个点，吸盘设置为“释放”。

（7）最后右击鼠标，选择“插入”一个运动点，并将坐标设置为初始坐标，吸盘设置为“释放”，这样做的目的是让机械臂回归到初始位置，具体示教运动点如图 2.21 所示。

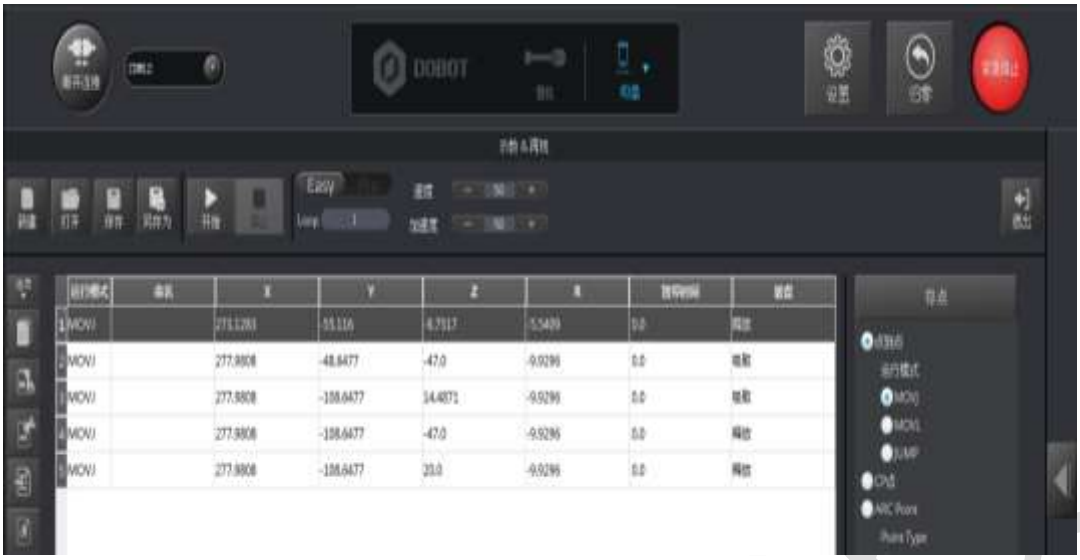

图 2.21　具体示教点

（8）点击“开始”，机械臂能够完成将物体从当前位置搬运到另一位置的任务，效果如图 2.22 所示。

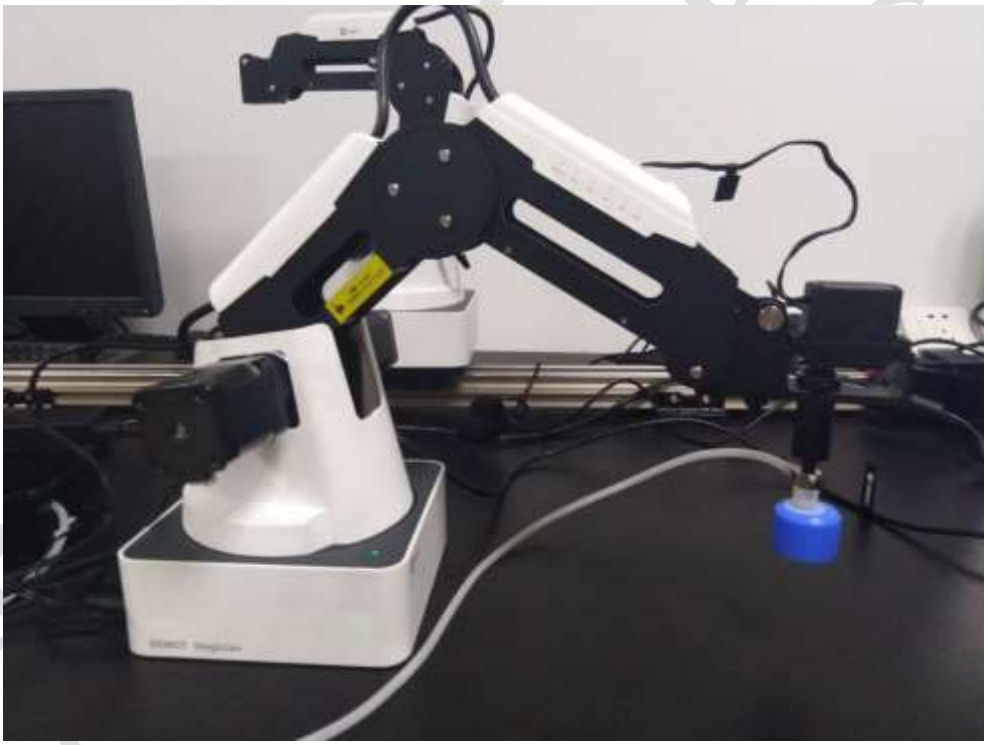

图 2.22　物体搬运效果图

2.5　通信接口及配置

越疆魔术师机械臂共有 13 个拓展接口、1 个可编程按键，分布在机械臂的小臂和底座上，对应的位置如图 2.23 所示。

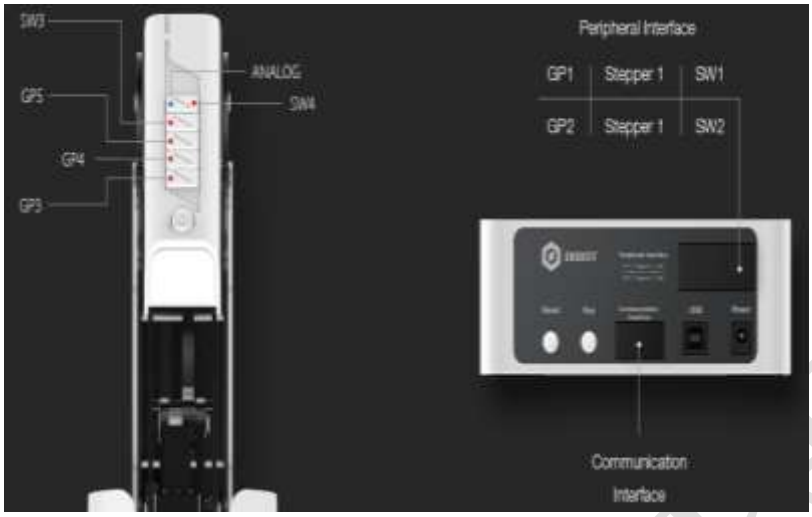

图 2.23　通信接口

小臂区域的接口是用来给机械臂末端所安装的工具供电或传输信号，具体定义如下：

- Peripheral Interface（外设接口）用来连接气泵挤出气体，或连接外部设备，如传感器等。
- Power 接口用来连接配件中的电源适配器。
- USB 接口用来连接电脑与机械臂。
- Reset（复位）按键，用来复位 MCU 的程序。
- Key 是功能按键，长按 2s 时会进入脱机模式。
- Communication Interface 接口用来进行串口通讯、蓝牙和 WIFI。

对于小臂接口与外设接口的具体作用如表 2-1 所示。

表 2-1　小臂接口与外设接口作用一览表

小臂接口		外设接口	
GP3	Joint4； 自定义通用接口	SW1	气泵盒电源输出； 自定义 12V 可控电源输出；
GP4	自定义通用接口	SW2	自定义 12V 可控电源输出；
GP5	激光雕刻信号； 自定义通用接口	STEPPER1	自定义步进电机输出； 3D 打印挤出机
SW3	自定义 12V 可控电源输出； 3D 打印加热端子	STEPPER2	自定义步进电机输出；
SW4	自定义 12V 可控电源输出； 3D 打印加热风扇 激光雕刻电源	GP1	气泵盒控制信号输出； 自定义通用接口
ANALOG	3D 打印温度传感器输出	GP2	自定义通用接口

2.6 机械臂的基本控制

越疆魔术师机械臂有多种不同的控制方式，包括 PC 端鼠标控制，手机 APP，语音控制，手势传感器等，在此简单介绍三种控制功能。

（1）鼠标控制

打开软件主页面，点击鼠标 Mouse 模块，进入鼠标控制界面。开始或结束鼠标控制，请按键盘上的 V 键，右上角的设置选项可以设置相关的 CP 和鼠标参数，如图 2.25 所示。

图 2.24　鼠标控制界面

1）CP 参数：线性加速度；拐角速度；速度；采样数。

2）鼠标参数：缩放比例（鼠标坐标系与机械臂坐标系之间的缩放比）；采样数；设置中心点。

（2）手势控制

首先将 Leap Motion（体感控制）模块通过 USB 线与电脑连接，之后到官网（https://www.leapmotion.com/setup）下载 Leap Motion Windows 驱动并安装。

1）安装成功后回到 Dobot Studio 的主页面，进入手势控制页面，如图 2.25 所示。点击“设置”选项可以设置相应的速度和缩放比例。

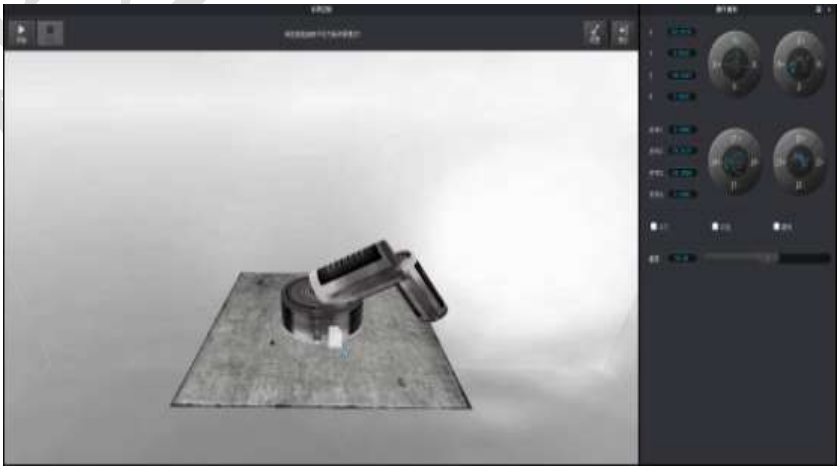

图 2.25　手势控制界面

2）点击“开始”，为保持机械臂的稳定，可以手掌翻向上移入 Leap Motion 控制区域。在 Leap Motion 上方移动手，机械手会向相应的方向移动，握紧拳头，会触发吸盘或机械爪。

3）当手掌翻向上，移出 LeapMotion 检测区域时，可以退出运动。

4）点击“停止”即可退出 Leap Motion 模式。

（3）手机 APP 控制

1）首先进入 DOBOT 官网，进入“支持”→“下载中心”界面，选择“魔术师机械臂”并下滑至“手机软件”处，根据手机系统版本选择对应的软件下载，如图 2.26 所示。

图 2.26 手机软件下载界面

2）安装下载好的 APP，并打开蓝牙，连接手机，如图 2.27 所示。

3）按下机械臂底座的电源键，设置无线模块，并按下复位按钮。

4）首次打开蓝牙连接时，需要输入密钥，连接成功后出现如图 2.27 所示的控制界面。

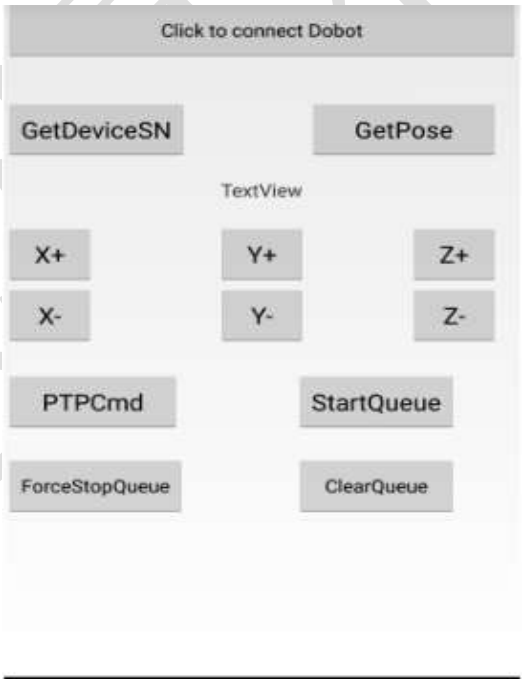

图 2.27 手机 APP 控制界面

这样可以通过手机 APP，控制机械臂“上，下，左，右，旋转”，也可以选择不同的运动模式。

3 机械臂的 SDK 概况

3.1 SDK 概况

SDK（软件开发工具包）是一些软件工程师为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件时的开发工具的集合。越疆魔术师机械臂不仅为用户提供了 Dobot Studio 软件进行控制，还为 Dobot Magician 机械臂的二次开发者提供了 Dobot 常用的 API 和搭建开发环境，用户在官网上可以获取 API 接口文档和 DobotDemo，对于通用桌面系统，越疆官方已经向 Dobot 二次开发者提供了动态链接库。二次开发者只要直接调用动态链接库即可控制 Dobot，而不需进行通讯协议相关的开发工作。

3.2 API 函数介绍

API（应用程序编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。API 函数的存在使得程序员可以直接调用某一些函数或者方法，其中部分的 API 函数在 Dobot Studio 的脚本控制界面左侧菜单栏已列出，如图 3.1 所示。

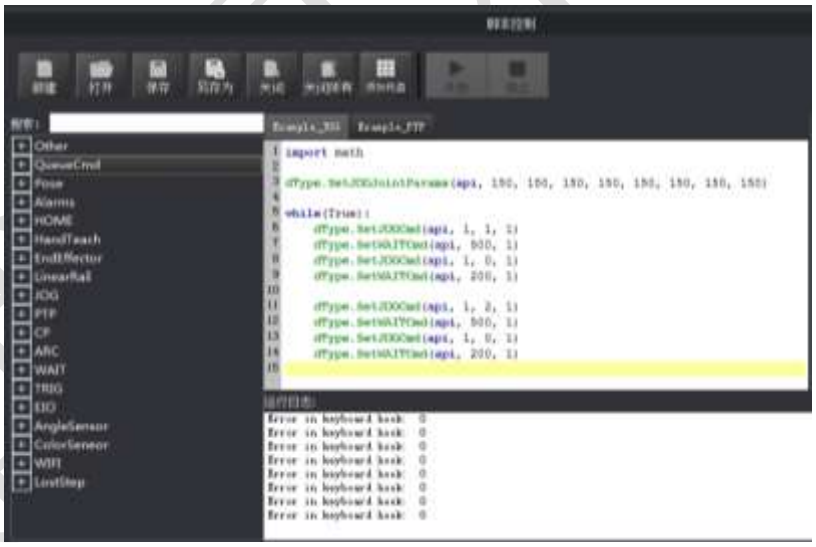

图 3.1 Dobot Studio 脚本控制界面

下面分类介绍 API 函数及其参数含义。

3.2.1 指令队列控制

（1）GetQueuedCmdCurrentIndex（api）：控制器队列计数索引。

其中 api 是 Doobot 库的对象，不可修改。

在 Dobot 控制器指令队列机制中，有一个 64 位内部计数索引。当控制器每执行完一条命令时，该计数器将自动加一。通过该内部索引，可以查询控制器已经执行了多少队列指令，以及当前已经执行到哪条指令（指示运行进度时）。

（2）SetQueuedCmdForceStopExec(api)：强制停止指令队列运行。

无论指令队列是否正在运行一条指令，控制器都会强制其停止运行。

（3）SetQueuedCmdStartExec（api）：启动指令队列运行。

（4）SetQueuedCmdStopExec(api)：停止指令队列运行。

若当前指令队列正在运行一条指令，则其将会在这条指令运行完成后，停止指令队列运行。

（5）SetQueuedCmdStartDownload()：启动指令队列下载。

（6）SetQueuedCmdStopDownload(void)：完成指令队列下载。

（7）SetQueuedCmdClear(void)：清空指令队列。

3.2.2 实时位姿

（1）GetPose(api)：获取实时位姿。

（2）ResetPose（）：重设姿态。

当角度传感器损坏，必须借助外部的角度测量手段或者角度传感器精度太差，借助外部的角度直接/间接测量手段确认其精确值，需要重设姿态。

（3）GetKinematics(api)：获取运动学参数。

3.2.3 ALARM 功能

（1）GetAlarmsState（api,maxLen=1000）：获取系统报警状态。

其中，maxLen:Incoming external buffer length to avoid overflow(输入外部缓冲区长度以避免溢出)。

（2）ClearAllAlarmsState(api)：清除系统所有报警。

3.2.4 HOME 功能

（1）SetHOMEParams(api,x,y,z,r,isQueued=0)：设置回零参数。

其中，x:X 轴坐标；y:Y 轴坐标；z:Z 轴坐标；r:R 末端坐标；isQueued：表示队列模式开关状态。1 是使用队列模式，0 反之。

（2）GetHOMEParams（api）：获取回零参数。

（3）SetHOMECmd（api）：执行回零功能。

3.3 使用 API 函数控制机械臂示例

实验目的:

本实验以机械臂在笛卡尔空间中按照给定的坐标实现点对点（Point to Point，PTP）点动。

实验原理:

使用 Dobot Studio 官方软件中自带的 API 函数来控制机械臂。这里涉及到的 API 函数如下：

（1）dType.SetPTPCommonParams(api, velocityRatio, accelerationRatio, isQueued=0)：设置点位公共参数。

（2）dType.GetPose(api)：获取机械臂实时位姿。

（3）dType.SetPTPCmd(api, ptpMode, x, y, z, rHead, isQueued=0)：执行 PTP 运动指令。

实验步骤:

（1）首先完成机械臂的软，硬件连接，具体步骤参考 2.3 节。

（2）在 Dobot Studio 软件界面中选择“脚本控制”，打开如图 3.2 所示界面。

图 3.2　Dobot Studio 脚本控制界面

（3） 进入脚本控制模块后点击“新建”，新建一个脚本工程，可取名为“TEST1”，并点击“确认”，如图 3.3 所示。

图 3.3　新建脚本工程

（4） 脚本框左边有各类 API 函数，单击 API 函数上的“？”字样，会出现函数的使用说明，双击你的

API 函数便可显示左侧脚本程序输入框中，如图 3.5 所示。

图 3.4　Dobot Studio 脚本 API 函数说明

（5）利用 python 编程语言，编写自己的脚本程序，代码具体示例如下。

```
import math
#设置点位公共参数
dType.SetPTPCommonParams(api, 100, 100)
#设置变量
moveX=0;moveY=0;moveZ=10;moveFlag=-1
#获取机械臂实时位姿
pos = dType.GetPose(api)
x = pos[0]
y = pos[1]
z = pos[2]
rHead = pos[3]
#点动路线与设定循环次数
while(True):
moveFlag *= -1
for i in range(5):
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x+moveX, y+moveY, z+moveZ, rHead, 1)
moveX += 10 * moveFlag
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x+moveX, y+moveY, z+moveZ, rHead, 1)
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x+moveX, y+moveY, z, rHead, 1)
```

（6）程序写好后，单击“开始”按钮，将程序编译，下载到机械臂中。机械臂即按照程序预定的轨迹进

行运动。

实验结果：

你会观察到机械臂在从预定义的初始位置处沿 X 轴和 Z 轴方向循环运动。

越疆科技

4 机械臂的基本控制方法

4.1 控制方式概述

越疆魔术师机械臂的控制器控制方式是采用指令队列机制，机械臂执行队列控制命令用于设置队列命令执行的相关参数，有一个 64 位内部计数索引，当控制器每执行完一条命令时，该计数器将自动加一。通过该内部索引，可以查询控制器已经执行了多少队列指令，以及当前已经执行到哪条指令（指示运行进度）。包括命令执行模式（在线/离线）、队列命令缓冲器当前状态、队列命令执行状态（TRUE / FALSE）、队列命令执行控制（START / PAUSE / STOP）等多种状态。

与 Dobot Magician 机械臂的控制器通信时，通信指令具有以下两个特点：

（1）控制器对所有的指令都有返回。

1）对于设置指令，控制器将指令参数域去掉，并返回；

2）对于获取指令，控制器将该指令将要获取的参数填入到参数域，并返回。

（2）控制器支持两类指令：立即指令与队列指令。

1）立即指令：Dobot 控制器将在收到立即指令时立即处理该指令，而无论当前控制器是否还有其余指令运行中；

2）队列指令：Dobot 控制器在收到队列指令时，将该命令压入控制器内部指令队列中。Dobot 控制器将根据指令压入队列的顺序执行指令。

两种指令的返回值不同。对于立即指令，控制器返回 0，对于立即指令，控制器返回队列命令索引。

4.2 队列控制方式函数介绍

Dobot Studio 内部采用 Python 语言来使机械臂实现更高级的功能以及灵活性，其 API 接口函数都采用 Python 语言实现。Python 是一种面向对象的解释型计算机程序设计语言，在设计中注重代码的可读性，是一种功能强大的通用型语言。

Python 自定义函数的规则如下：

- 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
- 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串——用于存放函数说明。
- 函数内容以冒号起始，并且缩进。
- return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的 return 相当于返回 None。所以 Python 自定义函数的通用语法是：

```
def 函数名称(形参列表):
    函数体
    return［表达式］
```

以获取指令队列索引函数 GetQueuedCmdCurrentIndex( )为例，函数具体的声明和定义如下：

```
def GetQueuedCmdCurrentIndex(api):
    queuedCmdIndex = c_uint64(0)
    while(True):
        result = api.GetQueuedCmdCurrentIndex(byref(queuedCmdIndex))
        if result != DobotCommunicate.DobotCommunicate_NoError:
            dSleep(2)
            continue
        break
    return [queuedCmdIndex.value]
```

Dobot Studio 内部把一系列函数的声明和定义放在 DobotDllType.py，将其作为模块导入，模块导入格式有多种，这里选择实现格式“import 模块名 as 新名称”，目的是导入模块时，并重命名这个模块，具体实现为“import DobotDllType as dType”，调用时需要模块名.方法名（），例如：dType.GetQueuedCmdCurrentIndex(api)。下面具体介绍队列控制方式函数。

（1）dType.GetQueuedCmdCurrentIndex(api)

功能：获取指令队列索引

参数：api：缺省

返回：list[0]，当前指令队列索引

说明：在 Dobot 机械臂的控制器指令队列机制中，有一个 64 位内部计数索引。当控制器每执行完一条命令时，该计数器将自动加一。通过该内部索引，可以查询控制器已经执行了多少队列指令，以及当前已经执行到哪条指令（指示运行进度时）。

（2）dType.SetQueuedCmdStartExec(api)

功能：启动指令队列运行

参数：api：缺省

返回：无

（3）dType.SetQueuedCmdStopExec(api)

功能：停止指令队列运行。 若当前指令队列正在运行一条指令，则其将会在这条指令运行完成后，停

止指令队列运行。

参数：api：缺省

返回：无

（4）dType.SetQueuedCmdForceStopExec(api)

功能：强制停止指令队列运行。 无论指令队列是否正在运行一条指令，控制器都会强制其停止运行。

参数：api：缺省

返回：无

（5）dType.SetQueuedCmdStartDownload(api, totalLoop, linePerLoop)

功能：启动指令队列下载

参数：api：缺省

totalLoop：脱机运行总次数

linePerLoop：单次循环的指令条数

返回：无

说明：Dobot 机械臂的控制器支持将指令存储到控制器外部 Flash 中，而后可以通过控制器上的按键 key 触发执行，也即脱机运行功能。指令下载的一般流程是：

1） 调用启动指令队列下载 API

2) 发送队列指令；

3) 重复，直至队列指令发送完成；

4) 调用完成指令队列下载控制 API。

（6）dType.SetQueuedCmdStopDownload(api)

功能：完成指令队列下载

参数：api：缺省

返回：无

（7）dType.SetQueuedCmdClear(api)

功能：清空 Dobot 控制器中缓存的指令队列。

参数：api：缺省

返回：无

4.3 队列控制方式示例

本实验以机械臂书写为例，介绍队列控制方式的使用及编程。

实验目的：

以机器人书写运动为例，熟悉机械臂部分队列控制指令的使用。

实验方法：

使用 Dobot Studio 中的“脚本控制”，执行自定义的 Python 程序，利用队列指令实现机械臂的 PTP

运动指令的脱机运行。

实验原理：

利用启动队列指令下载指令、完成指令队列下载指令，以及延时指令、获取实施位姿指令、执行 PTP 运动指令等函数，将写字运动的程序下载到机械臂中，实现机械臂的脱机写字运动。这里涉及到的 API 函数如下：

（1）dType.GetPose(api)：获取机械臂实时位姿

（2）dType.SetQueuedCmdStartDownload(api, totalLoop, linePerLoop)：启动指令队列下载

（3）dType.SetPTPCmd(api, ptpMode, x, y, z, rHead, isQueued=0)：执行 PTP 运动指令

（4）dType.SetQueuedCmdStopDownload(api)：完成指令队列下载指令

（5）dType.SetWAITCmd(api, waitTime, isQueued=0)：执行时间等待功能

实验步骤：

（1）安装写字套件（具体步骤参考本实验教程的 1.3 节）；

（2）在 Dobot Studio 选择末端执行器为“笔”，如图 4.1 所示；

图 4.1　末端器具选择

（3）进入 Dobot Studio 的“脚本控制界面”，点击“新建”按钮，如图 4.2 所示。

图 4.2　脚本控制界面

（4）在脚本界面的编辑框内输入如下书写示例程序：

```
#计算点动次数
def fun(A):
 num=0
 for i in range(len(A)):
        for j in range(len(A[i]['x'])):
                num+=1
 return num+len(A)
#“机械”二字的点坐标
j1=dict(x=[4,4,4,4],y=[0,1,6,7],z=[10,0,0,10])
j2=dict(x=[-1,0,15,16],y=[4,4,4,4],z=[10,0,0,10])
j3=dict(x=[4,5,6,7,8,9,10,11],y=[5,4,3,3,2,2,1,0],z=[10,0,0,0,0,0,0,10])
j4=dict(x=[6,7,8,8,9],y=[4,5,6,6,7],z=[10,0,0,0,10])
j5=dict(x=[0,1,12,13,14,15,16],y=[8,8,8,7,7,6,5],z=[10,0,0,0,0,0,10])
j6=dict(x=[1,1,0,13,14,14,11,10],y=[6,8,12,12,13,15,15,15],z=[10,0,0,0,0,0,0,10])
J=[j1,j2,j3,j4,j5,j6]

x1=dict(x=[4,4,4,4],y=[0,1,6,7],z=[10,0,0,10])
x2=dict(x=[-1,0,15,16],y=[4,4,4,4],z=[10,0,0,10])
x3=dict(x=[4,5,6,7,8,9,10,11],y=[5,4,3,3,2,2,1,0],z=[10,0,0,0,0,0,0,10])
x4=dict(x=[6,7,8,8,9],y=[4,5,6,6,7],z=[10,0,0,0,10])
x5=dict(x=[4,4,4,4],y=[6,7,15,16],z=[10,0,0,10])
x6=dict(x=[8,8,8,8],y=[5,6,12,13],z=[10,0,0,10])
x7=dict(x=[5,6,11,12,13,14,15],y=[8,8,8,7,7,6,5],z=[10,0,0,0,0,0,10])
x8=dict(x=[5,6,13,14],y=[10,10,10,10],z=[10,0,0,10])
x9=dict(x=[-1,0,10,11,13,15,12,10],y=[12,12,12,13,13,15,15,15],z=[10,0,0,0,0,0,0,10])
x10=dict(x=[6,7,10,15,16],y=[14,14,14,10,9],z=[10,0,0,0,10])
x11=dict(x=[0,1,2,3],y=[13,14,15,16],z=[10,0,0,10])
X=[x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11]
```

```
    #获取机械臂末端的当前位置
    pos = dType.GetPose(api)
    x = pos[0]
    y = pos[1]
    z = pos[2]
    rHead = pos[3]

    #启动队列下载指令
    dType.SetQueuedCmdStartDownload(api,   1, fun(J)+fun(X))
    #机械臂按字坐标点动指令
    for i in range(len(J)):
     dType.SetWAITCmd(api, 100, 1)
     for j in range(len(J[i]['x'])):
          dType.SetPTPCmd(api,  dType.PTPMode.PTPMOVLXYZMode,  x+J[i]['x'][j],  y+J[i]['y'][j],  z+J[i]['z'][j],
rHead, 1)
    for i in range(len(X)):
     dType.SetWAITCmd(api, 100, 1)
     for j in range(len(X[i]['x'])):
          dType.SetPTPCmd(api,     dType.PTPMode.PTPMOVLXYZMode,     x+X[i]['x'][j],     y+X[i]['y'][j]+20,
z+X[i]['z'][j], rHead, 1)

    #完成指令队列下载指令
    dType.SetQueuedCmdStopDownload(api)
```

（5）调整笔尖位置，按住小臂上的圆形按钮 Unlock key 不放并拖动小臂将笔尖调整至写字纸平面上。

（6）点击“开始”，队列指令将下载到外部 flash 中，按动后端 key 键可脱机运行。

（7）此程序实现先获取机械臂末端的当前位置，然后根据字的笔画坐标执行 PTP 点动指令并下载到机械臂的 flash 中，实现队列指令的脱机运行。

注 4.1：用户可以改变所需书写的字及相应代码，书写其他的汉字或者其他文字。

实验结果：

实验效果如图 4.3 所示。

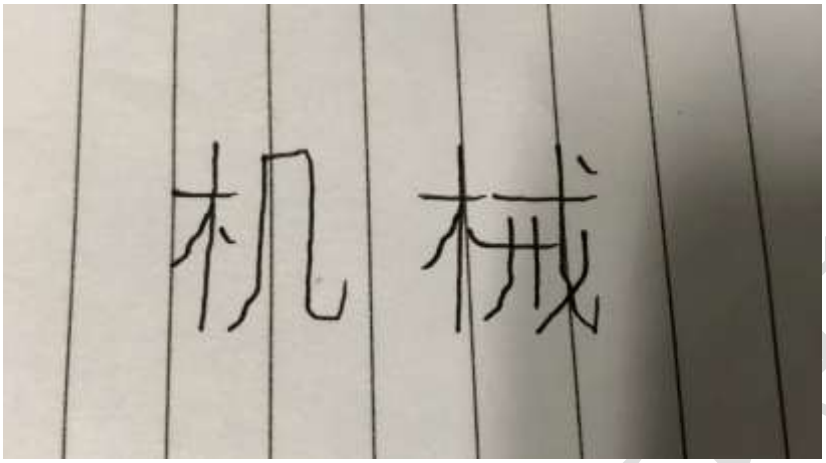

图 4.3　结果显示

越疆科技

5 机械臂的信息检测与报警

5.1 概述

Dobot 控制器会对系统状态进行监控，外部软件可以读取、清除相关的报警状态。在越疆魔术师机械臂中，其报警可以分为以下的几大类，具体分类如下表所示。

表 5-1 报警分类

类别	说明
公共报警	公共部分，如复位等
规划报警	规划过程中计算错误等造成的
运动报警	运动过程中计算错误、限制等造成的
超速报警	各关节实际超速
限位报警	各关节限位

在 Dobot Magician 机械臂的信息检测与报警中：

- 所有的 ALARM 状态存在一个数组中；
- 每个 ALARM 占用 1 个位（bit）；
- 在数组中的每个字节可以标识 8 个报警项的报警状态，且 ALARM 索引小的报警项在低位，ALARM 索引大的报警项在高位；

各类型报警的索引、触发和复位条件如表 5-2 所示。

表 5-2 各类型的报警信息

类别		索引	置位条件	复位条件
公共报警	复位报警	0x00	系统复位后，复位报警自动置位	协议指令清除
	未定义指令	0x01	接收到未定义指令	协议指令清除
	文件系统错误	0x02	文件系统错误	复位，若文件系统初始化成功，则自动复位报警
	MCU 与 FPGA 通信失败	0x03	初始化时，MCU 与 FPGA 通信失败	复位系统，若通信成功，则自动复位报警
	角度传感器读取错误	0x04	无法正确读取角度传感器值	断电，重新上电，若传感器读数正常，则复位报警
规划报警	逆解算报警	0x11	规划目标点不在机器人工作空间等导致的逆解算错误	协议指令清除

类别		索引	置位条件	复位条件
	逆解算限位	0x12	规划目标点逆解算超出关节限位值	协议指令清除
	数据重复	0x13	协议指令重复	协议指令清除
	圆弧输入参数报警	0x14	输入圆弧中间点，目标点无法构成圆弧	协议指令清除
	JUMP 参数错误	0x15	规划起始点或者目标点高度高于设置的 JUMP 最高高度	协议指令清除
运动报警	逆解算报警	0x21	运动过程中超出机器人工作空间等导致的逆解算错误	协议指令清除
	逆解算限位	0x22	运动过程中逆解算超出关节限位值	协议指令清除
限位报警	关节 1 正向限位报警	0x40	关节 1 正向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 1 负向限位报警	0x41	关节 1 反向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 2 正向限位报警	0x42	关节 2 正向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 2 负向限位报警	0x43	关节 2 反向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 3 正向限位报警	0x44	关节 3 正向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 3 反向限位报警	0x45	关节 3 反向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 4 正向限位报警	0x46	关节 4 正向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	关节 4 反向限位报警	0x47	关节 4 反向运动到达限位区域	协议指令手动复位报警；反向退出限位区域自动复位报警
	平行四边形正向限位报警	0x48	平行四边形拉伸运动到达变形极限位置	协议指令手动复位报警；收缩退出限位区域自动复位报警
	平行四边形负向限位报警	0x49	平行四边形收缩运动到达变形极限位置	协议指令手动复位报警；拉伸退出限位区域自动复位报警

5.2 信息检测与报警函数介绍

与队列控制方式函数定义相似，Dobot Studio 内部把信息检测与报警函数的声明和定义也放在DobotDllType.py，将其作为模块导入，具体实现为“import DobotDllType as dType”，调用时需要模块名.方法名（），即 dType.GetAlarmsState(api, maxLen=1000)。

下面具体介绍信息检测与报警函数。

（1）dType.GetAlarmsState(api, maxLen=1000)

功能：获取系统报警状态

参数：api：缺省

maxlen：为避免溢出 传入的外部缓冲区的长度

返回：list[0]：用于接收各报警位

list[1]：警报所占字节

（2）dType.ClearAllAlarmsState(api)

功能：清除系统所有报警

参数：api：缺省

返回：无

5.3 信息检测与报警示例

本实验对规划、运动、限位三种类型的故障报警进行具体说明。

实验目的：

熟悉并掌握 Dobot Magician 机械臂规划、运动、限位三种报警类型的信息检测。

实验方法：

使用 Dobot Studio 中的“脚本控制”，使用了 Python 队列控制指令，对故障进行模拟以及完成报警信息的检测。

实验原理：

例程调用各种类型的报警索引，发出故障模拟指令，使机械臂产生规划参数错误、逆解算限位、运动至极限位置等故障，通过获取系统报警状态的 API 函数来确定具体报警类型。这里涉及到的 API 函数如下：

（1）dType.SetARCCmd(api, cirPoint, toPoint, isQueued=0)：执行圆弧插补功能

（2）dType.GetPose(api)：获取当前位姿

（3）dType.SetPTPCoordinateParams(api, xyzVelocity, xyzAcceleration, rVelocity, rAcceleration, isQueued=0)：设置坐标轴点位参数

（4）dType.SetPTPCmd(api, ptpMode, x, y, z, rHead, isQueued=0)：执行 PTP 运动指令

（5）dType.SetJOGJointParams(api, j1Velocity, j1Acceleration, j2Velocity, j2Acceleration, j3Velocity,

j3Acceleration, j4Velocity, j4Acceleration, isQueued=0)：设置关节点位参数

（6）dType.SetJOGCmd(api, isJoint, cmd, isQueued=0)：执行点动指令

（7）dType.GetAlarmsState(api, maxLen=1000)：获取系统报警状态

实验步骤：

（1）参考 2.3 节，完成机械臂的软硬件连接。

（2）进入 Dobot Studio 的“脚本控制界面”，新建如下报警测试程序：

```
from math import log
err=0
alarm=[ ]        #实际报警索引存放的列表
ALA={}                #所有报警索引存放的字典
ALA.setdefault(0x00,'复位报警')
ALA.setdefault(0x01,'未定义指令')
ALA.setdefault(0x02,'文件系统错误')
ALA.setdefault(0x03,'MCU 与 FPGA 通信失败')
ALA.setdefault(0x04,'角度传感器读取错误')

ALA.setdefault(0x11,'规划逆解算报警')
ALA.setdefault(0x12,'规划逆解算限位')
ALA.setdefault(0x13,'规划数据重复')
ALA.setdefault(0x14,'规划圆弧输入参数报警')
ALA.setdefault(0x15,'规划 JUMP 参数错误')

ALA.setdefault(0x21,'运动逆解算报警')
ALA.setdefault(0x22,'运动逆解算限位')

ALA.setdefault(0x40,'关节 1 正向限位报警')
ALA.setdefault(0x41,'关节 1 负向限位报警')
ALA.setdefault(0x42,'关节 2 正向限位报警')
ALA.setdefault(0x43,'关节 2 负向限位报警')
ALA.setdefault(0x44,'关节 3 正向限位报警')
ALA.setdefault(0x45,'关节 3 负向限位报警')
ALA.setdefault(0x46,'关节 4 正向限位报警')
ALA.setdefault(0x47,'关节 4 负向限位报警')
ALA.setdefault(0x48,'平行四边形正向限位报警')
```

```
ALA.setdefault(0x49,'平行四边形负向限位报警')


#规划报警的故障模拟 圆弧插补参数错误
a=[100,100,100,100]
b=a
dType.SetARCCmd(api, a, b, 1)


#运动报警的故障模拟 规划逆解算限位
#pos=dType.GetPose(api)
#dType.SetPTPCoordinateParams(api, 20, 20, 20,  20,  1)
#dType.SetPTPCmd(api, 3, pos[0]+1000, pos[1], pos[2], pos[3], 1)


#限位报警的故障模拟 关节 1 负向限位
#dType.SetJOGJointParams(api, 10, 10, 10, 10, 10, 10, 10, 10, 1)
#dType.SetJOGCmd(api, 1, 2, 1)


#获取系统报警状态
ala=dType.GetAlarmsState(api,  maxLen=1000)


#计算报警索引并打印
print(' 报警所占总字节数:',ala[1] , '  检测结果如下:')
for i in range(1000):
    if ala[0][i]!=0:
        err=1
        index=i*8+int(log(ala[0][i],2))
        print(' 检测到报警 索引为:0x%x'%index,ALA[index])
        break
    else:
        pass
    if(i==999):
        print(' 没有检测到报警\n')
```

（3）分别保留规划报警的故障模拟、运动报警的故障模拟、限位报警的故障模拟程序运行。

注 5.1：保留其中一种故障类型时，其余两种故障类型需要在程序中注释掉

实验结果：

（1）对于规划报警的故障模拟程序，点击“开始”按钮，编译并下载程序，此时机械臂不执行任何动作。因为程序中参数 a 为圆弧上任意一点坐标，b 为目标点坐标，当使 b=a 时，圆弧半径规划为无穷，因此将导致规划圆弧输入参数报警，并且运行日志中会打印报警信息，如图 5.1 所示。

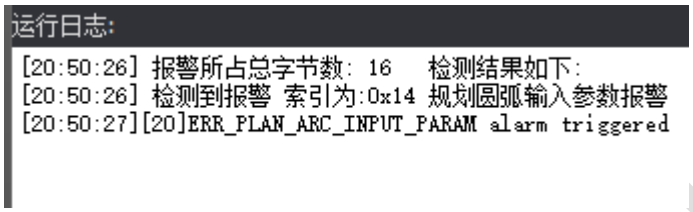

图 5.1 规划报警

（2）对于运动报警的故障模拟程序，点击”开始”，编译并下载程序，由于给定坐标参数 pos[0]+1000 超出关节限位值，此时机械臂不执行任何动作。并发出规划逆解算限位报警，运行日志中打印报警信息，如图 5.2 所示。

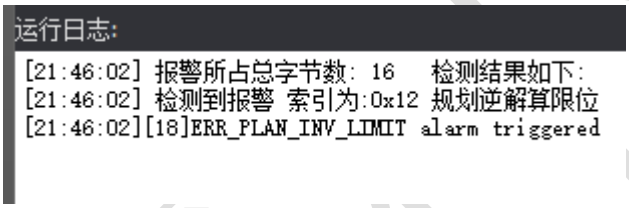

图 5.2 运动报警

（3）对于限位报警的故障模拟程序，点击”开始”，编译并下载程序，机械臂关节 1 负向运动，到达极限位置时停止，并发出关节 1 限位报警，运行日志中打印报警信息，如图 5.3 所示。

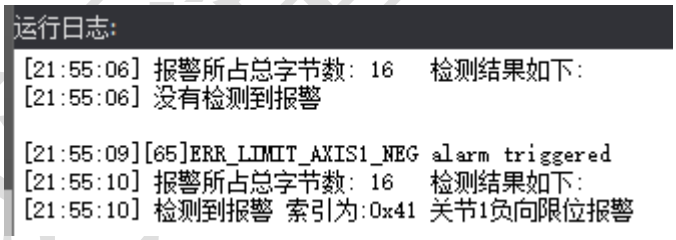

图 5.3 限位报警

6 机械臂的示教方法及实现

6.1 概述

6.1.1 功能介绍

Dobot Magician 机械臂的操作环境集成了非常强大的“示教&再现”功能模块，示教&再现方法的核心即只要手持机械臂进行示教，它就会复现同样的动作序列，并且可以自由设置重复执行的次数，无需再另外编程。示教&再现方法的另一个重要特性是配合使用矢量图，即使用矢量图时可以插入一段示教再现方法生成的脚本，十分的实用。

示教&再现方法的特点：

（1）操作直观，学习成本低，可以配合其他方法使用；

（2）重现的点有微小偏差，当示教点过多时操作较为复杂。

示教&再现方法的操作：

进入 Dobot Studio 的“示教&再现界面”，此时按住小臂的圆形解锁按钮（UnlockKey），拖动机械臂到任意位置，松开按钮就可以自动保存一个存点。

6.1.2 运动模式

示教&再现模块提供了 PTP，CP，ARC 的运行模式，其中 PTP 包含了 MOVJ，MOVL，JUMP 的运行方式。ARC 包含了 cirPoint 和 toPoint 的运行方式，下面将分别说明各个运行方式的功能。

（1）MOVJ：关节运动

由 A 点运动到 B 点，各个关节从起始位置 A 对应的关节角运行到结束位置 B 的关节角，运动过程中要求各轴运行时间一致，同时到达终点。

（2）MOVL：直线运动，A 点到 B 点的路径是直线。

MOVJ 和 MOVL 两种运动方式简化示意图如图 6.1 所示。

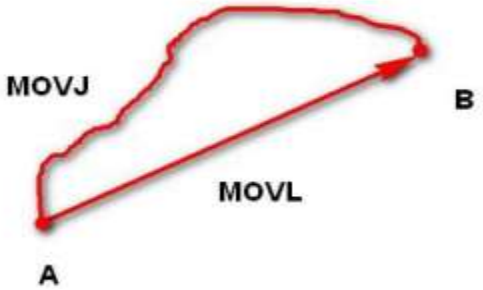

图 6.1 关节运动与直线运动

（3）JUMP：门型轨迹

如图 6.2 所示，由 A 点到 B 点的 JUMP 运动，等价于先设置一个 Height，将 A 点和 B 点的竖坐标 Z 加上 Height，分别得到 C 点和 D 点，再做以 A-C-D-B 为轨迹的三个 MOVL 运动。

图 6.2　门型轨迹

（4）CP：连续轨迹功能的指令，用于连续轨迹相关的运动设置和配置。

其中包括关节参数、坐标系参数、功能设置参数等。连续轨迹功能和上位机 Dobot CP 对应，可以实现写字、画画、激光雕刻等功能。

（5）ARC：设置和获取圆弧插补功能参数

6.1.3　示教再现高级功能

通过“简易/高级（Easy/Pro）”按钮可以切换到高级功能接口，其中单步运行功能使用方法如下：

- 点击“单步运行”按钮会进入单步执行模式，首次单击该按钮一次会执行第一个存点，随后每次单击该按钮可以执行下一个存点。
- 点击“停止”按钮可以退出单步执行模式。

脱机功能：

- 点击“下载”按钮可以将当前的存点列表下载到机械臂中，这样就可以实现存点脱机执行，无需插入 USB 线。
- 下载完成后，拔掉机械臂的 USB 线，按下主控盒后面的功能按钮（Key）开始执行下载的程序，再按一次停止执行。

注 6.1：在进行存点和脱机执行之前，都需要使用归零按钮将机械臂复位，这样才能保持存点复现时候坐标系的一致性，才能真正做到精确轨迹复现。

6.2　示教函数介绍

与队列控制方式函数定义相似，Dobot Studio 内部把示教函数的声明和定义也放在 DobotDllType.py，将其作为模块导入，具体实现为“import DobotDllType as dType”，调用时需要模块名.方法名（），即

dType.SetHHTTrigMode (api, hhtTrigMode)。下面具体介绍信息检测与报警函数。

（1）dType.SetHHTTrigMode (api, hhtTrigMode)

功能：设置手持示教触发模式，可设置为按键释放时更新或定时更新；

参数：api：缺省

hhtTrigMode：手臂方向

0：按键释放时更新

1：定时触发

返回：无

（2）dType.GetHHTTrigMode(api)

功能：获取当前手持示教触发模式；

参数：api：缺省

返回：list[0]：手持示教时的存点触发模式

0：当前为按键释放时更新

1：当前为定时触发

（3）dType.SetHHTTrigOutputEnabled(api, isEnabled)

功能：设置触发输出使能/禁止；

参数：api：缺省

isEnabled：触发输出使能状态

0：禁止触发输出

1：使能触发输出

返回：无

（4）dType.GetHHTTrigOutputEnabled(api)

功能：获取触发输出使能/禁止；

参数：api：缺省

返回：list[0]：触发输出使能状态

0：当前禁止触发输出

1：当前使能触发输出

（5）dType.GetHHTTrigOutput(api)

功能：获取触发输出状态；

参数：api：缺省

返回：list[0]：触发输出状态

0：有触发

1：无触发

越疆科技

6.3 示教模式示例

实验目的：

本实验以机器人画一个五角星为例，介绍示教模式的使用。

实验方法：

使用机械臂进行存点步骤如下：

（1）开启机械臂，绿灯亮后等待约 20 秒，等传感器稳定后再点击归零按钮进行归零；

（2）归零过程中，机械臂闪烁蓝灯，同时在零点附近，机械臂会自动进行微调，等待绿灯亮起后，才可以继续操作；

（3）归零完成后进行存点，然后下载。在弹出的对话框“脱机模式运行前是否先回零机械臂？”中需选择“确认”选项：

（4）复现过程即开启机械臂，绿灯亮后等待约 20 秒，等传感器稳定后再点击 key 键进行脱机复现。

实验原理：

使用 Dobot Studio 中的“示教&再现”，移动机械臂进行存点，下载存点列表，实现脱机运行。

实验步骤：

（1）将笔套件安装到机械臂上并正确将机械臂与 PC 连接，具体步骤参考 2.3 章节；

（2）在 Dobot Studio 中选择末端执行器为“笔”；

（3）进入 Dobot Studio 的“示教&再现界面”，如图 6.3 所示。

图 6.3 示教&再现说明

（4）界面左上角选项栏中选择“Easy”模式；

（5）在右侧“存点”操作界面中选择运动方式为“点到点”—“MOVL”，如图 6.4 所示；

图 6.4 运动模式选择

（6）按住机械臂小臂上的圆形按钮 Unlock key，每到需要的点时松开按钮，会自动存点，在纸上画出想要的图形；

（7）记录完毕后，在“设置”修改机械臂的最高速度、加速度及重复次数等参数；

（8）点击左上角选项栏中的“开始”，机械臂会复现同样的动作序列。

实验结果：

实验效果如图 6.5 所示。

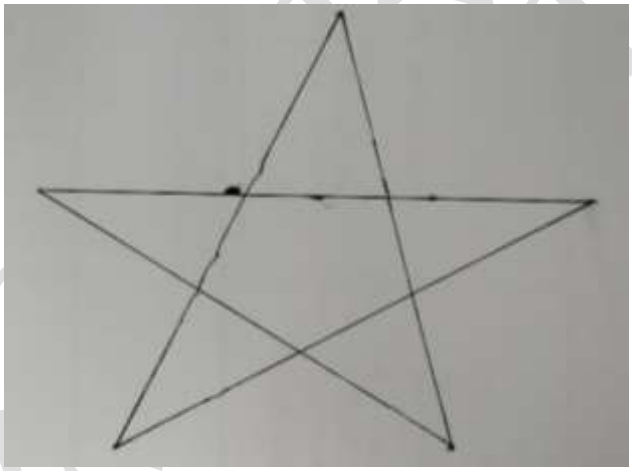

图 6.5　示教&再现结果

7 机械臂的动作规划与控制

7.1 概述

本章主要介绍机械臂单个关节和多个关节的动作规划与控制，动作规划属于机械臂高级规划的范畴。任务规划分两个层次，当机械臂把复杂的任务分解成一系列子任务时，这一层次的规划称为任务规划，然后再将每一个子任务分解为一系列动作时，这一层次的规划称为动作规划。其中每个动作的顺利实现，都离不开底层的路径规划和轨迹规划。

本章还介绍 Dobot Magician 机械臂的 SDK 中提供的关于机械臂动作规划与控制的一些基本函数，用户只需要指定坐标系、目标点等参数就可以让机械臂按指定的轨迹运动，操作非常便利。

7.2 基本函数介绍

与队列控制方式函数定义相似，Dobot Studio 内部把示例函数的声明和定义也放在 DobotDllType.py，将其作为模块导入，具体实现为“import DobotDllType as dType”，调用时需要模块名.方法名（），如 dType.GetEndEffectorLaser(api)，下面具体介绍基本函数。

（1） dType.GetEndEffectorLaser(api)

功能：获取激光开关状态。

参数：api：缺省

返回值：list[0]：两种可能结果

0：关

1：开

（2） dType.SetEndEffectorLaser(api, enableCtrl, on, isQueued=0)

功能：设置激光开关。

参数：api：缺省

enableCtrl：使能控制

0：Disable

1：Enable

on：开关状态

0：关

1：开

isQueued：队列模式使用状态

0：使用队列模式

1：不使用队列模式

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（3） dType.SetCPParams(api, planAcc, juncitionVel, acc, realTimeTrack = 0, isQueued=0)

功能：设置连续轨迹轨迹功能参数

参数：api：缺省

planAcc：规划加速度最大值

juncitionVel：拐角速度最大值

realTimeTrack：设置是否选择实时模式

acc：周期采用比率

realTimeTrack:在非实时模式下代表实际加速度最大值,实时模式下代表插补周期(单位 ms)。

isQueued：队列模式使用状态

0：使用队列模式

1：不使用队列模式

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（4） dType.SetCPLECmd(api, cpMode, x, y, z, power, isQueued=0)

功能：执行 CP 运动指令，激光雕刻使用。

参数：api：缺省

cpMode：模式

0：相对模式

1：绝对模式;

x, y, z：目标点坐标

power：激光功率

isQueued：队列模式使用状态

0：使用队列模式

1：不使用队列模式

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（5）dType.SetARCCmd(api, cirPoint, toPoint, isQueued=0)

功能：执行圆弧插补功能。

参数：api：缺省

cirPoint：当前点坐标

toPoint：目标点坐标。

isQueued：队列模式使用状态

0：使用队列模式

1：不使用队列模式

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

7.3 单个关节的动作规划与控制

Dobot Magician 机械臂有两个参考坐标系，分别为笛卡尔参考坐标系和关节参考坐标系。笛卡尔参考坐标系通过轴参数来控制 Dobot Magician 机械臂的多个关节的同时运动。关节参考坐标系用来描述机械手每个独立关节的运动，Dobot Magician 机械臂中对应于底座、大臂、小臂和头部舵机有四个独立的关节，各关节均以逆时针方向为正方向。

如果 Dobot Magician 机械臂需要从当前位置移动到目标位置，可以将路径分成若干小段，并使机械臂的运动经过所有的中间点。Dobot Magician 提供的 SDK 会帮助用户求解逆运动学方程，将这些点的全局坐标转换成关节坐标。最后，机械臂会根据获得的关节坐标对四个关节逐点进行角度控制。机械臂每次只调节单个关节的角度，锁定其他关节的角度，那么机械臂的运动控制就变得很简单。

当然无论是笛卡尔参考坐标系还是关节参考坐标系，都离不开对机械臂运动学分析，接下来介绍机械臂的运动学分析。

7.3.1 Dobot 机械臂 3D 模型的搭建与运动仿真

对 Dobot Magician 机械臂进行运动学分析其实就是描述机器人各连杆之间的相对位置和方向关系，因此需要根据关节结构在每个连杆上建立一个坐标系，一般建立 D-H 关节坐标系，其中限位说明如表 7-1 所示，D-H 建模如图 7.1 所示。

表 7-1 限位说明

关节	负限位°	正限位°
底座 J1	-125	125
大臂 J2	-5	90
小臂 J3	-15	90
末端 J4	-150	150

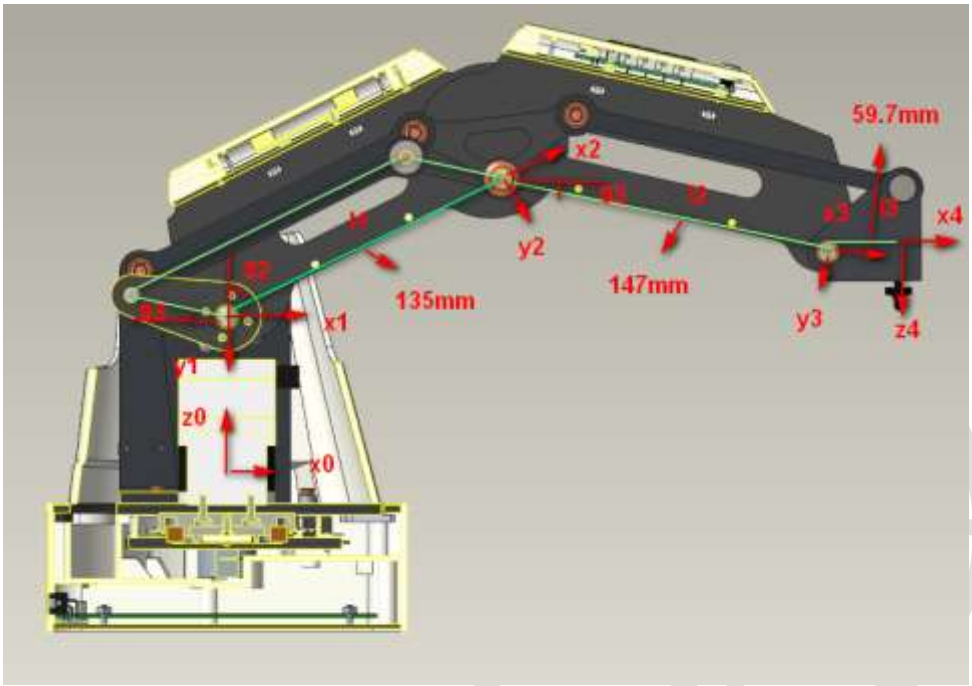

图 7.1　D-H 建模

D-H 建模方法是由 Denavit 和 Hartenberg 提出的一种建模方法，即在每个连杆上建立一个坐标系，通过齐次坐标变换矩阵来实现两个连杆上坐标的变换，在多连杆串联的系统中，多次使用齐次坐标变换，就可以求出机械臂末端与基座位姿关系。D-H 建模的几大知识点如下：

（1）齐次坐标变换

齐次坐标系就是把 n 维向量用 n+1 维表示，齐次坐标表示方法的主要优点就是将旋转变换和平移变换统一成了 4×4 的矩阵运算。齐次变换矩阵如下所示：

$$ {}_{i}^{j}T = \begin{bmatrix} {}_{i}^{j}R & {}^{o_j}{}_{i}P \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}_{4\times4} \tag{7.1} $$

4×4 的齐次变换矩阵中左上角的 3×3 矩阵表示旋转，最右边一列表示平移，最后一行是固定写法。

（2）需要知道 D-H 建模中 a，α，d，θ这四个参数对应于连杆上的那些地方。这里假设当前关节标号为 i-1，下一个关节标号为 i，简单的连杆关节如图所示。

1） 连杆 i-1 的长度，指图中 $a_{(i-1)}$所示的距离。

2） 连杆 i-1 的扭角，指图中 $\alpha_{(i-1)}$所示的角度。

3） 连杆 i 相对于连杆 i-1 的偏置，指图中的 di。

4） 关节角θi，指图中连杆 i 相对于连杆 i-1 绕 i 轴的旋转角度。

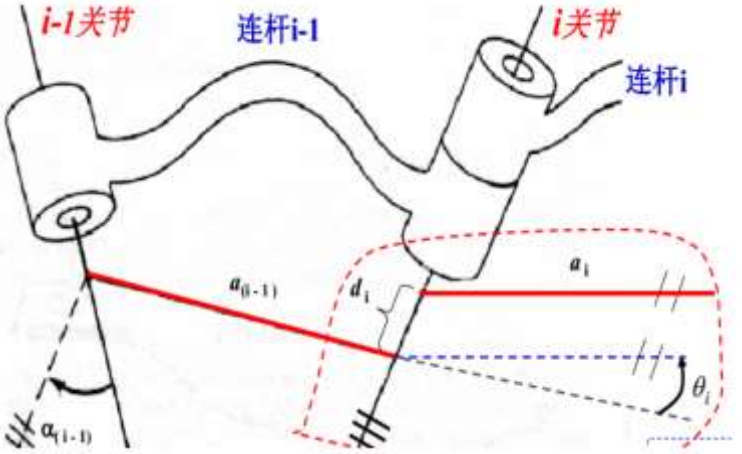

图 7.2　简单的连杆关节示意图

（3）针对 Dobot Magician 机械臂的连杆机构，可以列出如表 7-1 所示的 D-H 参数表。

表 7-2 Dobot 机械臂 D-H 参数表

L	θi	αi	di	ai
1	θ1	$-\frac{\pi}{2}$	8	0
2	θ2	0	0	135
3	θ3	0	0	147
4	θ4	$-\frac{\pi}{2}$	0	59.7

（4）列出最终坐标变换的等式，建立机械臂末端与基座的坐标关系。

根据 D-H 参数表，可以方便的写出每一步的齐次变换矩阵，D-H 参数表的每一行对应一个齐次变换矩阵。从{0}系到{4}系的齐次变换矩阵记为${}_0^4\mathrm{T}$，它表示{4}号坐标系相对于{0}号坐标系的位姿，最终坐标变换矩阵如下。

$$ {}_0^4T = {}_0^1T\,{}_1^2T\,{}_2^3T\,{}_3^4T \tag{7.2} $$

因为每次机器人关节旋转都是某个单一轴在转动，所以很多夹角余弦值都是特殊值（0 或 1），这些值就是要求解的关节变量，求出关节变量，也就可以实现机械臂的控制了。

7.3.2　Dobot 机械臂正、逆运动学求解

机械臂运动学包括正向运动学和逆向运动学，正向运动学即给定机械臂各关节变量，计算机械臂末端的位置姿态； 逆向运动学即已知机械臂末端的位置姿态，计算机械臂对应位置的全部关节变量。一般正向运动学的解是唯一和容易获得的，而逆向运动学往往有多个解，并且分析更为复杂。逆运动学问题实际上是一个非线性超越方程组的求解问题，其中包括解的存在性、唯一性及求解的方法等一系列复杂问题。正、

逆运动学求解理论基础如下：

正运动学求解即已知各关节角度和连杆长度，求解机械臂末端执行器相对笛卡尔坐标系下的位姿。为了表示方便，令

$$\mathrm{C1}=\cos\theta 1,\ \mathrm{C2}=\cos\theta 2,\ \mathrm{C1}=\cos\theta 3 \tag{7.3}$$

$$\mathrm{S1}=\sin\theta 1,\ \mathrm{S2}=\sin\theta 2,\ \mathrm{S2}=\sin\theta 3,\ \mathrm{S23}=\sin(\theta 2+\theta 3),\ \mathrm{C23}=\cos(\theta 2+\theta 3) \tag{7.4}$$

由 D-H 转换法则，得各坐标系转换矩阵分别为

$$^{1}_{0}T=\mathrm{A1}=\begin{bmatrix} c1 & 0 & -s1 & 0 \\ s1 & 0 & c1 & 0 \\ 0 & -1 & 0 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.5}$$

$$^{2}_{1}T=\mathrm{A2}=\begin{bmatrix} c2 & -s2 & 0 & a2c2 \\ s2 & c2 & 0 & a2s2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.6}$$

$$^{3}_{2}T=\mathrm{A3}=\begin{bmatrix} c3 & -s3 & 0 & a3c3 \\ s3 & c3 & 0 & a3s3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.7}$$

$$^{4}_{3}T=\mathrm{A4}=\begin{bmatrix} c23 & -s23 & 0 & 0 \\ -s23 & c23 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.8}$$

机器人基座和末端执行器之间的总变换为

$$^{4}_{0}T=\mathrm{A1A2A3A4}=\begin{bmatrix} c1 & 0 & -s1 & c1(a3c23+\mathrm{a2c2}) \\ s1 & 0 & c1 & s1(a3c23+\mathrm{a2c2}) \\ 0 & -1 & 0 & d1-a2s2-a3s23 \\ 0 & 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.9}$$

变换矩阵 T 可以表述各个关节、机械臂末端在笛卡尔空间的姿态（n，o，a）和位置（px，py，pz），即完成了正运动学求解。

逆运动学求解即已知机器人末端执行器期望的位姿，求解机械臂的所有关节变量。令 Dobot 机械臂末端为

$$^{R}_{H}T=\begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix}=A1A2A3A4=^{4}_{0}T=\begin{bmatrix} c1 & 0 & -s1 & c1(a3c23+\mathrm{a2c2}) \\ s1 & 0 & c1 & s1(a3c23+\mathrm{a2c2}) \\ 0 & -1 & 0 & d1-a2s2-a3s23 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.10}$$

两边同时左乘A_1^{-1}，

$$\begin{bmatrix} nxc1+nys1 & oxc1+oys1 & axc1+ays1 & pxc1+pys1 \\ -nz & -oz & -az & d1-pz \\ nyc1-nxs1 & oyc1-oxs1 & ayc1-axs1 & pyc1-pxs1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a3c23+\mathrm{a2c2} \\ 0 & 1 & 0 & a3c23+\mathrm{a2S2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.11)$$

令两边矩阵元素（3，4）分别对应相等，得：

$$py*c1-px*s1=0,$$

进而求得: $\theta = tan^{-1}\frac{py}{px}$。

令式两边矩阵元素（1，4）和（2，4）分别对应相等，得:

$$py*c1+px*s1 = a2c2+a3c23$$

$$d1-pz = a2s2+a3s23$$

令上式方程两边平方并相加，整理得

$$\cos\theta 3 = \frac{(PxC1+\mathrm{PyS1})^2+(d1-Pz)^2-\mathrm{a2}^2-a3^2}{2a2a3} \quad (7.12)$$

$$\sin\theta 3 = \sqrt{1-\cos 2_{\theta 3}} \quad (7.13)$$

$$\theta 3 = \tan^{-1}\frac{\sin\theta 3}{\cos\theta 3} \quad (7.14)$$

整理得

$$\mathrm{PxC1+PyS1=(a2+a3c3)c2-a3s3s2} \quad (7.15)$$

$$\mathrm{d1-Pz=a3s3c2+(a2+a3c3)s2} \quad (7.16)$$

进一步求得

$$\cos\theta 2 = \frac{(\mathrm{d1-Pz})\mathrm{a3s3}+(\mathrm{PxC1+PyS1})(\mathrm{a3c3+c2})}{(a3c3+a2)^2+(a3s3)^2} \quad (7.17)$$

$$\theta 2 = tan^{-1}\frac{sin\,\theta 2}{cos\,\theta 2},\ \theta 4 = -(\theta 2+\theta 3) \quad (7.18)$$

由最终期望的位姿通过逆运动学求解出的各关节的运动角度分别为$\theta 1$、$\theta 2$、$\theta 3$和$\theta 4$。

但是由执行器期望位姿逆解得关节角的表达式可知，对于每一个末端位姿，同时对应的关节角度值不唯一。而控制机器人各关节的角度是唯一的，因此我们需要从中得出最优解。若忽略避障要求，可按以下步骤得到最优解。

- 首先利用直线插补算法得到笛卡尔空间的位置和姿态序列；
- 再调用各关节角的逆解公式，得到各关节角序列；
- 其次，通过调用动态规划算法，选出一组最优关节角序列；
- 最后，通过样条插值进行连续化处理。

综上所述，机械臂正运动学问题就是由各关节角度去求解机械臂末端在笛卡尔空间位姿表示，而逆运动学问题是通过某时刻机械臂末端的笛卡尔空间位姿去求解该时刻各关节的角度。

7.4 多关节的动作规划与控制

锁定机械臂其余三个关节的角度而调节单个关节的角度，虽然控制简单，但是工作效率低，而且增加了机械臂执行规定任务的时间，所以有必要让四个关节同时运动。定量分析耦合之后，通过附加补偿就可以实现四个关节同时运动。它们运动的合成效果，即末端运动轨迹，是一条比较复杂的曲线。

机械臂的连续轨迹运动（CP，continuous-path motion），不仅要指定运动的起始点和目标点，还需要指明运动所经过的若干中间点，最终机械臂会沿着这些点确定的路径运动。为解决障碍约束问题，可以通过对插值点的位姿、速度和加速度进行显式约束或给出运动路径的解析式来进行轨迹规划。轨迹规划既可以在笛卡尔参考坐标系中进行，也可以在关节参考坐标系中进行，但必须保证轨迹函数的连续和平滑，从而使机械臂运动平稳。

轨迹规划是在坐标系中定义一个关节或抓手的期望路径，分为笛卡尔空间规划和关节空间规划两种。在笛卡尔空间坐标系中，末端执行器位姿的轨迹非常直观，但是每个中间点都要通过逆运动学方程求解关节角度，因此计算量较大，而且还有可能达到机器人的奇异点。关节空间轨迹规划的优点是计算量不大，不会出现奇异位形。

综上所述，根据 D-H 模型方法对 Dobot 机械臂进行了关节坐标系建模，计算出其齐次变换矩，依据 Dobot 机械臂的搬运功能的特点，可以对 Dobot 机械臂提出给定作业轨迹的规划要求，为 Dobot 机械臂的研究和开发提供了理论依据。

为了深刻理解机械臂的轨迹规划，以及各部分的运动关系，将建立 3D 虚拟模型和真机上，规划机械臂的末端沿特定的轨迹运动的过程。在后续章中会重点介绍如何进行机械臂末端的轨迹规划。

7.5 机械臂动作规划与控制示例

实验以激光雕刻为例，介绍如何使用基本函数来进行激光雕刻图案。

实验目的：

（1）掌握 Dobot Magician 配套的激光雕刻套件的安装配置和使用方法。

（2）利用激光雕刻套件雕刻一些期望的图形。

实验方法：

调整好激光雕刻套件的功率和焦距后，用户就可以通过 Dobot Magician 的 SDK 提供的 CP 函数和延时等函数雕刻出自己想要的图形。根据程序所提供图形中间点全局坐标，机械臂依次将激光移动到相应点处，从而雕刻出完整图形。

实验原理：

激光雕刻原理比较简单，类似于用打印机在纸张上打印图案。常用的雕刻方式有点阵雕刻和矢量雕刻，

点阵雕刻利用机械臂控制激光头移动，雕刻出一条由一系列点组成的一条线，同时激光头上下移动雕刻出多条线，最后构成整版的图形或文字，而矢量雕刻则是在图形或文字的外轮廓上进行雕刻。这里涉及到的 API 函数如下：

（1）dType.GetCPParams(api)：获取连续轨迹功能参数

（2）dType.SetEndEffectorLaser(api, enableCtrl, on, isQueued=0)：设置激光开关

（3）dType.GetPose(api)：获取当前位姿

（4）dType.SetWAITCmd(api, waitTime, isQueued=0)：执行时间等待功能

（5）dType.SetCPLECmd(api, cpMode, x, y, z, power, isQueued=0)：执行 CP 运动指令

实验步骤：

（1）将激光雕刻套件安装到机械臂上并正确将机械臂与 PC 连接，具体步骤参考 2.3 节，如图 7.7 所示。

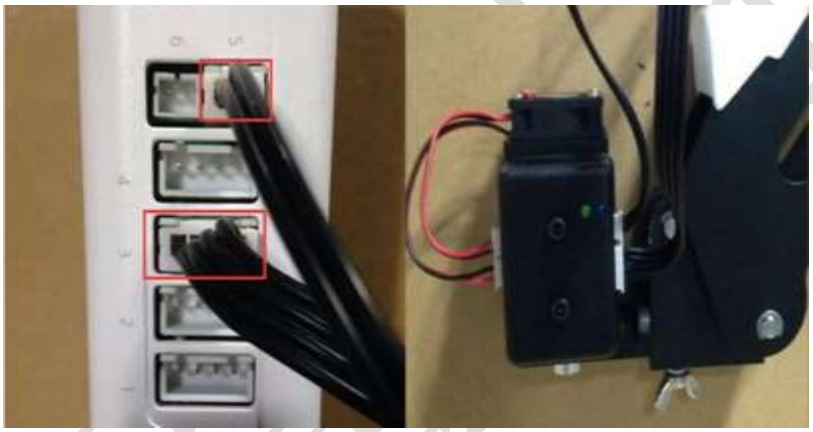

图 7.3 安装激光雕刻套件

（2）在 Dobot Studio 主界面选择末端执行器为“激光”，如图 7.8 所示。

图 7.4 选择激光为末端

（3）调整好激光的焦距以及激光焦距到雕刻材料表面的距离。

（4）进入 Dobot Studio 的“脚本控制界面”，新建脚本程序，输入下列代码：

```
import math
dType.SetCPParams(api, 50, 100, 50, 0, 1)   #设置激光雕刻的速度和加速度
dType.SetEndEffectorLaserEx(api, 1, 50, 1)    #打开激光，设置功率为 50
moveX=0;moveY=0;moveZ=10
pos = dType.GetPose(api)        #获取当前位置
r=20                            #圆的半径
x = pos[0]-r                    #圆心位置
y = pos[1]
z = pos[2]

for i in range(0,360,2):
    moveX = r*math.cos(i/180.0*math.pi)         #激光雕刻圆形时步长
    moveY = r*math.sin(i/180.0*math.pi)
    dType.SetCPLECmd(api, 1, x+moveX, y+moveY, z, 50, 1)    #雕刻圆形
    dType.SetWAITCmd(api, 100, 0)
    if i==72:                                   #五角星的尖端坐标
        x1=x+moveX
        y1=y+moveY
    if i==144:
        x2=x+moveX
        y2=y+moveY
    if i==216:
        x3=x+moveX
        y3=y+moveY
    if i==288:
        x4=x+moveX
        y4=y+moveY
dType.SetCPLECmd(api, 1, x2, y2, z, 50, 1)            #雕刻五角星
dType.SetCPLECmd(api, 1, x4, y4, z, 50, 1)
dType.SetCPLECmd(api, 1, x1, y1, z, 50, 1)
dType.SetCPLECmd(api, 1, x3, y3, z, 50, 1)
dType.SetCPLECmd(api, 1, pos[0], pos[1], z, 50, 1)
```

实验结果：

激光雕刻的最终效果是在雕刻材料上雕刻了一个内含五角星的圆，如图 7.9 所示。

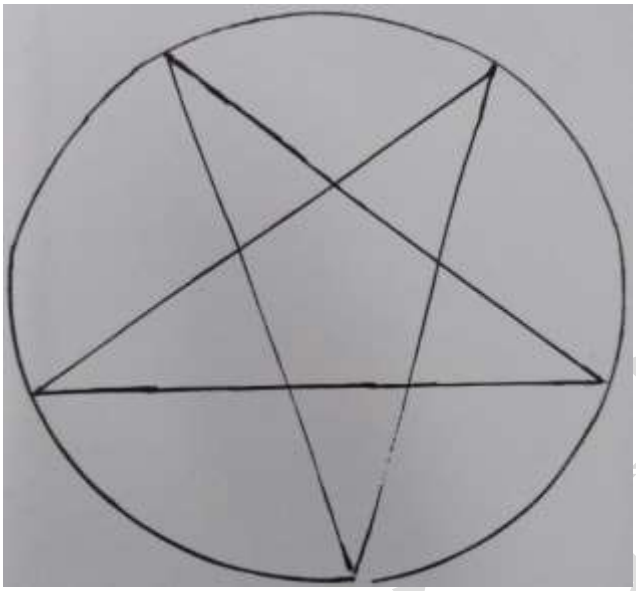

图 7.5　激光雕刻效果

越疆科技

8 机械臂的扩展

8.1 概述

8.1.1 扩展接口简介

Dobot Madician 机械臂在小臂上方和底座上集成了丰富的接口用来拓展，分为 VCC，GND 和 I/O 接口三类，其中 I/O 接口作为多功能外部扩展接口，又称为 EIO（Extended Input and Output），在 Dobot 系统中，EIO 使用统一编址，共有 20 个，编为 EIO1-EIO20。

8.1.2 EIO 接口分布

EIO 接口位于小臂以及底座上，具体分布及编址如图 8.1，图 8.2 及图 8.3 所示。

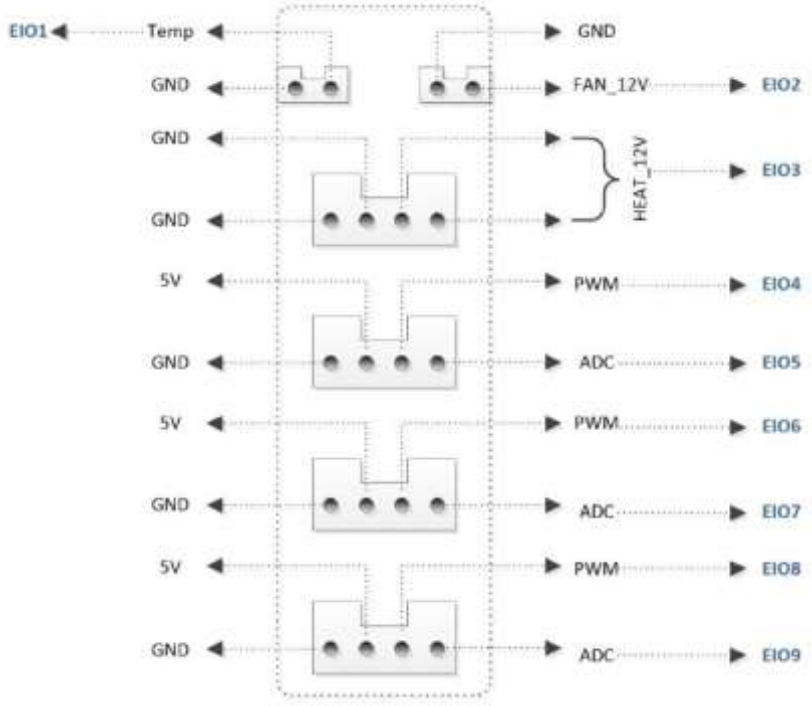

图 8.1 小臂 EIO 编址

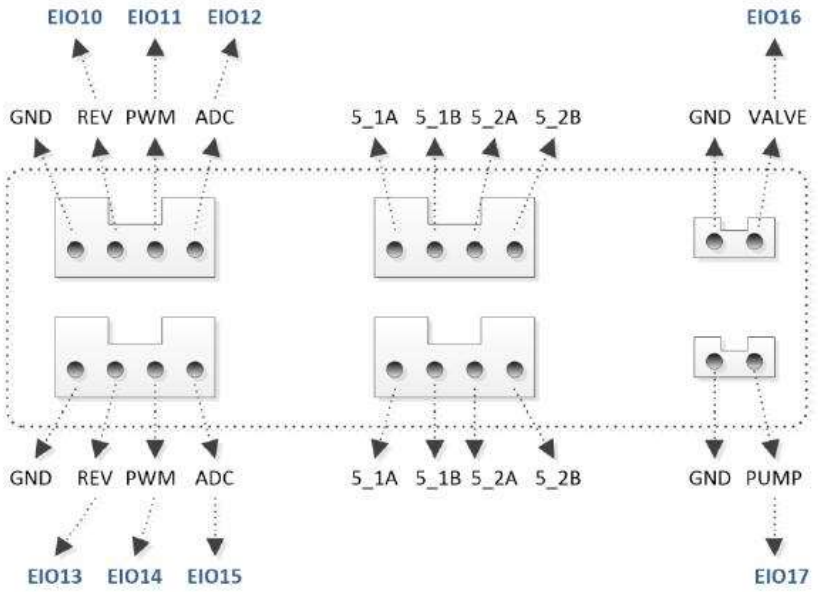

图 8.2　底座 18PinEIO 编址

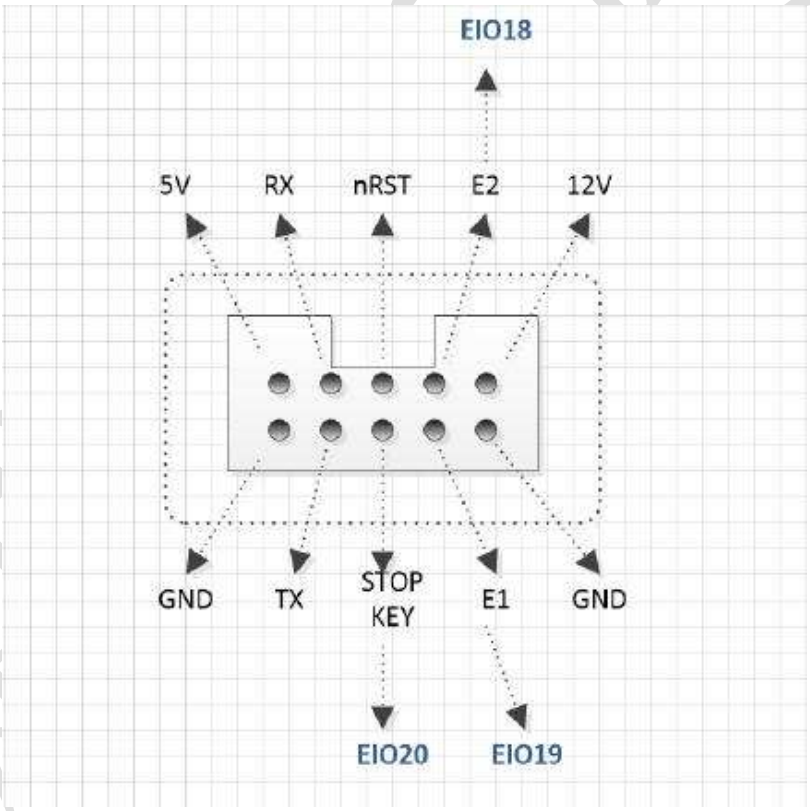

图 8.3　底座 10PinEIO 编址

8.1.3　EIO 功能

EIO 接口的功能有电平输出、PWM、电平输入和 ADC 功能，但是不同编址的 EIO 功能及适用的电平范围并不相同，使用的时候应当注意。

具体如表 8-1，表 8-2，表 8-3 所示。

表 8-1　小臂 EIO 功能表

EIO 编址	电平范围	电平输出	PWM	电平输入	ADC
O1	3.3V	✓		✓	✓
O2	12V	✓			
O3	12V	✓			
O4	3.3V	✓	✓	✓	
O5	3.3V	✓		✓	✓
O6	3.3V	✓	✓	✓	
O7	3.3V	✓		✓	✓
O8	3.3V	✓	✓	✓	
O9	3.3V	✓		✓	✓

表 8-2　底座 18PinEIO 功能表

EIO 编址	电平范围	电平输出	PWM	电平输入	ADC
10	5V	✓			
11	3.3V	✓	✓	✓	
12	3.3V	✓		✓	✓
13	5V	✓			
14	3.3V	✓	✓	✓	
15	3.3V	✓		✓	✓
16	12V	✓			
17	12V	✓			

表 8-3　底座 10PinEIO 功能表

EIO 编址	电平范围	电平输出	PWM	电平输入	ADC
18	3.3V	✓		✓	
19	3.3V	✓		✓	
20	3.3V	✓		✓	

8.2　扩展接口函数介绍

与队列控制方式函数定义相似，Dobot Studio 内部把扩展接口函数的声明和定义也放在 DobotDllType.py，将其作为模块导入，具体实现为“import DobotDllType as dType”，调用时需要模块名.方法名（），即 dType.SetIOMultiplexing (api, address, multiplex, isQueued=0)。下面具体介绍扩展接口函数。

（1）dType.SetIOMultiplexing (api, address, multiplex, isQueued=0)

功能：设置 I/O 复用

参数：address：EIO 地址

multiplex：EIO 配置类型

0：Dummy

1：PWM

2：OUTPUT

3：INPUT

4：AD

isQueued：队列模式使用状态

0：使用队列模式

1：不使用队列模式

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（2）dType.GetIOMultiplexing(api, addr)

功能：读取 I/O 复用

参数：address：EIO 地址

返回：list[0]: EIO 配置类型

0：当前地址 EIO 功能为 Dummy

1：当前地址 EIO 功能为 PWM

2：当前地址 EIO 功能为 OUTPUT

3：当前地址 EIO 功能为 INPUT

4：当前地址 EIO 功能为 AD

（3）dType.SetIODO(api, address, level, isQueued=0)

功能：设置 I/O 输出电平

参数：address：EIO 地址；

Level：电平

0：低电平

1：高电平

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（4）dType.GetIODO(api, addr)

功能：读取 I/O 输出电平

参数：address：EIO 地址

返回：list[0]：I/O 输出电平

0：当前地址 EIO 输出低电平

1：当前地址 EIO 输出高电平

（5）dType.SetIOPWM(api, address, frequency, dutyCycle, isQueued=0)

功能：设置 PWM 输出。

参数：address：EIO 地址

frequency：PWM 波频率

dutyCycle：PWM 波占空比

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（6）dType.GetIOPWM(api, addr)

功能：读取 PWM 输出

参数：addr：EIO 地址

返回：list[0]：PWM 波频率，PWM 波占空比

（7）dType.GetIODI(api, addr)

函数功能：读取 I/O 输入电平

参数：addr：EIO 地址

返回：list[0]：

0：当前地址 EIO 输入低电平；

1：当前地址 EIO 输入高电平。

（8）dType.SetEMotor(api, index, isEnabled, speed, isQueued=0)

功能：设置扩展电机速度接口

参数：IsEnabled：开关状态

0：关闭

1：开启

speed：转速

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

（9）dType.SetEMotorS(api, index, isEnabled, deltaPulse, isQueued=0)

功能：设置扩展电机移动距离接口

参数：IsEnabled：开关状态

0：关闭

1：开启

deltaPulse：电机移动的距离

返回：list[0]：两种可能结果

越疆科技

队列模式：队列命令索引

立即模式：0

（10）dType.GetIOADC(api, addr)

功能：读取 I/O 模数转换值

参数：addr：EIO 地址

返回：list[0]：两种可能结果

队列模式：队列命令索引

立即模式：0

8.3 扩展导轨的控制示例

实验以让机械臂写毛笔字为例，介绍了基本函数的使用。

实验目的：

以机器人书写运动为例，熟悉部分机械臂的扩展导轨部分的使用

实验方法及原理：

使用 DobotStudio 中的“脚本控制”，执行自定义的 Python 程序，利用基本函数实现用机械臂写毛笔字。这里涉及到的 API 函数如下：

（1）dType.SetPTPJointParams(api, j1Velocity, j1Acceleration, j2Velocity, j2Acceleration, j3Velocity, j3Acceleration, j4Velocity, j4Acceleration, isQueued=0)：设置关节点位参数

（2）dType.SetPTPCoordinateParams(api, xyzVelocity, xyzAcceleration, rVelocity, rAcceleration, isQueued=0)：设置协同运动参数

（3）dType.GetPTPJumpParams(api)：设置门型运动参数

（4）dType.SetPTPCommonParams(api, velocityRatio, accelerationRatio, isQueued=0)：设置点位共用参数

（5）dType.SetPTPWithLCmd(api, ptpMode, x, y, z, rHead, l, isQueued=0)：执行滑轨点动

（6）dType.SetDeviceWithL(api, isWithL)：控制滑轨开关

实验步骤：

（1）将笔套件安装到机械臂上并正确将机械臂与 PC 连接，具体步骤参考 2.3 节，结果如图 8.4 所示；

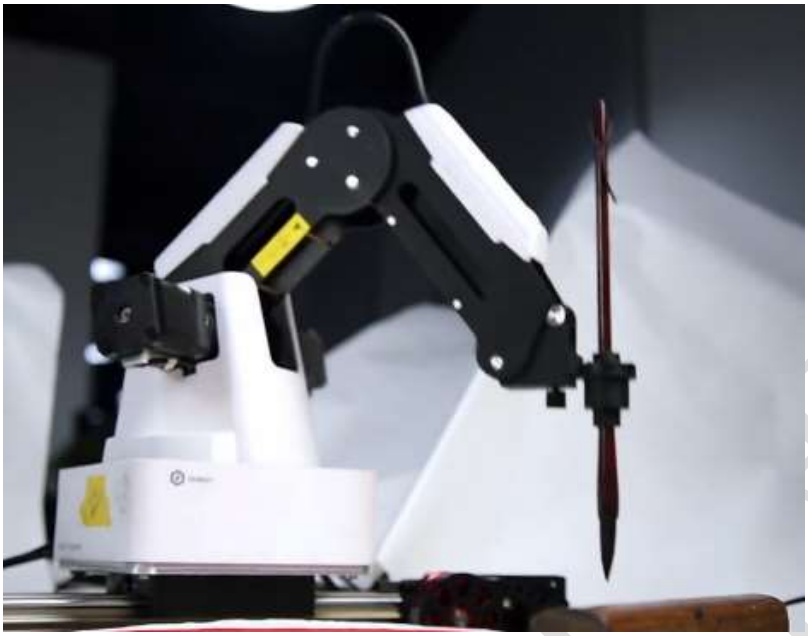

图 8.4 毛笔安装效果图

（2）在 Dobot Studio 中选择末端执行器为“笔”；

（3）进入 Dobot Studio 的“脚本控制界面”，新建如下扩展导轨的控制示例程序：

```
import math
dType.SetPTPJointParams(api,200,200,200,200,200,200,200,200)#设置关节运动参数
dType.SetPTPCoordinateParams(api,200,200,200,200)#设置 PTP 协同运动参数
dType.SetPTPJumpParams(api, 10, 200)#设置门型运动参数
dType.SetPTPCommonParams(api, 100, 100)#设置 PTP 共用参数

dType.SetDeviceWithL(api, 1)#设置导轨
dType.SetPTPLParams(api, 200, 200)#设置导轨运动参数

lx=10#文字尺寸（mm）
pos = dType.GetPose(api)#获取当前位置
posL= dType.GetPoseL(api)#获取导轨位置
#设置使用的三个 X 值
x0= pos[0]
x1= x0-lx
x2= x0-2*lx

#设置使用的一个 Y 值
y0= pos[1]
```

```
#设置使用的两个 Z 值
z0= pos[2]
z1= z0+3*lx

L0= posL[0]
rHead = pos[3]

x = x0
y = y0
z = z0
L = L0

#下文中将字母的点称为点*.*，例如第一个字母的第二个点为点 1.2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#直线运动到点 1.1，下同
x = x2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#1.2
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#1.2.5

x = x0
L +=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#1.3
x = x2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#1.4
z=z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#1.4.5

x = x1
L -=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#1.5
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#1.6
```

```
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#1.6.5

x = x0
L +=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.1
x = x2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.2
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.3
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#2.3.5

x = x0
L -=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.4
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.5
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#2.5.5

x = x1
L -=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.6
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#2.7
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#2.7.5
x = x0
L +=lx
z = z0
```

```
 dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#3.1
x = x2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#3.2
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#3.3
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#3.3.5

x = x0
L +=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#4.1
x = x2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#4.2
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#4.3
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z1, rHead, L)#4.3.5

x = x0
L +=lx
z = z0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#5.1
x = x2
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#5.2
L +=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#5.3
x = x0
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#5.4
L -=lx
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#5.5
z = z1
dType.SetPTPWithLCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, L)#end
```

（4） 调整笔尖位置，按住小臂上的圆形按钮 Unlock key 拖动小臂将笔尖调整至写字纸平面上。

（5）点击“开始”，编译下载程序到机械臂中。

实验结果：

实验效果如图 8.5 所示。

图 8.5　实验结果

8.4　远程控制示例

8.4.1　蓝牙功能

（1）关机状态下将 Wireless-1 蓝牙模块连接 Dobot Magician 机械臂底座 10Pin 扩展接口；

（2）开启主电源，三声响声后表示初始化完毕，此时蓝牙模块的蓝灯常亮，绿灯闪烁，如图 8.6 所示；

图 8.6　蓝牙模块连接效果图

（3）在手机（IOS）设置里打开蓝牙，再打开 Dobot App 即可连接上机械臂。

8.4.2　WiFi 功能

（1）关机状态下将 Wireless-2WiFi 模块连接 DobotMagician 机械臂底座 10Pin 扩展接口；

（2）首次使用前，需要对 WiFi 模块的设置，完成后即可实现 WiFi 模块的独立使用，不再需要连接 USB 线；

（3）打开 DobotStudio，点击“连接”按钮通过 COM 口连接机械臂，选择“设置”选项，开启 Wi-Fi 参数设置界面，如图 8.6 所示；

图 8.6　Wi-Fi 参数设置界面

（4）输入要接入的无线局域网相关参数，在“SSID”输入框填入 Wi-Fi 名字，在“Password”输入框填入 Wi-Fi 密码，使用 DHCP 功能，请勾选单选框（不使用 DHCP 功能，填写“IPAddress”、“Netmask”、“Gateway”、“DNS”等参数），一般情况下，直接使用 DHCP 功能并点击确定 OK 按钮完成设置，如图 8.7 所示；

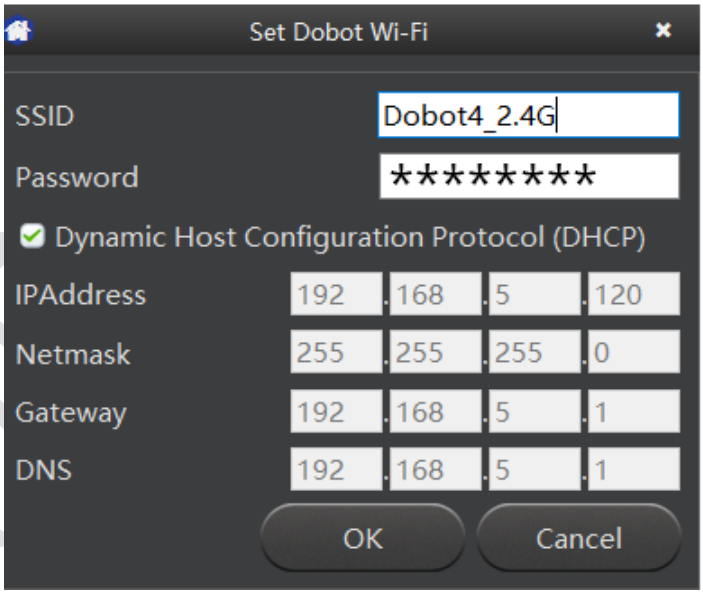

图 8.7　Wi-Fi 具体参数设置

（5）等待若干秒，Dobot Wi- Fi 无线模块的绿色信号灯常亮，表示 Dobot 已经接入局域网，如图 8.8 所示；

图 8.8　Wi-Fi 模块信号灯指示

（5）此时在 Dobot Studio 主界面可以看到自动搜索出来的设备地址，如图 8.9 和图 8.10 所示，选择该地址，点击连接，成功后即可通过 WiFi 控制机械臂。并且后续插入 WiFi 模块即可通过该地址连接到机械臂，无需使用 USB 线。

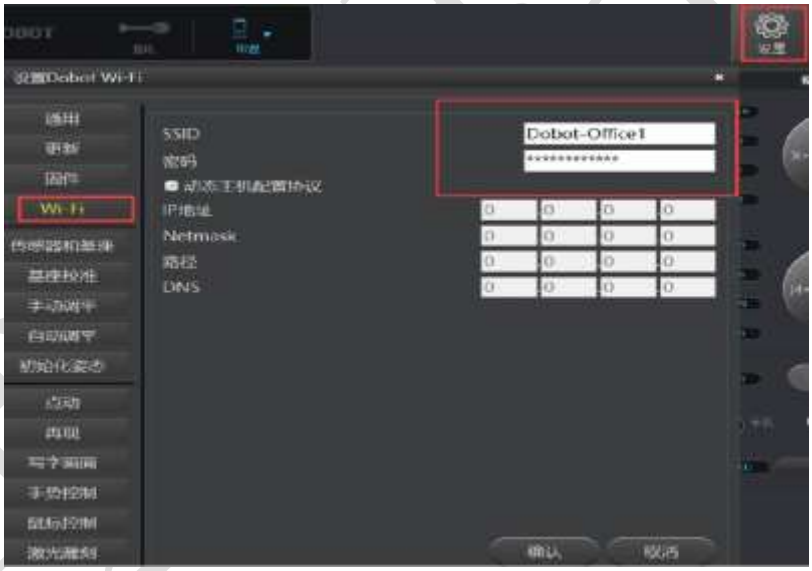

图 8.9 连接 WiFi

图 8.10 选择 WiFi 模式连接

9 基于机器视觉的机械臂控制

本章以分拣功能的实现为例介绍机械臂与视觉处理系统的集成应用。

9.1 视觉部件的安装与连接

本例程选用的是深圳市越疆科技公司生产的机器视觉运动控制实验平台，该视觉套件主要包括可调相机安装模块、可调节光源安装模块、固定底板、延长杆及支座、工业相机组件（带 USB）、四色积木块一包以及黑白格标定板，视觉套件清单如图 9.1 所示。

图 9.1 视觉套件组件示意图

视觉硬件参数如下表 9-1、表 9-2 和表 9-3 所示。

表 9-1 工业相机参数规格

高清彩色工业相机	
产品编号	JHSM300f
传感器尺寸	1/2‘CMOS
有效像素	300 万
色彩	彩色
帧率/分辨率	12@2048x1536
信噪比	42dB
快门类型	卷帘曝光
曝光控制	自动/手动
外壳尺寸	40x45x45mm
数据接口	USB2.0
工作温度	0~70℃
镜头接口	C 接口

表 9-2 16mm 定焦镜头参数规格

16mm 定焦镜头					
型号	M1614-MP2				
焦距	16mm				
工作范围	光圈	F1.4-F16C			
	焦点	0.3m-Inf			
控制	光圈	手动			
	焦点	手动			
分辨率	在中心处和边缘处超过 100 线对/mm				
后焦距	13.1mm				
工作距离	300mm	250mm	200mm	150mm	100mm
光学放大率	0.05X	0.06X	0.08X	0.10X	0.15X

表 9-3 光源参数说明

参数	说明
光源型号	JHZM
照度	40000Lux
亮度	连续可调式。调节范围：0%~100%
输出电压	12V
输出功率	3.5~5W
工作距离	35~110mm

视觉套件安装分两部分，分别为相机的安装和机械臂的安装，具体安装教程如下：

（1）安装相机；

1）将摄像头法兰支座用螺丝固定在延长杆底部，如图 9.2 所示。

图 9.2 安装摄像头法兰支座

2）将两个延长杆连接在一起，如图 9.3 所示。

图 9.3 连接延长杆

3）利用四颗 M5*12 的内六角螺丝将延长杆带法兰支座的一头固定在摄像头底座上，如图 9.4 所示。

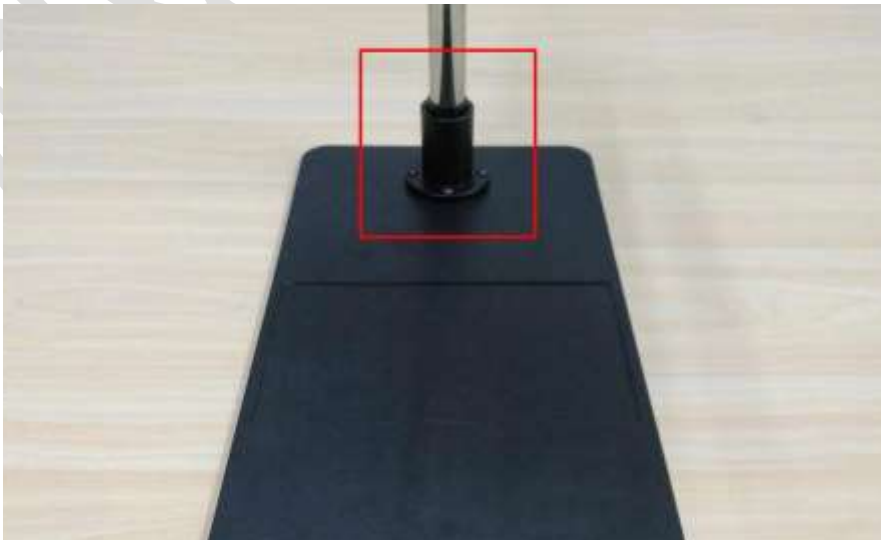

图 9.4 固定延长杆

4）安装相机支架，，安装样式如图 9.5 所示。

图 9.5 安装相机支架

5）将工业相机固定在相机夹具中间，调整好间距，如图 9.6 所示。

图 9.6 固定相机

注 9.1：固定相机时将相机 USB 接口朝向相机夹具外侧，如图 9.7 所示，以免呈现的图像与实际图像相反。

图 9.7 相机 USB 接口位置

6）将相机支架套在延长杆上，调节相机支架的固定夹具，使相机支架固定在延长杆上，如图 9.8 所示。

注 9.2：请根据实际需求调整相机支架的高度，相机支架不能阻挡机械臂运动。

图 9.8 固定相机支架

7）将 USB 线一头接入相机，另一头接入电脑，如图 9.9 所示。

图 9.9 连接 USB

8）将相机光源套件套在相机上，拧紧光源套件的三个固定旋钮，使其固定在相机上，如图 9.10 所示。

图 9.10 固定光源套件

9）连接光源套件电源线，如图 9.11 所示。

图 9.11 连接光源套件电源线

（2）安装 Dobot Magician 机械臂

1）在工作台上安装 Dobot Magician 机械臂，末端执行器为吸盘，具体步骤参考 1.3 节。

2）调整相机支架，确保相机能监测到 Dobot Magician 的运动范围，最终安装好的视觉套件如图 9.12 所示。

图 9.12 安装 Dobot Magician

9.2 相机标定及图像处理

在图像测量过程以及机器视觉应用中，为确定空间物体表面某点的三维几何位置与其在图像中对应点之间的相互关系，必须建立相机成像的几何模型，这些几何模型参数就是相机参数。在大多数条件下这些参数必须通过实验与计算才能得到，这个求解参数的过程就称之为相机标定。

无论是在图像测量或者机器视觉应用中，相机参数的标定都是非常关键的环节，其标定结果的精度及算法的稳定性直接影响相机工作产生结果的准确性。因此，做好相机标定是做好后续工作的前提，提高标定精度是科研工作的重点所在。在视觉系统中，获得一张高质量可处理的图像是至关重要，而图像处理就是对图像进行分析、加工、和处理，使其满足视觉、心理以及其他要求的技术。

9.2.1 相机标定

本教程的相机标定将把视觉套件和机械臂结合使用，根据标定板上三点图像坐标和机械臂对应位置的笛卡尔坐标获取变换系数。假设标定板上三点图像坐标$A=\begin{bmatrix} x1 & x2 & x3 \\ y1 & y2 & y3 \\ 1 & 1 & 1 \end{bmatrix}$，机械臂对应位置的三点笛卡尔坐标$B=\begin{bmatrix} x1' & x2' & x3' \\ y1' & y2' & y3' \\ 1 & 1 & 1 \end{bmatrix}$，变换矩阵为 R^T ，则 $B=R^TA$。

相机标定工作流程如下如图 9.13 所示。

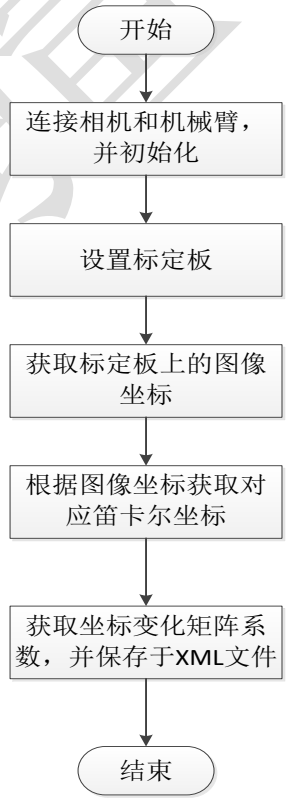

图 9.13 相机标定流程

相机标定涉及的基本函数如下：

（1）连接相机和机械臂并初始化

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    isCalibrate(false)
{
    ui->setupUi(this);
    connectCam();
    initCamParams();
    connectDobot();
    initDobotParams();

    connect(ui->imgBtn, SIGNAL(clicked(bool)), this, SLOT(onImgBtnClicked()));
    connect(ui->redPointBtn, SIGNAL(clicked(bool)), this, SLOT(onRedPointBtnClicked()));
    connect(ui->greenPointBtn, SIGNAL(clicked(bool)), this,SLOT(onGreenPointBtnClicked()));
    connect(ui->bluePointBtn, SIGNAL(clicked(bool)), this, SLOT(onBluePointBtnClicked()));
    connect(ui->homeBtn, SIGNAL(clicked(bool)), this, SLOT(onHomeBtnClicked()));
    connect(ui->okBtn, SIGNAL(clicked(bool)), this, SLOT(onOkBtnClicked()));
}
```

（2）设置标定板的宽度，高度和单个方格的长度，需设置为标定板横向内角点个数和纵向内角点个数。本例程使用的标定板宽度为 7，高度为 6，标定板单个方格为 15mm。

```
void MainWindow::onImgBtnClicked()
{
m_thread->pause();
isCalibrate = true;
SetBoardSize(7, 6, 15);//设置标定板参数
CameraSetSnapMode(m_index, CAMERA_SNAP_TRIGGER);
m_thread->stream();
}
```

（3）获取投影在标定板上的红绿蓝三点图像坐标，并窗口中显示。

```
void MainWindow::process(QImage img, unsigned char *buffer)
{
    if (isCalibrate == false) {
```

```
        qDebug() << "Camera testing...";
        QImage imresize = img.scaled(ui->imgLabel->width(), ui->imgLabel->height());
        ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
    }else {
        qDebug() << "enter Calibration";
        int width, height, len;
        CameraGetImageSize(0, &width, &height);
        CameraGetImageBufferSize(0, &len, CAMERA_IMAGE_RGB24);

        mmat.pBuffer = buffer;
        mmat.numChannels = 3;
        mmat.width = width;
        mmat.height = height;
        InputSourceImage(&mmat);
        Tracking(); //find image points(red,green,blue)
        OutPutImage(&mmat);
        //获取图像上的红，绿，蓝坐标点
        MvPoint mvPoint1;
        GetImg2DPoints(0, &mvPoint1);
        MvPoint mvPoint2;
        GetImg2DPoints(1, &mvPoint2);
        MvPoint mvPoint3;
        GetImg2DPoints(2, &mvPoint3);
        //在文本编辑栏中显示图像坐标点
        ui->Edit_i1x->setText(QString::number(mvPoint1.x, 'f', 3));
        ui->Edit_i1y->setText(QString::number(mvPoint1.y, 'f', 3));
        ui->Edit_i1z->setText(QString::number(1));

        ui->Edit_i2x->setText(QString::number(mvPoint2.x, 'f', 3));
        ui->Edit_i2y->setText(QString::number(mvPoint2.y, 'f', 3));
        ui->Edit_i2z->setText(QString::number(1));
        ui->Edit_i3x->setText(QString::number(mvPoint3.x, 'f', 3));
        ui->Edit_i3y->setText(QString::number(mvPoint3.y, 'f', 3));
        ui->Edit_i3z->setText(QString::number(1));
```

```
            //显示物体图像
            QImage imshow = Pk24bitColorToQIm(mmat.pBuffer, mmat.width, mmat.height);
            QImage imresize = imshow.scaled(ui->imgLabel->width(), ui->imgLabel->height());
            ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
        }
        delete buffer;
        return;
}
```

（4）根据投影在标定板上的红绿蓝三点，获取机械臂对应位置的笛卡尔坐标，以红点为例。

```
void MainWindow::onRedPointBtnClicked()
{
    Pose pose;
    if (GetPose(&pose) == DobotCommunicate_NoError)
    {
        ui->Edit_w1x->setText(QString::number(pose.x, 'f', 3));
        ui->Edit_w1y->setText(QString::number(pose.y, 'f', 3));
        ui->Edit_w1z->setText("1");
        Set3DPoints(pose.x, pose.y, 0);
    }
}
```

（5）根据 $B=R^TA$ 计算坐标变换矩阵系数，并保存至“CalibrationParams.xml”文件中。

```
void MainWindow::onOkBtnClicked()
{
    Calibration();//相机标定
    QString fileName = QString("%1/CalibrationParams.xml").arg(qApp->applicationDirPath());
    QFile file(fileName);
    file.open(QIODevice::WriteOnly);
    QXmlStreamWriter xmlWriter(&file);
    xmlWriter.setAutoFormatting(true);
    xmlWriter.writeStartDocument();
    xmlWriter.writeStartElement("CalibrationParams");

    float RT00,RT01,RT02,RT10,RT11,RT12,RT20,RT21,RT22;//定义变换矩阵个元素
    GetRtValue(0, 0, &RT00);
```

```
    GetRtValue(0, 1, &RT01);
    GetRtValue(0, 2, &RT02);
    GetRtValue(1, 0, &RT10);
    GetRtValue(1, 1, &RT11);
    GetRtValue(1, 2, &RT12);
    GetRtValue(2, 0, &RT20);
    GetRtValue(2, 1, &RT21);
    GetRtValue(2, 2, &RT22);
    //保存至“CalibrationParams.xml”文件
    xmlWriter.writeTextElement("RT00", QString::number(RT00, 'f', 3));
    xmlWriter.writeTextElement("RT01", QString::number(RT01, 'f', 3));
    xmlWriter.writeTextElement("RT02", QString::number(RT02, 'f', 3));

    xmlWriter.writeTextElement("RT10", QString::number(RT10, 'f', 3));
    xmlWriter.writeTextElement("RT11", QString::number(RT11, 'f', 3));
    xmlWriter.writeTextElement("RT12", QString::number(RT12, 'f', 3));

    xmlWriter.writeTextElement("RT20", QString::number(RT20, 'f', 3));
    xmlWriter.writeTextElement("RT21", QString::number(RT21, 'f', 3));
    xmlWriter.writeTextElement("RT22", QString::number(RT22, 'f', 3));

    xmlWriter.writeEndElement();
    xmlWriter.writeEndDocument();
    file.close();
    QMessageBox::information(this, "Result", "Calibrating dobot and camera success! \n"
                                             "You can get a CalibraionParams.xml file in your folder,"
                                             "which save a coefficient for using.");
}
```

9.2.2 颜色特征提取

下面介绍颜色特征提取中设计图像处理中的一些基本概念：

（1）像素

像素是组成图象的最基本单元要素，分辨率是指在长和宽的两个方向上各拥有的像素个数。如一张640X480 的图片，表示这张图片在每一个长度的方向上都有 640 个像素点，每一个宽度方向上都有 480 个

像素点，总数就是 640×480=307200（个像素），简称 30 万像素。

（2）RGB 颜色空间

在计算机技术中使用最广泛的颜色空间是 RGB 颜色空间，它是一种与人的视觉系统结构密切相关的颜色模型。根据人眼睛的结构，所有的颜色都可以看成由三个基本颜色：红色（red）、绿色（green）和蓝色（blue）的组合而成，而且大部分显示器都采用这种颜色模型。

在 RGB 颜色模型中，黑色在原点处，白色位于离原点最远的角上，灰度级沿着这两点的连线分布，每一个分量图像都是其原色图像。在彩色图像/RGB 图像中，图像是一个三维矩阵，如 400×300×3，其中 400 表示列数，300 表述行数，3 代表三个分量，也就是 R,G,B。

（3）HSV 颜色空间

HSV 颜色空间是图像处理中另外一个常用的颜色空间，它从人的视觉系统出发，由色调（Hue）、饱和度（Saturation）、亮度（Value）三个分量构成，HSV 更接近于人眼的主观感受。其中，色调 H 由角度表示，饱和度 S 是 HIS 彩色空间中轴线到彩色点的半径长度，彩色点离轴线的距离越近，表示颜色的白光越多。亮度 V 用轴线方向上的高度表示。圆锥体的轴线描述了灰度级，强度最小值时为黑色，强度最大值时为白色。HSV 颜色空间和 RGB 颜色空间只是同一物理量的不同表示方法。

（4）灰度颜色空间

灰度是通过把白色与黑色之间按对数关系分为若干等级，灰度范围为 0-255，因此灰度图像每个像素需要一个字节存放。灰度值只是表征单色的亮暗程度，灰度色彩空间对中间处理特别有效。

（5）灰度图与 RGB 彩色图像之间的关系

将彩色图像转化成为灰度图像的过程称为图像的灰度化处理。彩色图像中的每个像素的颜色由 R、G、B 三个分量决定，且每个分量有 255 种值可取，这样一个像素点可以有 1600 多万（255*255*255）的颜色的变化范围。而灰度图像是 R、G、B 三个分量相同的一种特殊的彩色图像，其一个像素点的变化范围为 255 种，所以在数字图像处理中一般先将各种格式的图像转变成灰度图像以便减少后续图像的计算量。

（6）RGB 到灰度的映射公式

1）对于彩色转灰度，有一个很著名的心理学公式：

$$\mathrm{Gray} = \mathrm{R} \times 0.299 + \mathrm{G} \times 0.587 + \mathrm{B} \times 0.114$$

2）整数算法，将上式缩放 1000 倍来实现整数运算算法：

$$\mathrm{Gray} = (\mathrm{R} * 299 + \mathrm{G} * 587 + \mathrm{B} * 114 + 500) / 1000$$

3）另一种是 Adobe Photoshop 里的公式 Adobe RGB (1998) [gamma=2.20]：

$$\mathrm{Gray} = (\mathrm{R}^{2.2} * 0.2973 + \mathrm{G}^{2.2} * 0.6274 + \mathrm{B}^{2.2} * 0.0753)^{\frac{1}{2.2}}$$

该方法运行速度稍慢，但是效果很好。

4）平均值方法

$$\mathrm{GRAY} = \frac{\mathrm{RED} + \mathrm{BLUE} + \mathrm{GREEN}}{3}$$

用（GRAY,GRAY,GRAY ） 替代 （RED,GREEN,BLUE），但是这样做的精度比较低，图像转化为灰度效果不是太好。

（7）二值化：

图像的二值化处理就是将图像像素点的灰度值置为 0 或 255，使整个图像呈现出明显的黑白效果。即将 256 个亮度等级的灰度图像通过选取适当的阀值，从而获得反映图像整体和局部特征的二值化图像。

在数字图像处理中，二值图像占有非常重要的地位，但是在二值化前需要先将图像“灰度化”，然后才能二值化，将所有灰度大于或等于阀值的像素被判定为属于特定物体，其灰度值为 255 表示，否则这些像素点被排除在物体区域以外，灰度值为 0，表示背景或者例外的物体区域。当再对图像做进一步处理时，图像的集合性质只与像素值为 0 或 255 的点的位置有关，使处理变得简单。

（8）安装 opencv

opencv 是一个跨平台计算机视觉库，点击下载好的 opencv-2.4.12.exe，一路 next 下去即可。

下面介绍一些 opencv 的算子。

（1）图像处理

1）cv2.cvtColor(input_image，flag)

功能：对图像进行颜色空间转换，比如从 BGR 到 Gray，或者从 BGR 到 HSV 等

参数：input_image：输入图像；flag：转换类型。

对于 BGR↔Gray 的转换，使用的 flag 就是 cv2.COLOR_BGR2GRAY。同样对于 BGR↔HSV 的转换，flag 就是 cv2.COLOR_BGR2HSV

2) cv2.inRange( )

功能：提取特定颜色的区域

参数：第一个参数：hsv 指的是原图；第二个参数：lower_red 指的是图像中低于这个 lower_red 的值，图像值变为 255；第三个参数：upper_red 指的是图像中高于这个 upper_red 的值，图像值变为 255；而在 lower_red～upper_red 之间的值变成 0。

3）cv2.medianBlur( )

功能：中值滤波，目的是降低图像的变化率，消除噪声。

参数：第一个参数是待处理图像，第二个参数是孔径的尺寸，为大于 1 的奇数。

（2）形态学转换

形态学操作是根据图像形状进行的简单操作。一般情况下对二值化图像进行的操作。需要输入两个参数，一个是原始图像，第二个被称为结构化元素或核，它是用来决定操作的性质的。两个基本的形态学操作是腐蚀和膨胀。他们的变体构成了开运算，闭运算，梯度等。

1）cv2.getStructuringElement( )

功能：设置图像结构元素

参数：第一个参数表示内核的形状，有三种形状可以选择。矩形：MORPH_RECT；交叉形：MORPH_CORSS；椭圆形：MORPH_ELLIPSE；第二和第三个参数分别是内核的尺寸以及锚点的位置。一般在调用 erode 以及 dilate 函数之前，先定义一个 Mat 类型的变量来获得。

返回：锚点的位置

2）cv2.erode()

功能：它提取的是内核覆盖下的相素最小值。进行腐蚀操作时，将内核 B 划过图像,将内核 B 覆盖区域的最小相素值提取，并代替锚点位置的相素。

参数：第一个参数 src：输入图，可以多通道，深度可为 CV_8U、CV_16U、CV_16S、CV_32F 或 CV_64F。第二个参数 dst：输出图，和输入图尺寸、型态相同。第三个参数 kernel：结构元素，如果 kernel=Mat()则为预设的 3×3 矩形，越大侵蚀效果越明显。第四个参数 anchor：原点位置，预设为结构元素的中央。第五个参数 iterations：执行次数，执行行越多次侵蚀效果越明显。

返回：第一个为 最优阈值 retVal，第二个就是阈值化之后的结果图

3）cv2.dilate()

功能：此操作将图像 A 与任意形状的内核 (B)，通常为正方形或圆形,进行卷积。内核 B 有一个可定义的锚点, 通常定义为内核中心点。进行膨胀操作时，将内核 B 划过图像,将内核 B 覆盖区域的最大相素值提取，并代替锚点位置的相素。显然，这一最大化操作将会导致图像中的亮区开始"扩展" (因此有了术语膨胀 dilation)。

参数：第一个参数 src：输入图，可以多通道，深度可为 CV_8U、CV_16U、CV_16S、CV_32F 或 CV_64F。第二个参数 dst：输出图，和输入图尺寸、型态相同。第三个参数 kernel：结构元素，如果 kernel=Mat()则为预设的 3×3 矩形，越大侵蚀效果越明显。第四个参数 anchor：原点位置，预设为结构元素的中央。第五个参数 iterations：执行次数，执行行越多次侵蚀效果越明显。

返回：第一个为 最优阈值 retVal，第二个就是阈值化之后的结果图像。

（3）Canny 边缘检测

Canny 边缘检测是一种非常流行的边缘检测算法，由于边缘检测很容易受到噪声影响，所以第一步是使用高斯滤波器；第二步对平滑后的图像使用 Sobel 算子计算水平方向和竖直方向的一阶导数（图像梯度）（Gx 和 Gy）。根据得到的这两幅梯度图（Gx 和 Gy）找到边界的梯度和方向；第三步非极大值抑制，排除非边缘像素， 仅仅保留了一些细线条；第四步使用滞后阈值，确定真正的边界。

1）cv2.Canny()

功能：标识数字图像中亮度变化明显的点。

参数：第一个参数是输入图像。第二和第三个分别是 minVal 和 maxVal。第三个参数设置用来计算图像梯度的 Sobel 卷积核的大小，默认值为 3。最后一个参数是 L2gradient，它可以用来设定求梯度大小的方程。

（5）轮廓提取

轮廓可以简单认为成将连续的点（连着边界）连在一起的曲线，具有相同的颜色或者灰度。轮廓在形状分析和物体的检测和识别中很有用。为了更加准确，要使用二值化图像。在寻找轮廓之前，要进行阈值化处理或者 Canny 边界检测。在 OpenCV 中，查找轮廓就像在黑色背景中找白色物体，要找的物体应该是白色而背景应该是黑色。

1）cv2.findContours()

功能：在二值图像中查找轮廓

参数：第一个参数是输入图像，第二个参数是轮廓检索模式，第三个参数是轮廓近似方法。

返回：第一个返回值是图像，第二个返回值是轮廓，为一个 Python 列表，其中存储这图像中的所有轮廓。每一个轮廓都是一个 Numpy 数组，包含对象边界点（x，y）的坐标。第三个是轮廓的层析结构。

本教程通过设置 HSV 颜色空间的参数来获取物块的 HSV 空间特征进行视觉识别。颜色特征提取工作流程如下如图 9.14 所示。

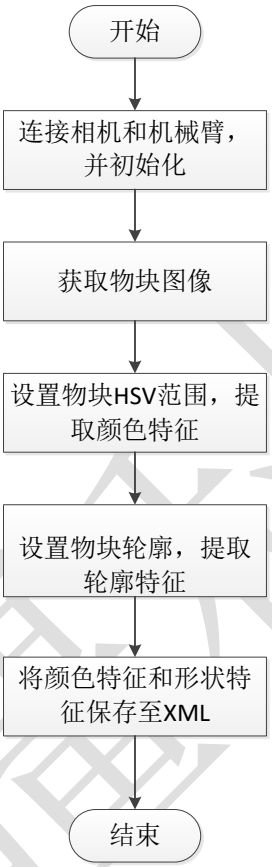

图 9.14 颜色特征提取实现流程图

颜色特征提取涉及的基本函数如下：

（1）获取物块影像

```
void MainWindow::onImgBtnClicked()
{
    m_thread->pause();
    CameraSetSnapMode(m_index, CAMERA_SNAP_TRIGGER);
    m_thread->stream();
    CameraTriggerShot(m_index);

    static bool isFirstCheck = true;
    if (isFirstCheck) {
```

```
            isFirstCheck = false;
            setSliderEnable(true);
            ui->redCheckBox->setChecked(true);
        }
    }
    void MainWindow::process(QImage img, unsigned char *buffer)
    {
        qDebug() << "Camera testing...";
        int width, height, len;
        CameraGetImageBufferSize(0, &len, CAMERA_IMAGE_RGB24);
        CameraGetImageSize(0, &width, &height);
        mmat.pBuffer = buffer;
        mmat.numChannels = 3;
        mmat.width = width;
        mmat.height = height;
        InputSourceImage(&mmat);
        OutPutImage(&mmat);
        QImage imresize = img.scaled(ui->imgLabel->width(), ui->imgLabel->height());
        ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
        delete buffer;
        return;
    }
```

（2）设置物块的 HSV 范围，提取颜色特征

```
void MainWindow::hsvpro(void)//Used to deal with changes in HSV values
{
    qDebug() << Q_FUNC_INFO;
    ColorDetect(gHSV.lowH, gHSV.lowS, gHSV.lowV,
                gHSV.highH, gHSV.highS, gHSV.highV,
                BLUR_SIZE);
    GetGrayImage(&mmat);
    //Show image
    QImage imshow = Pk8bitGrayToQIm(mmat.pBuffer, mmat.width, mmat.height);
    QImage imresize = imshow.scaled(ui->imgLabel->width(), ui->imgLabel->height());
    ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
}
```

（3）设置物块的轮廓，提取轮廓特征。

```
void MainWindow::areapro(void)//Used to deal with the size of the area
{
    qDebug() << Q_FUNC_INFO;
    ColorDetect(gHSV.lowH, gHSV.lowS, gHSV.lowV,
                gHSV.highH, gHSV.highS, gHSV.highV,
                BLUR_SIZE);
    ShapeSelect(gHSV.minArea, gHSV.maxArea, &result_data);
    OutPutImage(&mmat);
    //show image
    QImage imshow = Pk24bitColorToQIm(mmat.pBuffer, mmat.width, mmat.height);
    QImage imresize = imshow.scaled(ui->imgLabel->width(), ui->imgLabel->height());
    ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
}
```

（4）将颜色特征和轮廓特征保存至“HSVParams.xml”文件，这里以提取红色特征为例。

```
void MainWindow::onOkBtnClicked()
{
    //First get the HSV value of the modified pattern
    setHSVtoPattern();
    QString fileName = QString("%1/HSVParams.xml").arg(qApp->applicationDirPath());
    QFile file(fileName);

    if (file.open(QIODevice::WriteOnly | QIODevice::Truncate) == false) {
        QMessageBox::information(this, "error", "open file error!!!", QMessageBox::Ok);
        return;
    }
    QDomDocument doc;
    QDomProcessingInstruction instruction;
    instruction = doc.createProcessingInstruction("xml","version='1.0' encoding='UTF-8'");
    doc.appendChild(instruction);
    QDomElement root = doc.createElement("HSVParams");
    doc.appendChild(root);

    QDomElement redHSV = doc.createElement("redHSV");
    root.appendChild(redHSV);
```

```
QDomElement lowHRed = doc.createElement("lowH");
QDomText _lowHRed = doc.createTextNode(QString("%1").arg(gRedHSV.lowH));
lowHRed.appendChild(_lowHRed);
redHSV.appendChild(lowHRed);

QDomElement highHRed = doc.createElement("highH");
QDomText _highHRed = doc.createTextNode(QString("%1").arg(gRedHSV.highH));
highHRed.appendChild(_highHRed);
redHSV.appendChild(highHRed);

QDomElement lowSRed = doc.createElement("lowS");
QDomText _lowSRed = doc.createTextNode(QString("%1").arg(gRedHSV.lowS));
lowSRed.appendChild(_lowSRed);
redHSV.appendChild(lowSRed);

QDomElement highSRed = doc.createElement("highS");
QDomText _highSRed = doc.createTextNode(QString("%1").arg(gRedHSV.highS));
highSRed.appendChild(_highSRed);
redHSV.appendChild(highSRed);

QDomElement lowVRed = doc.createElement("lowV");
QDomText _lowVRed = doc.createTextNode(QString("%1").arg(gRedHSV.lowV));
lowVRed.appendChild(_lowVRed);
redHSV.appendChild(lowVRed);

QDomElement highVRed = doc.createElement("highV");
QDomText _highVRed = doc.createTextNode(QString("%1").arg(gRedHSV.highV));
highVRed.appendChild(_highVRed);
redHSV.appendChild(highVRed);

QDomElement minAreaRed = doc.createElement("minArea");
QDomText _minAreaRed = doc.createTextNode(QString("%1").arg(gRedHSV.minArea));
minAreaRed.appendChild(_minAreaRed);
redHSV.appendChild(minAreaRed);
```

```
        QDomElement maxAreaRed = doc.createElement("maxArea");
        QDomText _maxAreaRed = doc.createTextNode(QString("%1").arg(gRedHSV.maxArea));
        maxAreaRed.appendChild(_maxAreaRed);
    redHSV.appendChild(maxAreaRed);
    QTextStream out(&file);
        doc.save(out, 4);


        file.close();
        initHSVParams();
        CameraSetSnapMode(0, CAMERA_SNAP_TRIGGER);
        QMessageBox::information(this, "Result", "Save the HSV range parameter success! \n"
                                                 "You can get a HSVParams.xml file in your folder.");
    }
```

9.3 视觉分拣功能的实现

Dobot 二次开发需要调用动态链接库才可以控制 Dobot 机械臂。对于通用桌面系统，Dobot 官网已经向 Dobot 二次开发者提供了动态链接库。在 DobotDll 文件夹下可以找到动态链接库的源码和预编译文件，Windows 系统中，将动态链接库所在目录添加到系统 Path 环境变量；Linux 系统中，在~/.bash_profile 文件最后添加下列语句，重启电脑；Mac 系统中，在~/.bash_profile 文件最后添加下列语句，重启电脑。

本视觉分拣实验例程其实现流程如图 9.15 所示。

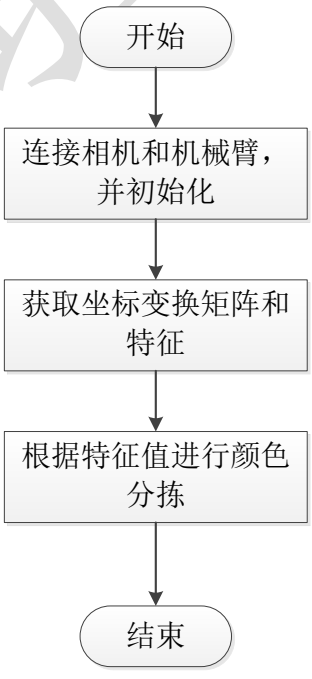

图 9.15 视觉分拣流程

实验环境搭建：

本视觉分拣实验例程是基于 Qt SDK 和 OpenCV 平台开发，因此需安装 Qt Creator 和 OpenCV。Qt Creator 作为集成开发环境，在 Windows 环境下支持两种编译方式：MinGW 和 MSVC。例程采用 MSVC 编译方式，因此还需安装配套的 VisualStudio。

具体安装步骤如下。

（1）准备文件

Qt Creator5.6 以上版本，下载链接：http://download.qt.io/archive/qt/，下载 32 位带 MSVC 编译器的 Qt Creator。

VisualStudio2013 以上版本，下载链接：https://www.visualstudio.com/zh-hans/vs/older-downloads/，VisualStudio 版本必须与 MSVC 编译器版本（包括位数）一致。如：MSVC 编译器版本为 2013，则 VisualStudio 版本也必须为 2013。

（2）VS2015 及 Qt Creator 的安装

本例程以 “ qt-opensource-windows-x86-msvc2015-5.6.1.exe ”（32 位）、“VS2015 ” 和例，在 Windows 10 操作系统下进行安装配置说明，请用户根据实际情况替换。

1）安装“VS2015”，直接一路点击下一步，安装 VS2015 时，切记勾选“Visual C++”选项。

2）安装 Qt5.6.1，直接一路点击下一步，但需要配置编译器与调试器。

首先配置编译器，打开 Qt Creator，进入编译器部分，选择自动检测。如图 9.16 所示，可以看到 Qt 已经自动检测到编译器，不需要手动配置。

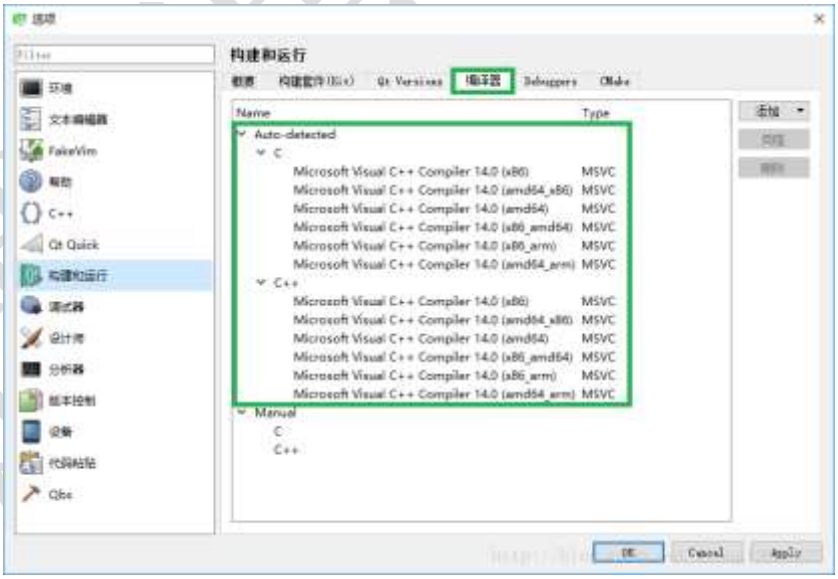

图 9.16 配置编译器

其次配置调试器，由图 9.17 可以看到，自动检测出来的构建套件前面显示的警告符号，调试器部分显示“None”，这说明还没有配置调试器。

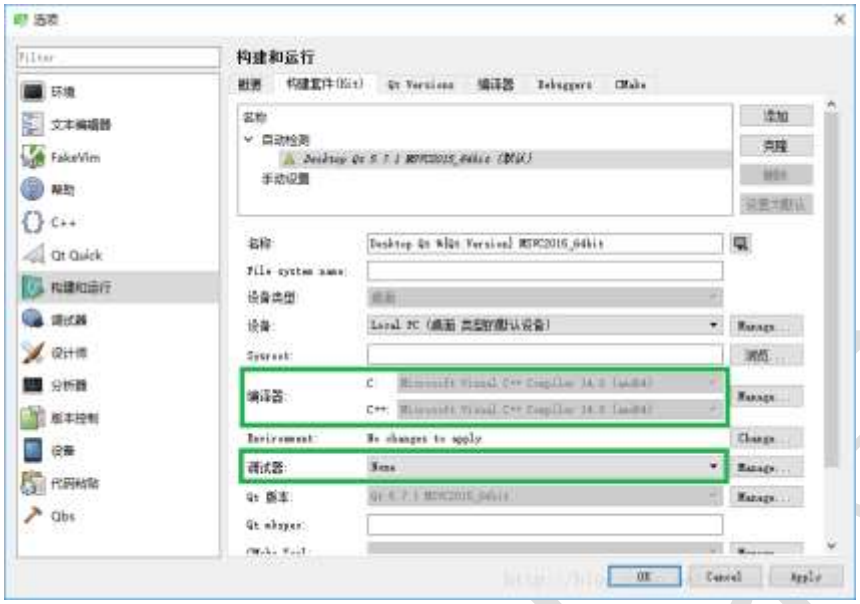

图 9.17 未配置调试器警告

调试器默认情况下是没有的，必须手动下载 windbg 。在安装过程中，需要勾选“Debugging Tools for Windows”，如图 9.18 所示。

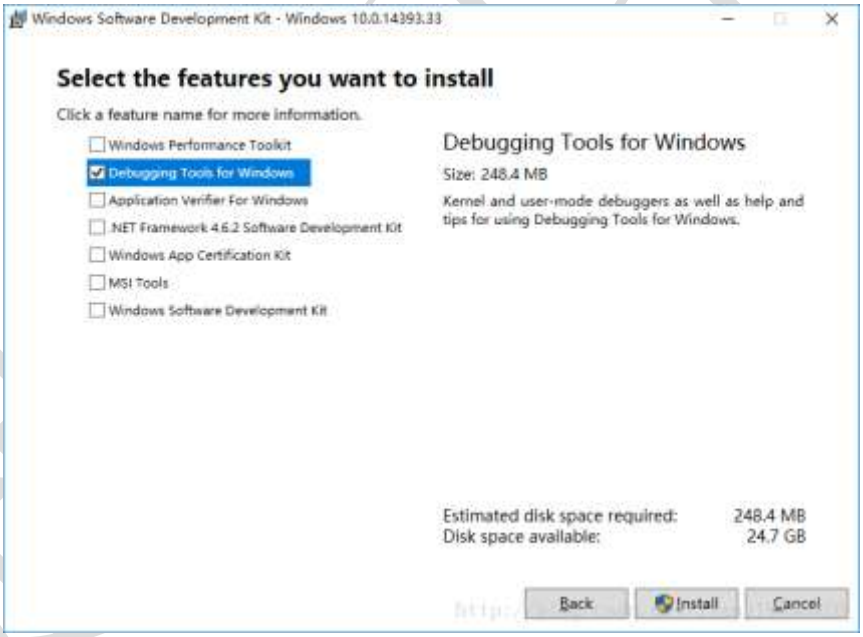

图 9.18 下载 windbg

安装完成之后，打开 Qt Creator（如果已经打开，请先关闭，再重新打开）。这时可以看到 Qt 已经自动检测出调试器了，如图 9.19 所示。

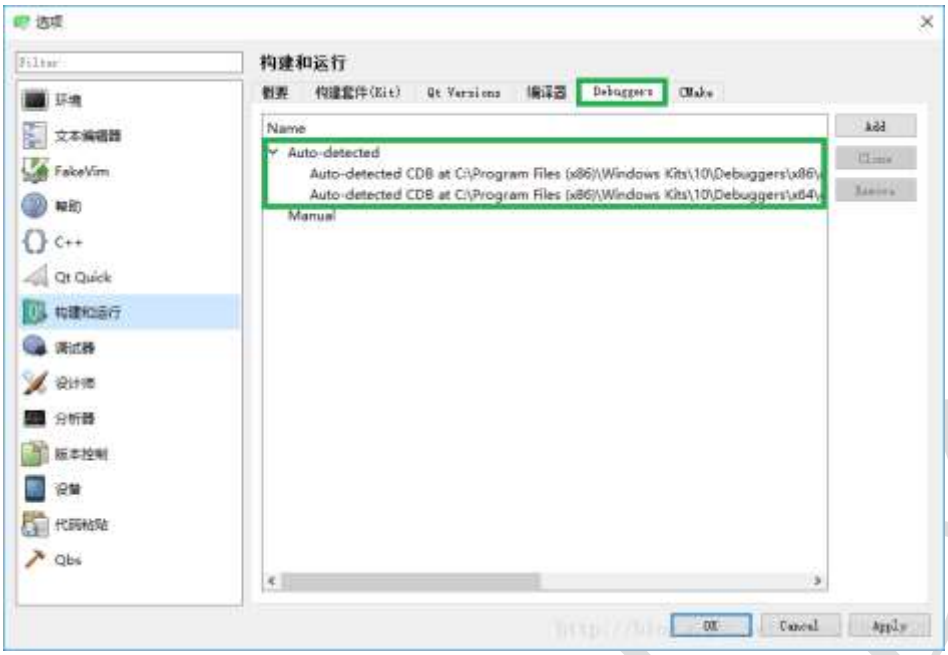

图 9.19 自动检测出调试器

在“构建套件（Kit）”中选择自动检测出来的调试器即可，如图 9.20 所示。

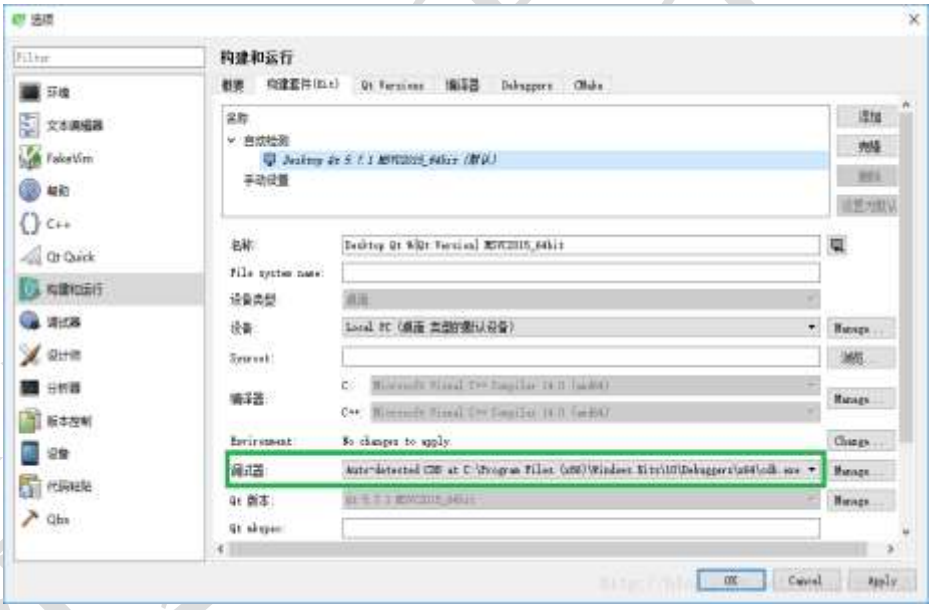

图 9.20 配置调试器

注 9.2：将 Qt 的 bin 文件和 OpenCV 的 bin 文件加入系统环境变量。

实验目的：

颜色分拣是将物品按颜色不同进行分门别类地堆放的作业，本实验以吸盘实现分别抓取蓝色，红色和绿色方块为例。视觉分拣是一款控制 Dobot Magician 机械臂视觉分拣小方块的软件，通过调用之前保存的“CalibraionParams.xml”和“HSVParams.xml”文件，控制 Dobot Magician 机械臂将待抓取物体移动到目标位置。

实验原理:

（1）相机标定，将图像坐标系的坐标点转化为笛卡尔坐标系下机械臂对应的坐标点，即 x,y,z → x',y',z'，以便获取变换矩阵。

（2）HSV 颜色空间由色度（Hue）、饱和度（Saturation）、亮度（Value）三个分量构成，HSV_Debug demo 通过设置 HSV 颜色空间的参数来提取物块颜色特征。

（3）Dobot Magician 机械臂视觉分拣部分，通过调用之前保存的“CalibraionParams.xml”和“HSVParams.xml”文件，并设置待抓取物体需移动到的目标位置和机械臂末端所在的高度来视觉分拣小方块。

这里涉及到的功能实现函数如下：

（1）void MainWindow::connectCam()

（2）void MainWindow::initCamParams()

（3）void MainWindow::connectDobot()

（4）void MainWindow::initDobotParams()

（5）void MainWindow::initSortDlgParams()

（6）void MainWindow::gotoPoint(float x, float y, float r)

（7）void MainWindow::suctionCupCtrl(bool isSuck)

（8）void MainWindow::onConnectBtnClicked()

（9）void MainWindow::onGetImageBtnClicked()

（10）void MainWindow::onSortDlgOkBtnClicked()

（11）void MainWindow::hsvpro(void)

（12）void MainWindow::areapro(void)

（13）void MainWindow::trackproc(int num)

（14）void MainWindow::process(QImage img, unsigned char *buffer)

实验步骤：

（1）参考 2.3 节，完成机械臂的软硬件连接。

（2）在将红、绿、蓝小方块放置于摄像头底板上。

（3）在机械臂末端安装吸盘套件。吸盘的安装方法，参考 1.3 节机械臂的末端执行器部分。

（4）运行视觉分拣程序，可以在弹出的窗口看到带分拣的小方块在相机镜头中的影像，如图 9.21 所示。请确保小方块影像清晰，否则可能无法进行视觉分拣。如果不清晰，可调节相机光圈、焦距或光源亮度。

图 9.21 带分拣物体的影像

（5）用手按住机械臂上的圆形解锁按钮，同时移动机械臂，将末端吸盘移动至待放置带分拣小方块的目标位置，然后松开圆形解锁按钮。获取红、绿、蓝小方块需要移动到的目标位置。

（6）获取机械臂末端所在的高度。用手按住机械臂上的圆形解锁按钮，同时移动机械臂，直至末端吸盘紧贴小方块，然后松开圆形解锁按钮。

（7）完成上述步骤便可以进行，视觉分拣。

注 9.3：部分功能程序介绍如下

（1）读取坐标变换矩阵系数和特征值

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    isSort(false),
    gIsRed(false),
    gIsGreen(false),
    gIsBlue(false)
{
    ui->setupUi(this);
    connectCam();
    initCamParams();
    connectDobot();
    initDobotParams();
    initCalibrateParams();
    initHSVParams();

    connect(ui->homeBtn, SIGNAL(clicked(bool)), this, SLOT(onHomeBtnClicked()));
```

```
    connect(ui->rPlaceBtn, SIGNAL(clicked(bool)), this, SLOT(onRPlaceBtnClicked()));
    connect(ui->gPlaceBtn, SIGNAL(clicked(bool)), this, SLOT(onGPlaceBtnClicked()));
    connect(ui->bPlaceBtn, SIGNAL(clicked(bool)), this, SLOT(onBPlaceBtnClicked()));
    connect(ui->suckZBtn, SIGNAL(clicked(bool)), this, SLOT(onSuckZBtnClicked()));
    connect(ui->startBtn, SIGNAL(clicked(bool)), this, SLOT(onStartBtnClicked()));
    connect(ui->stopBtn, SIGNAL(clicked(bool)), this, SLOT(onStopBtnClicked()));
}
```

（2）设置机械臂末端执行器——吸盘

```
void MainWindow::suctionCupCtrl(bool isSuck)
{
    int result = SetEndEffectorSuctionCup(true, isSuck, true, &queuedCmdIndex);
    if (result == DobotCommunicate_NoError) {
    }
    while(1) {
        if (queuedCmdIndex == queuedCmdCurrentIndex) {
            break;
        }
        GetQueuedCmdCurrentIndex(&queuedCmdCurrentIndex);
    }
    if (isSuck == true) {
        waitCmd(100);
    }
    if (isSuck == false) {
        waitCmd(100);
    }
}
```

（3）设置机械臂的零点位置 Home=（200,0,0,0）

```
void MainWindow::onHomeBtnClicked()
{
    HOMEParams homeParams;
    homeParams.x = 200;
    homeParams.y = 0;
    homeParams.z = 0;
    homeParams.r = 0;
```

```
    SetHOMEParams(&homeParams, false, NULL);
    HOMECmd homeCmd;
    homeCmd.reserved = 0;
    SetHOMECmd(&homeCmd, false, NULL);
}
```

（4）根据不同颜色的特征值，利用机械臂将不同颜色的物块依次分拣至指定的位置。

```
void MainWindow::process(QImage img, unsigned char *buffer)
{
    if (isSort == false) {
        qDebug() << "Camera testing...";
        QImage imresize = img.scaled(ui->imgLabel->width(), ui->imgLabel->height());
        ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
    }else {
        qDebug() << "Sorting Color";
        int width, height, len;
        CameraGetImageSize(0, &width, &height);
        CameraGetImageBufferSize(0, &len, CAMERA_IMAGE_RGB24);

        mmat.pBuffer = buffer;
        mmat.numChannels = 3;
        mmat.width = width;
        mmat.height = height;
        InputSourceImage(&mmat);

        ColorDetect(gHSV.lowH, gHSV.lowS, gHSV.lowV,
                    gHSV.highH, gHSV.highS, gHSV.highV,
                    BLUR_SIZE);
        ShapeSelect(gHSV.minArea, gHSV.maxArea, &result_data);

        OutPutImage(&mmat);
        //Show image
        QImage imshow = Pk24bitColorToQIm(mmat.pBuffer, mmat.width, mmat.height);
        QImage imresize = imshow.scaled(ui->imgLabel->width(), ui->imgLabel->height());
        ui->imgLabel->setPixmap(QPixmap::fromImage(imresize));
```

```
if (result_data.size() != 0) {
    for (int i = 0; i < result_data.size(); i++){
        double r;
        MvPoint mp;
        mp.x = result_data.at(i).center.x;
        mp.y = result_data.at(i).center.y;
        r = result_data.at(i).angle;

        MvPoint2D32f result;
        Transform(mp, &result);//图像坐标系坐标转换到机械臂坐标系
        float dobot_x = result.x;
        float dobot_y = result.y;
        gotoPoint(dobot_x, dobot_y);

        if (gIsRed == true) {
            gotoFirstPoint();
        }
        if (gIsGreen == true) {
            gotoSecondPoint();
        }
        if (gIsBlue == true) {
            gotoThirdPoint();
        }
        CameraTriggerShot(m_index);
        result_data.clear();
    }
}else if (result_data.size() == 0 && gIsRed == true) {
        gIsRed = false;
        gIsGreen = true;
        setGreenHSV();
        CameraTriggerShot(m_index);
}else if (result_data.size() == 0 && gIsGreen == true) {
        gIsGreen = false;
        gIsBlue = true;
```

```
                    setBlueHSV();
                    CameraTriggerShot(m_index);
                }
        }
        delete buffer;
        return;
}
```

实验结果：

机械臂可以将红色小方块从不同颜色的物体中分拣出来，如图 9.22 所示。

图 9.22 实验结果

10 多机械臂协作

10.1 多机械臂协作系统的构成

由于机械臂的工作环境和工作任务复杂度的逐年上升，单个机械臂越来越难以满足任务的需求。于是出现多机械臂协作系统，通过共享资源来弥补单个机械臂能力的不足，完成单个机械臂无法完成的复杂任务或获得更优的系统性能。

根据协作机制的不同，多机械臂协作系统可以分成两类：无意识协作系统和有意识协作系统。无意识机械臂协作系统多数由同样构造的个体机械臂组成，适合于大空间、无时间要求的重复性操作任务；有意识机械臂协作系统多数由构造不同的个体机械臂组成，对通信要求比较高，对协调控制机制的要求也比较高，适合其他复杂的任务。根据系统内部成员的相互关系和拓扑结构，多机械臂协作系统主要有集中式、分布式和混合式这三种。

以多机械臂分拣搬运为例，该系统为集中式无意识机械臂协作系统，以其中一台机械臂作为中心节点，主要实现分拣功能，其余机械臂为普通节点，主要实现搬运功能。

10.2 多机械臂协作系统的通信架构

多机械臂协作系统的成员之间的通信方式可以分为两类：隐式通信和显式通信。隐式通信利用机械臂的运动对产生环境的变化来影响其他其他机械臂的运动，显式通信需要专用硬件通信设备实现机械臂之间直接进行信息交换。通信的传输层一般有三个重要的传输控制协议：串行通讯协议，TCP 协议，UDP 协议。

串行通讯协议： 串行接口简称串口，是采用串行通信方式的扩展接口。串行通讯可以分为单工、半双工和全双工三种。串行通信协议分同步协议和异步协议。最重要的参数是比特率、数据位、停止位和奇偶校验。对于两个进行通信的端口，这些参数必须匹配。

TCP 协议：传输控制协议，采用的是超时重传、数据确认等方式来确保数据可以可靠的传递。TCP 协议的通讯使用时必须建立链接，TCP 服务采用的是基于流，基于流的数据是没有边界限制的，发送端可以源源不断的给里面写数据，而接收端可以一直从里面读数据。

UDP 协议：用户数据报协议，它为网路层提供不可靠、无连接和基于数据报的服务。“不可靠”：意味着无法保证数据可以准确的从发送端发送到接收端。在数据发送失败或者接受端接受不正确的，UDP 只会简单的通知应用程序发送失败。那么应用程序自己得处理超时重传和数据确认等逻辑。每一次发送数据的时候都必须指定接收端的地址。UDP 是基于报的，所以它的长度有限，接收端必须以最小的单位一次性都出来，否则数据将会被丢弃。

表 10-1　TCP 协议和 UDP 协议的区别

	TCP	UDP
是否连接	面向连接	面向非连接
传输可靠性	可靠	不可靠
应用场合	传输大量的数据，对可靠性要求较高的场合	传送少量数据、对可靠性要求不高的场景
速度	慢	快
数据	字节流	报文
连接方式	点到点	一对一，一对多，多对一和多对多
首部开销	20 字节	8 字节

综上所述，本章所涉及的多机械臂分拣搬运系统采用了基于无线局域网 WLAN 的显式通信方式，它利用中心节点的机械臂和普通节点的各个机械臂进行通信。中心节点和普通节点采用的是 TCP/IP 网络模型中传输层的 TCP 协议。在这种模式下，通信双方分别为服务器和客户端，多机械臂分拣搬运系统中的一台机械臂作为中心节点，起到服务器的作用，实现分拣功能；其余机械臂作为普通节点，起到客户端的作用，实现搬运功能。中心节点和普通节点通信之前先建立连接，然后再收发数据。普通节点之间无法进行通信，但所有普通节点和中心节点均可进行通信，所有节点构成星型拓扑结构。

10.2.1　TCP 连接的建立（三次握手）

（1）TCP 服务器进程先创建传输控制块 TCB，时刻准备接受客户进程的连接请求，此时服务器就进入了 LISTEN（监听）状态；

（2）TCP 客户进程也是先创建传输控制块 TCB，然后向服务器发出连接请求报文，这是报文首部中的同部位 SYN=1，同时选择一个初始序列号 seq=x ，此时 TCP 客户端进程进入了 SYN-SENT（同步已发送状态）状态。TCP 规定，SYN 报文段（SYN=1 的报文段）不能携带数据，但需要消耗掉一个序号。

（3）TCP 服务器收到请求报文后，如果同意连接，则发出确认报文。确认报文中应该 ACK=1，SYN=1，确认号是 ack=x+1，同时也要为自己初始化一个序列号 seq=y，此时，TCP 服务器进程进入了 SYN-RCVD（同步收到）状态。这个报文也不能携带数据，但是同样要消耗一个序号。

（4）TCP 客户进程收到确认后，还要向服务器给出确认。确认报文的 ACK=1，ack=y+1，自己的序列号 seq=x+1，此时，TCP 连接建立，客户端进入 ESTABLISHED（已建立连接）状态。TCP 规定，ACK 报文段可以携带数据，但是如果不携带数据则不消耗序号。

（5）当服务器收到客户端的确认后也进入 ESTABLISHED 状态，此后双方就可以开始通信了，TCP 连接的建立过程示意如图 10.2 所示。

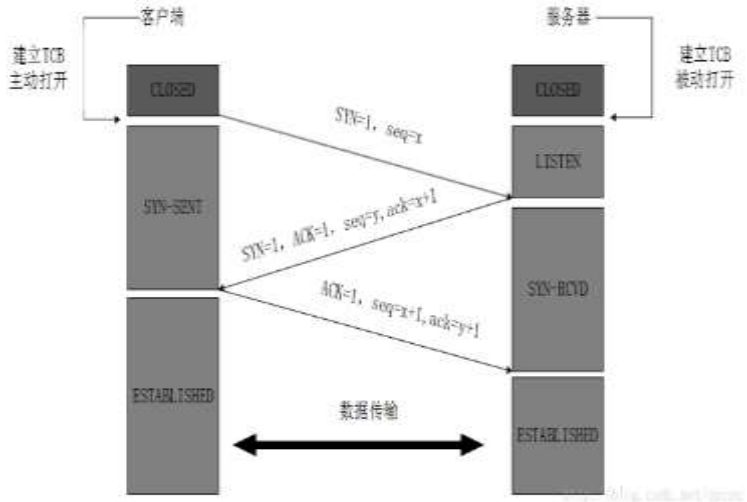

图 10.1 TCP 连接的建立

10.2.2 TCP 连接的释放（四次挥手）

（1）客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为 seq=u（等于前面已经传送过来的数据的最后一个字节的序号加 1），此时，客户端进入 FIN-WAIT-1（终止等待 1）状态。 TCP 规定，FIN 报文段即使不携带数据，也要消耗一个序号。

（2）服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号 seq=v，此时，服务端就进入了 CLOSE-WAIT（关闭等待）状态。TCP 服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个 CLOSE-WAIT 状态持续的时间。

（3）客户端收到服务器的确认请求后，此时，客户端就进入 FIN-WAIT-2（终止等待 2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）。

（4）服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为 seq=w，此时服务器就进入了 LAST-ACK（最后确认）状态，等待客户端的确认。

（5）客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是 seq=u+1，此时，客户端就进入了 TIME-WAIT（时间等待）状态。注意此时 TCP 连接还没有释放，必须经过 2**MSL（最长报文段寿命）的时间后，当客户端撤销相应的 TCB 后，才进入 CLOSED 状态。

（6）服务器只要收到了客户端发出的确认，立即进入 CLOSED 状态。同样，撤销 TCB 后，就结束了这次的 TCP 连接。可以看到，服务器结束 TCP 连接的时间要比客户端早一些。TCP 连接的释放过程示意如图 10.2 所示。

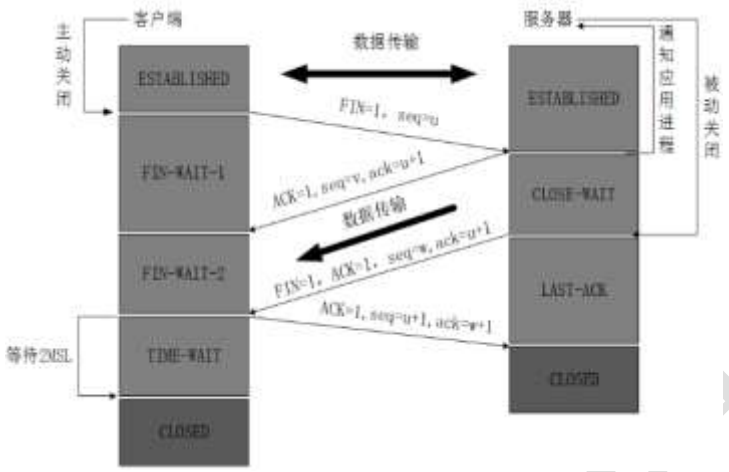

图 10.2 TCP 连接的释放

10.2.3 PC 端调试助手和机器人通讯

串口助手是一款通过电脑串口（包括 USB 口）收发数据并且显示的应用软件，一般用于电脑与嵌入式系统的通讯，借助于它来调试串口通讯或者系统的运行状态。USR-TCP232-Test 串口转网络调试助手把 TCP/IP 网络调试助手与串口调试助手二合一，软件使用简单，软件界面如图 10.3 所示。

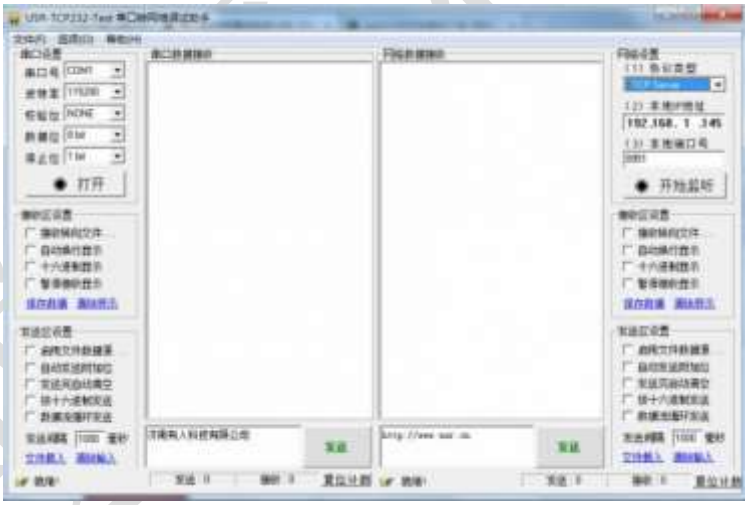

图 10.3 串口转网络调试助手界面

10.3 多机械臂协作系统的实现

实验目的：

（1）掌握 TCP 通讯方式的基本概念和工作流程。

（2）掌握 TCP 服务器和 TCP 客户端的实现方法。

（3）掌握 PC 端调试助手和机器人通讯的实现。

实验方法：

两台机械臂分别连接一个 WiFi 套件，并在 PC 端将这两个 WiFi 套件分别设置成服务器端和客户端。与作服务器的 WiFi 套件相连的机械臂实现分拣功能，分拣结束后会将物体的坐标发给与作客户端的 WiFi 套件相连的机械臂，该机械臂会进行坐标转换，得到物体相对于该机械臂的坐标，然后将物体搬运到另外一个设定好的地方。

实验原理:

多机械臂分拣搬运系统采用的是 TCP/IP 网络模型中传输层的 TCP 通讯方式,这种方式比较灵活方便，传输数据可靠，不会出现丢包等现象。在这种模式下，通信双方分别为服务器和客户端，多机械臂分拣搬运系统中的一台机械臂作为中心节点，起到服务器的作用，实现分拣功能；其余机械臂作为普通节点，起到客户端的作用，实现搬运功能。中心节点和普通节点通信之前先建立连接，然后再收发数据。普通节点之间无法进行通信，但所有普通节点和中心节点均可进行通信，所有节点构成星型拓扑结构。这里涉及到的 API 函数如下：

（1）dType.GetColorSensor(api)：获取颜色传感器值

（2）dType.SetPTPCmd(api, ptpMode, x, y, z, rHead, isQueued=0)：执行 PTP 运动指令

（3）dType.SetEndEffectorSuctionCup(api, enableCtrl, on, isQueued=0)：设置吸盘吸放

（4）dType.GetEndEffectorSuctionCup(api)：获取吸盘吸放状态

（5）dType.SetEndEffectorParams(api, xBias, yBias, zBias, isQueued=0)：设置末端参数

（6）dType.SetColorSensor(api, isEnable)：控制颜色传感器开关

（7）dType.SetInfraredSensor(api, isEnable, infraredPort)：设置红外端口号和状态

（8）dType.GetPose(api)：获取当前位姿

（9）dType.GetInfraredSensor(api, infraredPort)：获取红外端口号和状态

（10）dType.SetPTPJointParams(api, j1Velocity, j1Acceleration, j2Velocity, j2Acceleration, j3Velocity, j3Acceleration, j4Velocity, j4Acceleration, isQueued=0)：设置关节点位参数

（11）dType.SetPTPCommonParams(api, velocityRatio, accelerationRatio, isQueued=0)：设置点位共用参数

（12）dType.SetPTPJumpParams(api, jumpHeight, zLimit, isQueued=0)：设置门型运动参数

（13）dType.SetWAITCmd(api, waitTime, isQueued=0)：执行时间等待功能

实验步骤:

（1）在两台 Dobot Magician 扩展口上分别安装 WIFI 套件和吸盘，然后开启电源，上电后模块会响两声， 并且蓝灯亮，如图 10.4 所示。

图 10.4 安装 WiFi 套件

（2）在 PC 端设置 WIFI 套件的参数。

（3）在与服务器的 WiFi 套件相连的机械臂附近平放一堆颜色不同的物体。

（4）在服务器端编写分拣脚本完成后脱机运行，再在客户端编写搬运脚本完成后脱机运行。

TCP 编程的服务器端一般步骤是：

1) 创建一个 socket，用函数 socket()；
2) 设置 socket 属性，用函数 setsockopt(); * 可选
3) 绑定 IP 地址、端口等信息到 socket 上，用函数 bind();
4) 开启监听，用函数 listen()；
5) 接收客户端上来的连接，用函数 accept()；
6) 收发数据，用函数 send()和 recv()，或者 read()和 write();
7) 关闭网络连接；
8) 关闭监听；

TCP 编程的客户端一般步骤是：

1) 创建一个 socket，用函数 socket()；
2) 设置 socket 属性，用函数 setsockopt();* 可选
3) 绑定 IP 地址、端口等信息到 socket 上，用函数 bind();* 可选
4) 设置要连接的对方的 IP 地址和端口等属性；
5) 连接服务器，用函数 connect()；
6) 收发数据，用函数 send()和 recv()，或者 read()和 write();
7) 关闭网络连接；

具体程序如下：

```
#服务器程序：
import socket
import struct
sk=socket.socket()              #套接字配置
host='192.168.1.106'
```

```
port=9999
sk.bind((host, port))
sk.listen(1)
"""
```

首次启动机械臂时需要归零，归零操作需要在机械臂启动 30S;运行后机械臂吸盘会移动到颜色传感器（带 4 个灯的）上方等待，若位置不准可以根据情况调整位置;站在机械臂后面看，往前是+X，往左是+Y,根据情况调整下面的位置坐标;颜色传感器是 4P 排线接口，接在机械臂底座 GP4 接口;红外传感器是 4P 接口，接在机械臂 GP2，即气泵下面接口。

```
"""
IdeColorPosX = None       #识别颜色位置坐标
IdeColorPosY = None
IdeColorPosZ = None
R = None          #识别颜色
G = None
B = None
MAX = None
GrapX = None     #抓取位置
GrapY = None
GrapZ = None
ToPosY = None    #放置位置
AddRed = None    #颜色叠加
ToPosX = None
AddGreen = None
AddBlue = None
ToInterval = None    #放置间隔
ToPosZ = None
"""获取颜色
"""
def getcoler():
 global deColorPosX,IdeColorPosY,
 IIdeColorPosZ,R,G,B,MAX,ToPosX,ToInterval,ToPosY,ToPosZ,AddRed,AddGreen,AddBlue
    dType.SetPTPCmdEx(api, 0, IdeColorPosX, IdeColorPosY, IdeColorPosZ, 0, 1)
    dType.SetWAITCmdEx(api, 1, 1)
    R = dType.GetColorSensorEx(api, 0)
    G = dType.GetColorSensorEx(api, 1)
```

```
        B = dType.GetColorSensorEx(api, 2)
        MAX = max([R, G, B])
        if MAX == R:
            print('红')
            dType.SetPTPCmdEx(api, 0, (ToPosX + ToInterval), ToPosY, (ToPosZ + AddRed), 0, 1)
            AddRed = AddRed + 25
        elif MAX == G:
            print('绿')
            dType.SetPTPCmdEx(api, 0, ToPosX, ToPosY, (ToPosZ + AddGreen), 0, 1)
            AddGreen = AddGreen + 25
        else:
            print('蓝')
            dType.SetPTPCmdEx(api, 0, (ToPosX - ToInterval), ToPosY, (ToPosZ + AddBlue), 0, 1)
            AddBlue = AddBlue + 25
        dType.SetEndEffectorSuctionCupEx(api, 0, 1)
        dType.SetWAITCmdEx(api, 1, 1)


GrapX = 273
GrapY = 79
GrapZ = 5
IdeColorPosX = 195
IdeColorPosY = 125
IdeColorPosZ = 27
ToPosX = 173
ToPosY = -150
ToPosZ = -49
ToInterval = 40
dType.SetEndEffectorParamsEx(api, 59.7, 0, 0, 1)
AddRed = 0
AddBlue = 0
AddGreen = 0
dType.SetColorSensor(api, 1 ,2)
dType.SetInfraredSensor(api, 1 ,1)
dType.SetWAITCmdEx(api, 1, 1)
```

```
dType.SetPTPJointParamsEx(api,400,400,400,400,400,400,400,400,1)
dType.SetPTPCommonParamsEx(api,100,100,1)
dType.SetPTPJumpParamsEx(api,50,100,1)
current_pose = dType.GetPose(api)
dType.SetPTPCmdEx(api, 2, IdeColorPosX, IdeColorPosY, IdeColorPosZ, current_pose[3], 1)
dType.SetEndEffectorSuctionCupEx(api, 0, 1)
while True:
if (dType.GetInfraredSensor(api, 1)[0]) == 1:
    dType.SetEndEffectorSuctionCupEx(api, 1, 1)
    dType.SetPTPCmdEx(api, 0, GrapX, GrapY, GrapZ, 0, 1)
    getcoler()
    dType.SetPTPCmdEx(api, 0, GrapX, GrapY, IdeColorPosZ, 0, 1)
    break
pos=[ToPosX,ToPosY,ToPosZ]      #分拣后待搬运物体坐标
s=struct.Struct('iii')
packaged_data=s.pack(*pos)      #数据打包
while True:
    print('server waiting...')
    conn,addr=sk.accept()
    client_data=conn.recv(1024)
    print(str(client_data,'utf8'))
    conn.sendall(packaged_data)
    while True:
        pass
conn.close()


#客户端程序：
import socket
import struct
ip_port=('192.168.1.106',9999)
sk=socket.socket()
sk.connect(ip_port)
s=struct.Struct('iii')
sk.sendall(bytes('请发送分拣后待搬运物体坐标','utf8'))
```

```
server_reply=sk.recv(1024)
unpackaged_data=s.unpack(server_reply)   #数据解包
print(unpackaged_data)
pos = dType.GetPose(api)   #以下为搬运相关程序
rHead = pos[3]
dType.SetPTPJointParams(api,200,200,200,200,200,200,200,200)
dType.SetPTPCoordinateParams(api,200,200,200,200)
dType.SetPTPJumpParams(api, 10, 200)
dType.SetPTPCommonParams(api, 100, 100)
x = unpackaged_data[0]
y = unpackaged_data[1]
z = unpackaged_data[2]
moveX=20;moveY=20;moveZ=40;
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z, rHead, 1)
dType.SetEndEffectorSuctionCup(api, 1, 1, 1)
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x, y, z+moveZ, rHead, 1)
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x+moveX, y+moveY, z+moveZ, rHead, 1)
dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode, x+moveX, y+moveY, z, rHead, 1)
dType.SetEndEffectorSuctionCup(api, 0, 1, 1)
sk.close()
```

实验结果：

与作服务器的 WiFi 套件相连的机械臂会用吸盘将一种颜色的物体吸起，然后移动到指定的区域后释放物体，此时与作客户端的 WiFi 套件相连的机械臂会移动到该区域，将这种颜色的物体吸起然后移动到另一处指定区域释放，如图 10.5 所示。

图 10.5 分拣搬运效果图

11 机械臂的 LabVIEW 仿真

本教程结合 LabVIEW 对 Dobot 机械臂进行 3D 建模仿真，LabVIEW 是一种程序开发环境，由美国国家仪器（NI）公司研制开发，类似于 C 和 BASIC 开发环境，但是 LabVIEW 与其他计算机语言的显著区别是：其他计算机语言都是采用基于文本的语言产生代码，而 LabVIEW 使用的是图形化编辑语言 G 编写程序，产生的程序是框图的形式。

使用 LabVIEW 对 Dobot 机械臂进行 3D 建模仿真，需要用到 Dobot SDK for NI LabVIEW，Dobot SDK 分为两个版本，分别为 Dobot_VI 和 Dobot_Serial，本教程选用 Dobot_VI 函数包，适用于 Windows PC 机，函数数量有 90+，编程灵活性高。使用 USB 连接 PC 机与 Dobot Magician，安装 Dobot Windows 驱动，以确认 PC 识别到 Dobot 机械臂。打开 LabVIEW，使用 Dobot_VI 函数包，结合 LabVIEW 其他自带的函数进行机械臂控制与编程。

为了深刻理解机械臂的组成结构，以及各部分的运动关系，将建立 3D 虚拟模型上，实现对机械臂的运动仿真，共分成运动学部分和动力学部分。

11.1 机械臂的运动学分析与应用

运动学，从几何的角度描述和研究物体位置随时间的变化规律的力学分支。以研究质点和刚体这两个简化模型的运动为基础，并进一步研究变形体的运动。运动学分析的是机械构件之间的运动轨迹及受外力所产生外力的变化，只研究速度、加速度、位移、位置、角速度等参量的常以质点为模型的题。这些都随所选参考系的不同而异；而刚体运动学还要研究刚体本身的转动过程、角速度、角加速度等更复杂些的运动特征。运动学是考虑结构在正常工况下有没有干涉运动，和质量、外力没有关系，注意：外力不会影响系统的自由度。

基于对 Dobot 机械臂的运动学分析，结合 LabVIEW 从 Dobot Magician3D 模型的搭建，3D 模型运动仿真，运动学正、逆解算，机械臂末端运动路径规划这四方面进行具体说明和应用。

11.1.1 实验一：Dobot Magician 3D 模型的搭建

实验目的：

（1）深刻理解机械臂的组成结构，以及各部分的运动关系

实验素材：

（1）LabVIEW 软件
（2）wrl 格式的单零件模型文件
（3）Dobot Magician 各部分尺寸图

实验内容及步骤：

（1）使用 LabVIEW 软件，将单零件模型拼接成完整的机械臂；

（2）在机械臂运动副（关节转轴）的位置设置控制接口；给定各关节的角度数值，控制模型的仿真运动；设计友好的 GUI 界面，方面用户对模型的控制，LabVIEW 图形程序框图如图 11.1 所示。

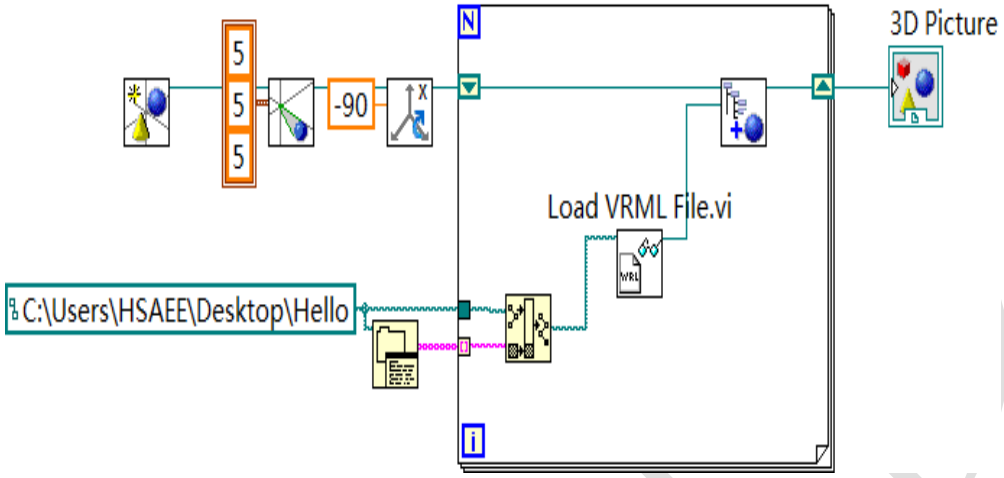

图 11.1 机械臂 3D 模型的搭建程序

实验结果：

Dobot 机械臂 3D 模型的搭建效果如图 11.2 所示。

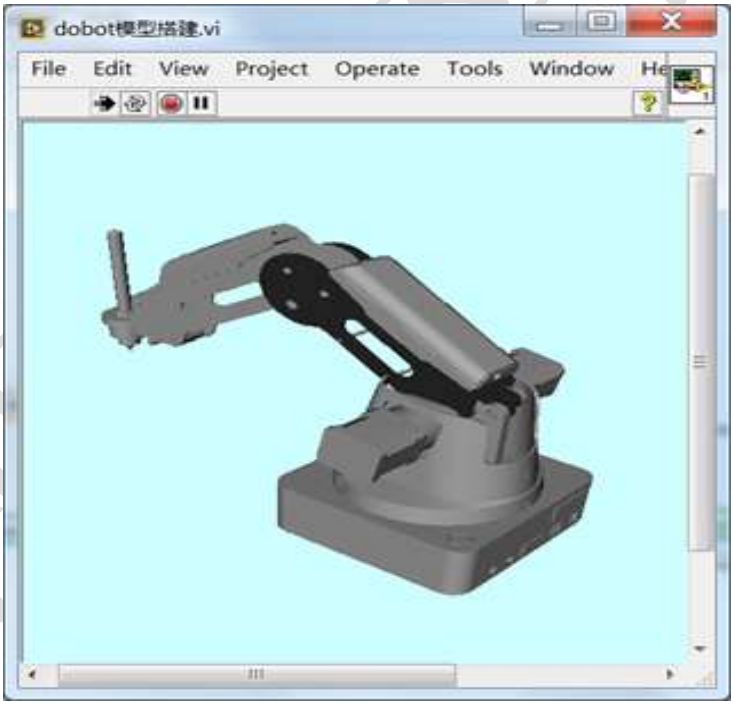

图 11.2 机械臂 3D 模型的效果图

11.1.2 实验二：Dobot Magician 3D 模型运动仿真

实验目的：

（1）在 3D 虚拟模型上，实现对机械臂的运动仿真

实验素材：

（1）LabVIEW 软件

（2）实验 1 搭建的模型

实验内容及步骤：

（1）使用 LabVIEW 软件，将单零件模型拼接成完整的机械臂；

（2）在机械臂运动副（关节转轴）的位置设置控制接口；给定各关节的角度数值，控制模型的仿真运动；

（3）设计友好的 GUI 界面，方面用户对模型的控制；

实验结果：

Dobot 机械臂 3D 模型的运动仿真效果如图 11.3 所示。

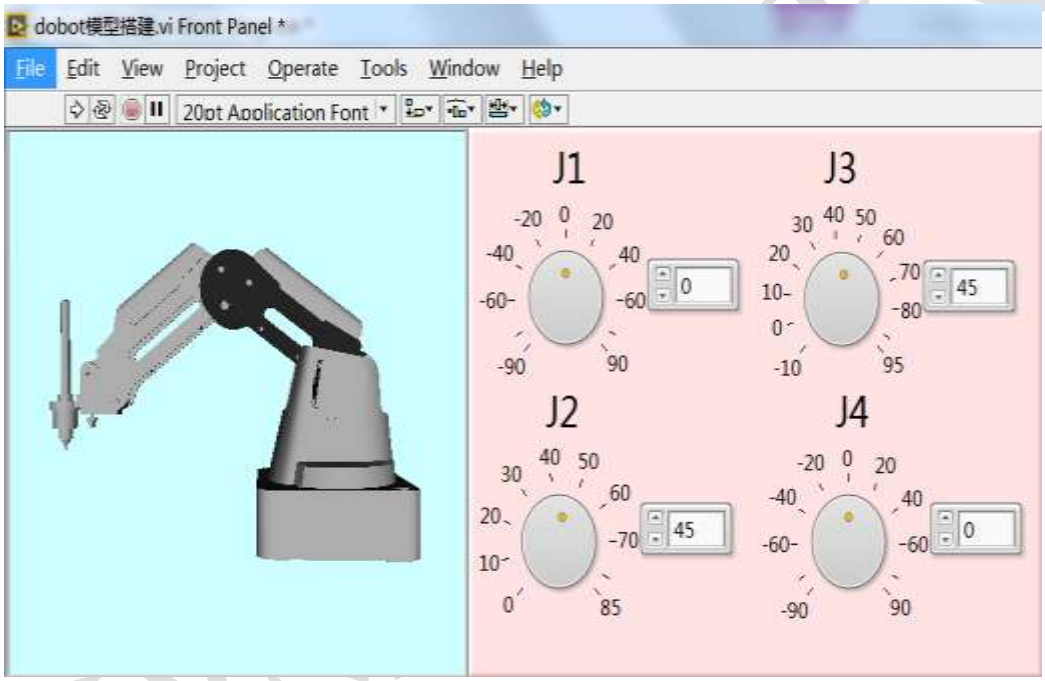

图 11.3 机械臂 3D 模型的运动仿真效果图

11.1.3 实验三：运动学正、逆解算

实验目的：

（1）为了深刻理解机械臂的组成结构，以及各部分的运动关系，将建立 3D 虚拟模型和真机上，练习运动学正、逆解算的过程。

实验素材：

（1）LabVIEW 软件

（2）实验 1 搭建的模型

（3）Dobot Magician 真机

实验内容及步骤：

（1）使用 LabVIEW 软件，将单零件模型拼接成完整的机械臂；

（2）在机械臂运动副（关节转轴）的位置设置控制接口；给定各关节的角度数值，控制模型的仿真运动；设计友好的 GUI 界面

（3）给定一组关节角度，计算末端所在点的直角坐标（正解算）；

（4）给定一个空间点的直角坐标，计算使末端到达此点时的关节角度（逆解算）；

（5）分别在 3D 模型与真机上验证正、逆解算的结果。

实验结果：

Dobot 机械臂 3D 模型的正、逆运动学求解仿真效果如图 11.4 所示。

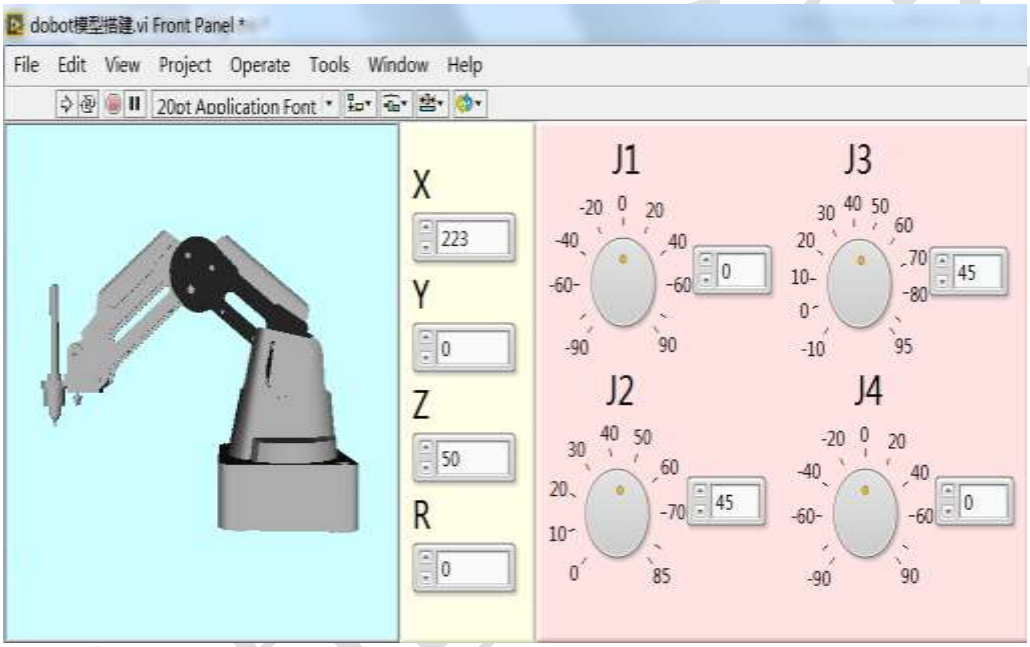

图 11.4 机械臂 3D 模型的正、逆运动学求解效果图

11.1.4 实验四：机械臂末端运动路径规划

实验目的：

（1）规划机械臂的末端沿特定的轨迹运动

实验素材：

（1）LabVIEW 软件

（2）实验 1 搭建的模型

（3）Dobot Magician 真机

实验内容及步骤：

（1）在机械臂的末端执行器安装画笔，通过按机械臂的小臂上 UNLOCK 键，将笔尖定位到纸面高度；

（2）使用 LabVIEW 软件，将单零件模型拼接成完整的机械臂；

（3）在机械臂运动副（关节转轴）的位置设置控制接口；给定各关节的角度数值，控制模型的仿真运

动；

（4）设计友好的 GUI 界面，方面用户对模型的控制；

（5）运行 VI；

（6）通过点击鼠标左键，规划一条直线路径；规划一条圆弧路径；规划一条任意曲线路径；

（7）分别在 3D 模型上实现上述路径，并且 Dobot Magician 真机同时会模仿用户绘制的线条；

实验结果：

Dobot 机械臂末端运动路径规划效果如图 11.5 所示。

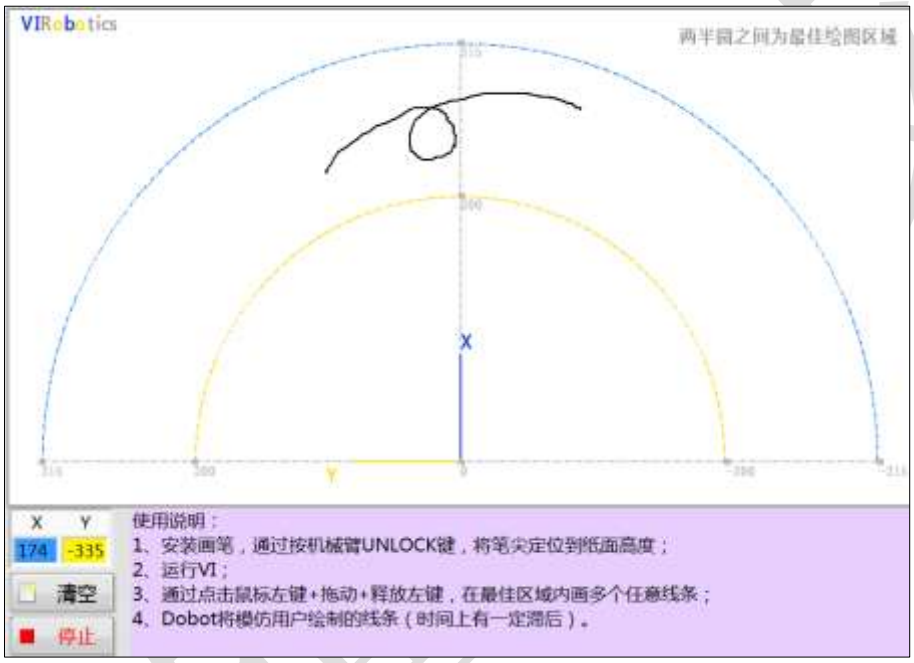

图 11.5 机械臂轨迹规划效果图

注 11.1：Dobot 真机运动与 Labview 仿真图运动在时间上有一定滞后

综上所述，基于 LabVIEW 下的 Dobot_VI 函数包建立了机械臂模型并进行了运动学和轨迹规划仿真，由仿真图可知能够达到给定作业轨迹要求，为 Dobot 机械臂的研究和开发提供了理论依据。

11.2 机械臂的动力学分析与应用

动力学是理论力学的一个分支学科，它主要研究作用于物体的力与物体运动的关系。动力学的研究对象是运动速度远小于光速的宏观物体。动力学是物理学和天文学的基础，也是许多工程学科的基础。许多数学上的进展也常与解决动力学问题有关，区别于运动学，动力学即既涉及运动又涉及受力情况的，或者说跟物体质量有关系的问题。常与牛顿第二定律或动能定理、动量定理等式子中含有质量的相联系。动力学分析的是机械构件之间相互作用引起的动力学变化一般分析应力、变形、震动等，动力学关心的是这些结构在受到载荷时的强度，刚度及稳定性。

基于对 Dobot 机械臂的动力学分析，结合 LabVIEW 从 Dobot Magician 与步进电机，转矩计算与电机选型这几方面进行具体说明和应用。

11.2.1 实验一：步进电机原理与应用

实验目的：

（1）掌握步进式电机的工作原理与控制方法

实验素材：

（1）LabVIEW 软件
（2）外接步进电机
（3）Dobot Magician 真机

实验内容及步骤：

（1）使用 LabVIEW 软件，将单零件模型拼接成完整的机械臂；

（2）在机械臂运动副（关节转轴）的位置设置控制接口；给定各关节的角度数值，控制模型的仿真运动；

（3）设计友好的 GUI 界面，方面用户对模型的控制；

（4）完成对给定步进电机的接线与信号时序设计；

（2）用软件输出信号，控制步进电机的角位移和角速度；

实验结果：

Dobot 机械臂与步进电机连接结果如图 11.6 所示。

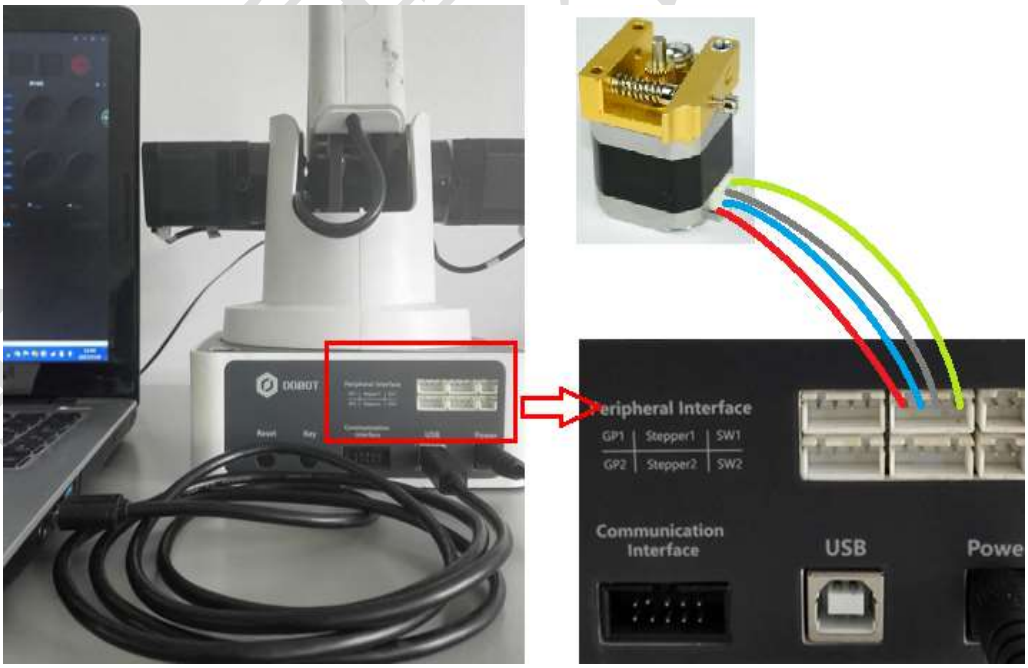

图 11.6 机械臂外接步进电机效果图

11.2.1 实验二：转矩计算与电机选型

实验目的：

（1）掌握刚体旋转的动力学方程，并计算机械臂的最大转矩；
（2）根据最大转矩选择电机的型号；

实验素材：

（1）LabVIEW 软件
（2）Dobot Magician 真机
（3）Dobot Magician 零件参数表（各部分的尺寸、质量、转动惯量、机构传动比等）

实验内容及步骤：

（1）使用 LabVIEW 软件，将单零件模型拼接成完整的机械臂；

（2）在机械臂运动副（关节转轴）的位置设置控制接口；给定各关节的角度数值，控制模型的仿真运动；

（3）设计友好的 GUI 界面，方面用户对模型的控制；

（4）了解机械臂的结构、资源和连线方法；

（5）掌握 Dobot_VI 编程结构；

（6）设计第一个 Dobot_VI 版本的机械臂程序；

实验结果：

Dobot 机械臂与步进电机连接，Dobot_VI 编程运行结果如图 11.7 所示。

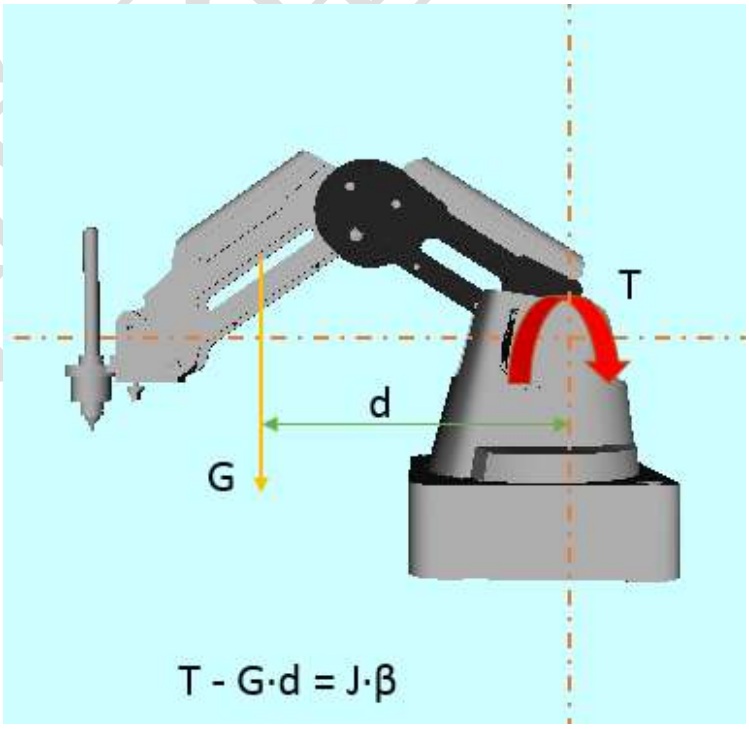

图 11.7 机械臂外接步进电机转矩计算效果图