Atlas 200I DK A2 开发者套件
23.0.RC3
机械臂应用开发指南

文档版本 01

发布日期 2023-11-14

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞声明

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该政策可参考华为公司官方网站的网址：https://www.huawei.com/cn/psirt/vul-response-process。
如企业客户须获取漏洞信息，请访问：https://securitybulletin.huawei.com/enterprise/cn/security-advisory。

目 录

1 样例介绍

E2E智能机械臂样例是基于Atlas 200I DK A2开发者套件的AI推理功能和机械臂的二次开发。该款样例旨在为用户提供方便快捷的基于Atlas 200I DK A2在机器人场景下的开发思路，用户可快速上手、验证，可广泛应用于高校教育、科学研究、工业验证等场景。

1.1 外观结构

图 1-1 正面图

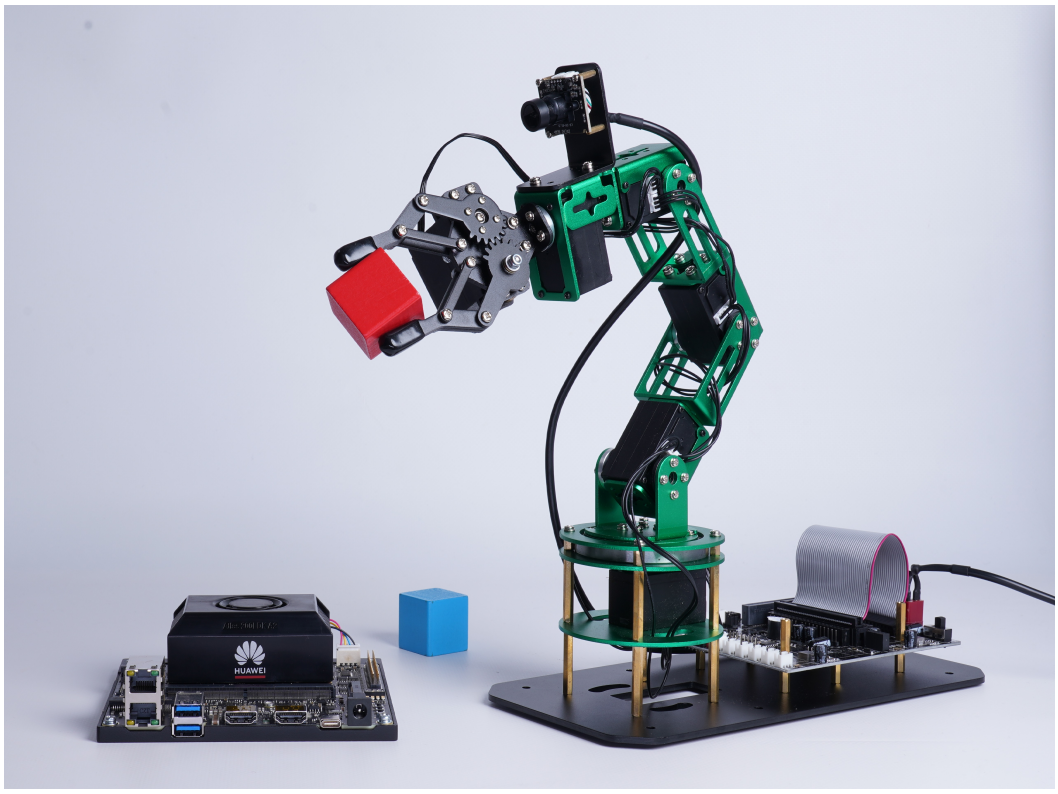


图 1-2 俯视图

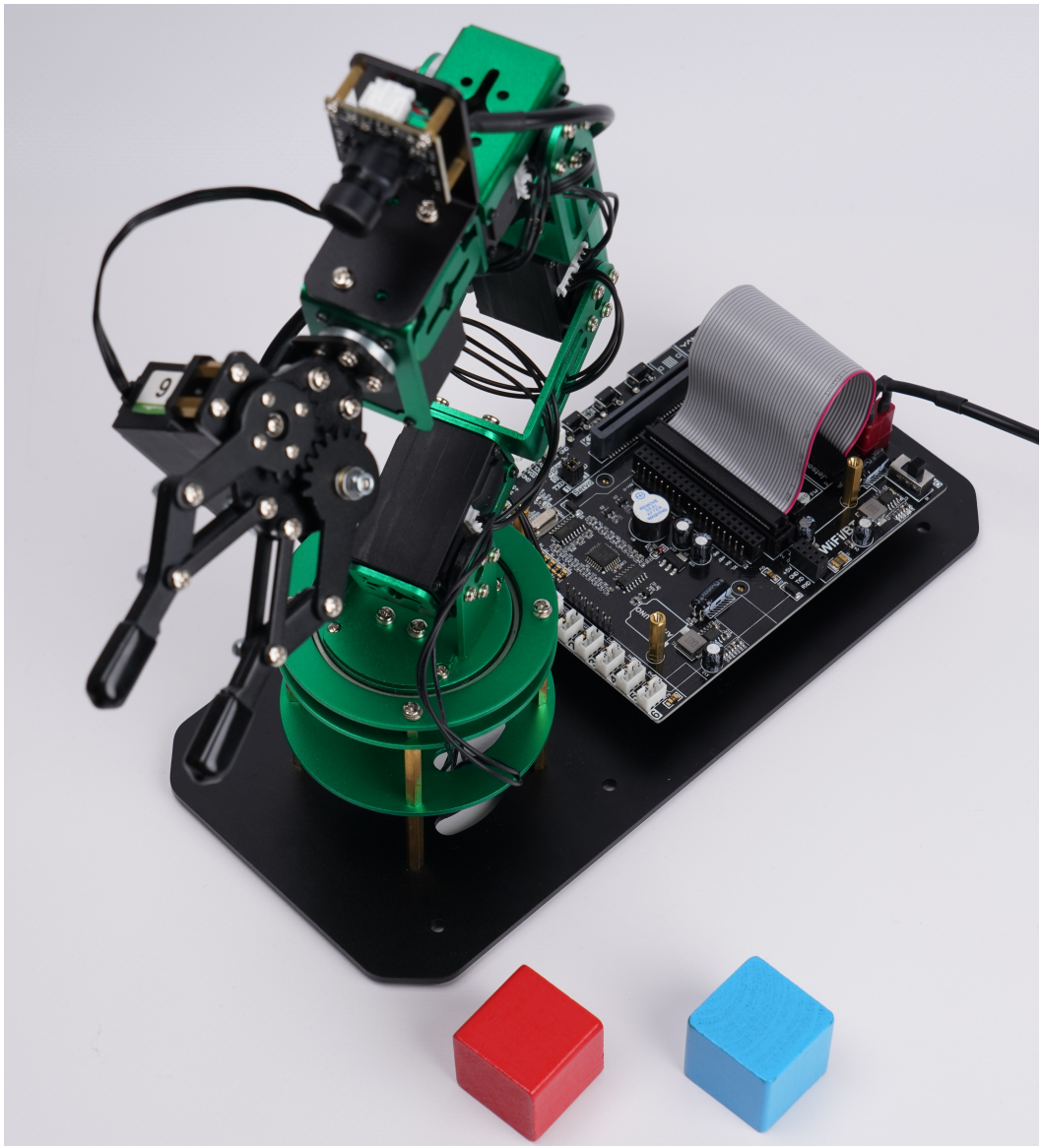
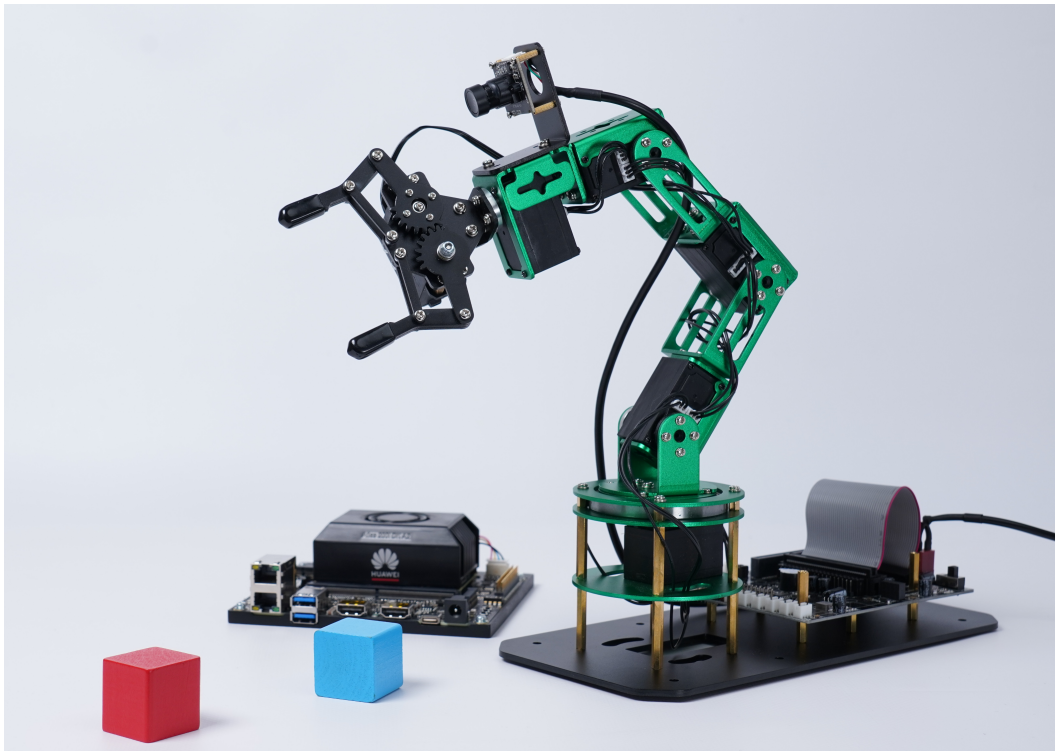


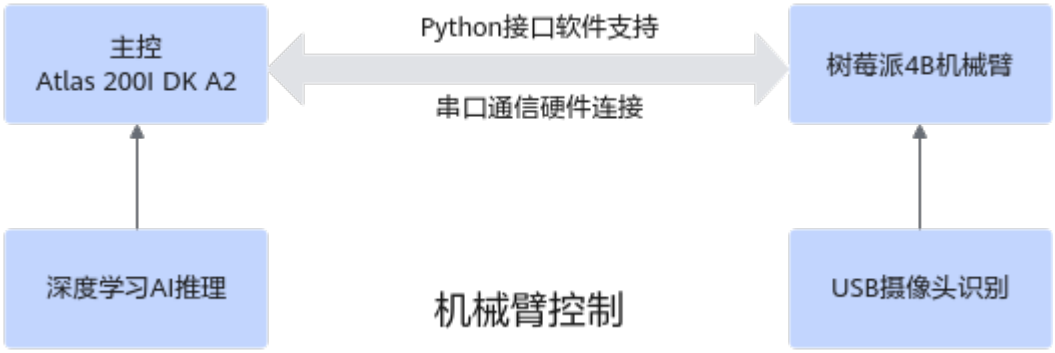
图 1-3 侧面图



1.2 功能与原理介绍

智能机械臂基于Atlas 200I DK A2 开发者套件的深度学习AI推理功能和机械臂的二次开发，通过串口硬件通信和python接口软件支持实现。

图 1-4 实现原理



须知

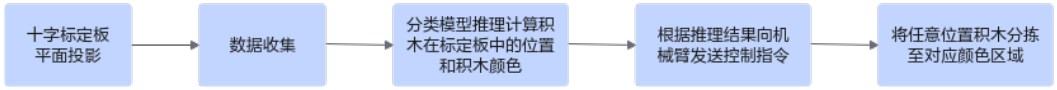
当前功能所用模型，均为已训练并进行模型转换的“.om”文件，如果用户需要使用自己的场景或地图，需参考Atlas 200I DK A2 开发者套件《**使用模型适配工具生成推理应用**》，使用模型适配工具自行训练生成模型，并上传至开发者套件。

任意位置积木分拣

基于开发者套件的内置YoloV5神经网络模型的推理结果，标记十字标定板中积木的位置，并使用机械臂进行分拣。

1. 通过摄像头提取十字标定板轮廓，通过机器视觉方法提取轮廓的四个角点，透视变换轮廓内的平面。
2. 使用模型适配工具训练转换的om模型进行推理，最后根据推理结果下发指令，控制机械臂抓取在摄像头可视范围内的任意位置摆放的积木，并移动至对应颜色区域。

图 1-5 任意位置积木分拣功能原理

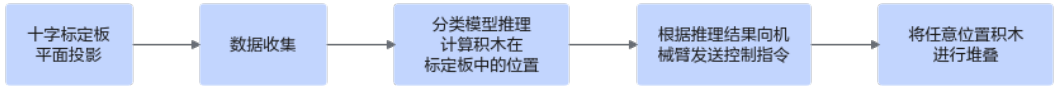


任意位置积木堆叠

基于开发者套件的内置YoloV5神经网络模型的推理结果，标记十字标定板中积木的位置，并使用机械臂进行堆叠。

1. 通过摄像头提取十字标定板轮廓，通过机器视觉方法提取轮廓的四个角点，透视变换轮廓内的平面。
2. 使用模型适配工具训练转换的om模型进行推理，最后根据推理结果下发指令，控制机械臂抓取在摄像头可视范围内的任意位置摆放的积木，并移动至灰色区域并堆叠在一起。

图 1-6 任意位置积木堆叠功能原理



2 样例组装

2.1 准备组件

机械臂应用样例所需组件及组件关键部位参数见**表2-1**。

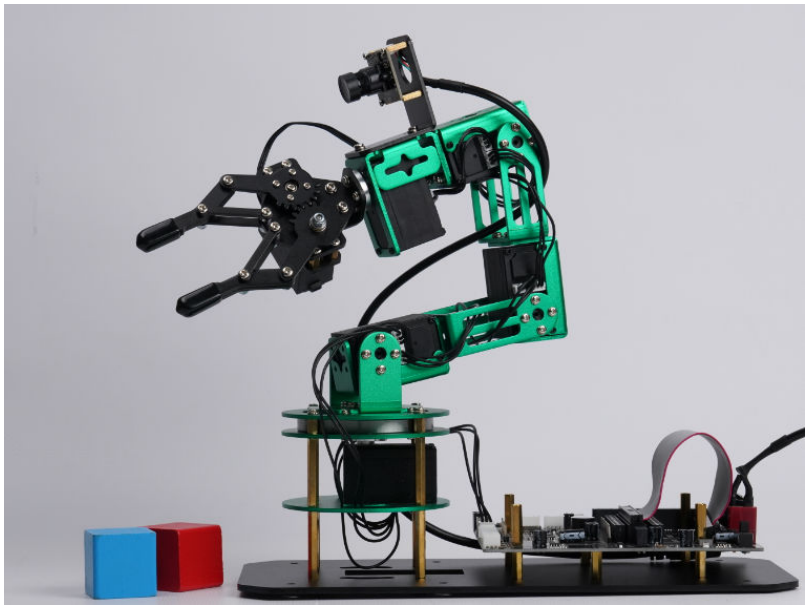
说明

机械臂可以通过**链接**进行单独购买（选择无主板类型），其中物料选型如**图2-1**所示。

表 2-1 机械臂应用样例组件表

模块名称	数量/个	是否需要单独购买
开发者套件及电源适配器	1	否
机械臂及电源适配器	1	是（机械臂套装中包含）
机械臂扩展板	1	
USB摄像头	1	
标定板地图	1	
SD卡	1	
40PIN排线	1	
路由器	1	是
网线	1	是

图 2-1 物料选型图



2.2 组装步骤

须知

因USB摄像头连线长度有限且需要与开发者套件连接，应尽可能地将开发者套件与机械臂放置在邻近的位置。避免后续在机械臂展示样例时，USB摄像头的连线脱落。

步骤1 将开发者套件通过RJ45网线与PC连接。

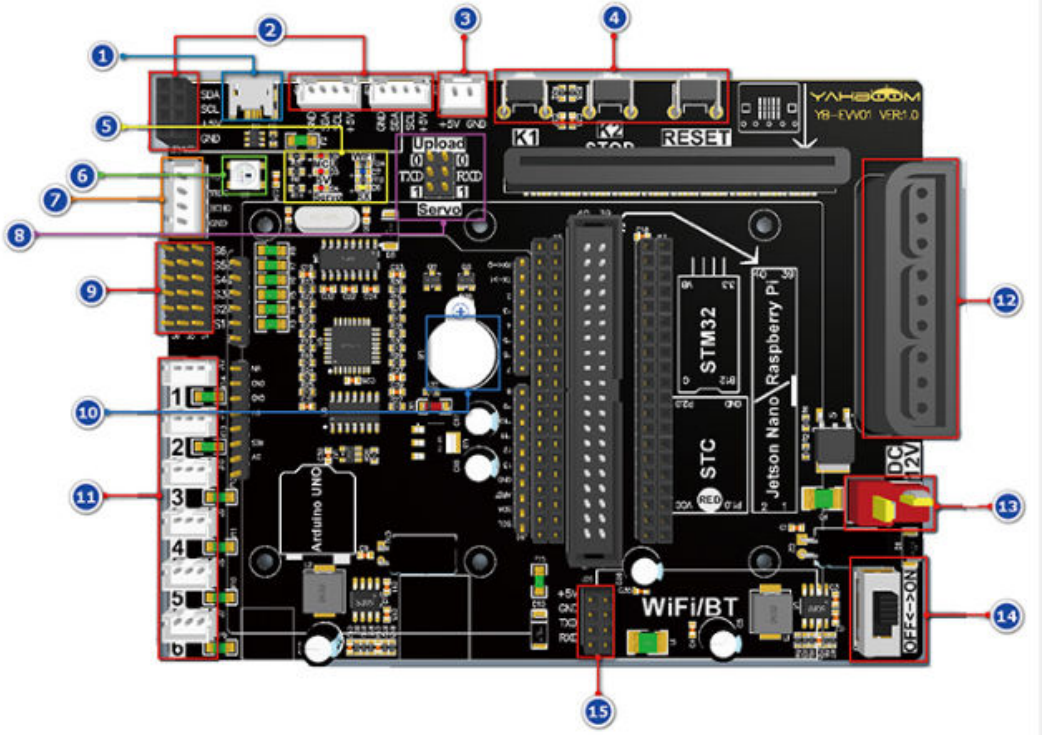
步骤2 参考《快速开始》《快速开始》“一键制卡”章节烧录镜像，将SD卡插入开发者套件的卡槽，**当前样例仅在Ubuntu OS适配验证过，未在openEuler OS适配验证，推荐烧录Ubuntu OS镜像。**

步骤3 参考**通过路由器联网**章节，连接路由器与开发板eth0网口。

步骤4 将开发者套件的40PIN接口与机械臂扩展板的40PIN接口用40PIN排线连接。

机械臂扩展板功能分布图如下：

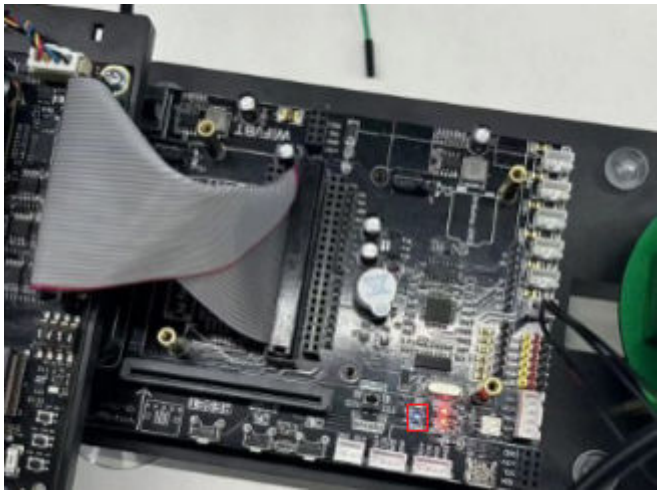
图 2-2 扩展板功能分布



1、Micro USB接口	2、I2C接口	3、散热风扇接口
4、按键：K1+K2键+RESET键	5、状态指示灯	6、RGB灯
7、超声波接口	8、串口功能选择	9、PWM舵机接口
10、蜂鸣器	11、总线舵机接口	12、PS2手柄接收器座
13、T型供电接口	14、电源开关	15、串口接口

机械臂与开发者套件排线的正确接法如下图所示。当正确连接时，扩展板的蓝灯点亮（下图红框所示）。为防止排线被拧断，建议购买一根长的排线。

图 2-3 机械臂与开发者套件排线连接



步骤5 将USB摄像头插入开发者套件USB接口，并将摄像头固定在机械臂上。

步骤6 将开发者套件和机械臂分别接入电源，并将机械臂扩展板电源接口旁的开关拨至“ON”。

----结束

3 快速体验

3.1 获取代码

说明

每开启一个新的远程窗口，都需配置环境变量。

步骤1 远程登录开发者套件。

1. 将PC和路由器接入路由器网络，并登录路由器管理后台查看开发者套件的IP地址（方法请参见**通过路由器联网**）。
2. 使用**root**用户（密码：Mind@123）在PC端的MobaXterm远程连接工具登录开发者套件。

步骤2 远程登录开发者套件，进入“/usr/local”目录运行脚本拉取代码。

```
cd /usr/local
```

步骤3 运行脚本拉取代码。

```
bash E2E_samples_download_tool.sh -d download_destination_path -s source_repository -b branch target_path
```

参数说明：

- -d：指定代码的下载路径。
- -s：指定开源仓库的clone url。
- -b：指定开源仓库分支名称及待下载的项目目录。
- -f：强制更新下载路径中的目录。当样例目录已删除，但重新下载时提示“Already up to date”时可使用此参数。

命令示例：


```
bash E2E_samples_download_tool.sh -d /home/HwHiAiUser/E2ESamples -s https://gitee.com/HUAWEI-ASCEND/ascend-devkit.git -b master src/E2E-Sample/ros2_robot_arm
```

回显如下：

```
Download E2E samples successfully!
```

执行完成后，会在“/home/HwHiAiUser/E2Esamples”目录下生成“src/E2E-Sample/ros2_robot_arm/”目录。

步骤4 进入机械臂目录。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm
```

----结束

3.2 准备环境

步骤1 修改“~/.bashrc”文件。

1. 执行命令打开文件。

   ```
   vi ~/.bashrc
   ```

   在文件末尾新增以下内容。

   ```
   conda deactivate
   ```

   执行**:wq**保存。

   说明

   不使用机械臂时，请将此段代码注释。

2. 执行命令生效环境变量。

   ```
   source ~/.bashrc
   ```

步骤2 下载配置文件与模型文件，

配置文件：将解压后的所有文件上传至“/home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm”目录中。

步骤3 安装依赖。

1. 安装ROS，请依次执行以下命令。

   ```
   apt install -y software-properties-common
   curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
   echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
   apt update
   apt install -y libegl-mesa0 ros-humble-desktop python3-colcon-common-extensions pip
   ```

2. 安装ais_bench，请以此执行以下命令。

   ```
   pip3 install -v 'git+https://gitee.com/ascend/tools.git#egg=aclruntime&subdirectory=ais-bench_workload/tool/ais_bench/backend'
   pip3 install -v 'git+https://gitee.com/ascend/tools.git#egg=ais_bench&subdirectory=ais-bench_workload/tool/ais_bench'
   ```

3. 安装其他依赖。

   ```
   pip3 install -r requirements.txt
   dpkg -i libconsole-bridge0.4_0.4.4+dfsg-1build1_arm64.deb
   dpkg -i liburdfdom-world_1.0.0-2ubuntu0.1_arm64.deb
   ```

步骤4 添加环境变量。

1. 执行命令打开文件。

   ```
   vi ~/.bashrc
   ```


在文件末尾新增以下内容。

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib:/usr/lib/aarch64-linux-gpu:/usr/local/lib
```

执行**:wq**保存。

2. 执行命令生效环境变量。
   ```
   source ~/.bashrc
   ```
3. 替换库文件。
   ```
   cp ./libdofbot_kinemarics.so /usr/lib
   ```

步骤5 安装orocos_kd。

1. 创建编译目录。
   ```
   cd orocos_kdl && mkdir build && cd build
   ```
2. 执行命令进行编译。
   ```
   cmake ..
   make -j4
   ```
3. 执行命令安装依赖。
   ```
   make install
   ```

步骤6 安装机械臂驱动。

```
cd ../.. && cd 0.py_install
python3 setup.py install
cd ..
```

步骤7 编译工作空间。

1. 配置环境变量
   ```
   source setenv.sh
   ```
2. 执行命令编译。
   ```
   cd ros2_ws && colcon build
   ```

----结束

3.3 （可选）校准摄像头

说明

该步骤仅在首次使用机械臂时执行，后续使用时可跳过此步骤。

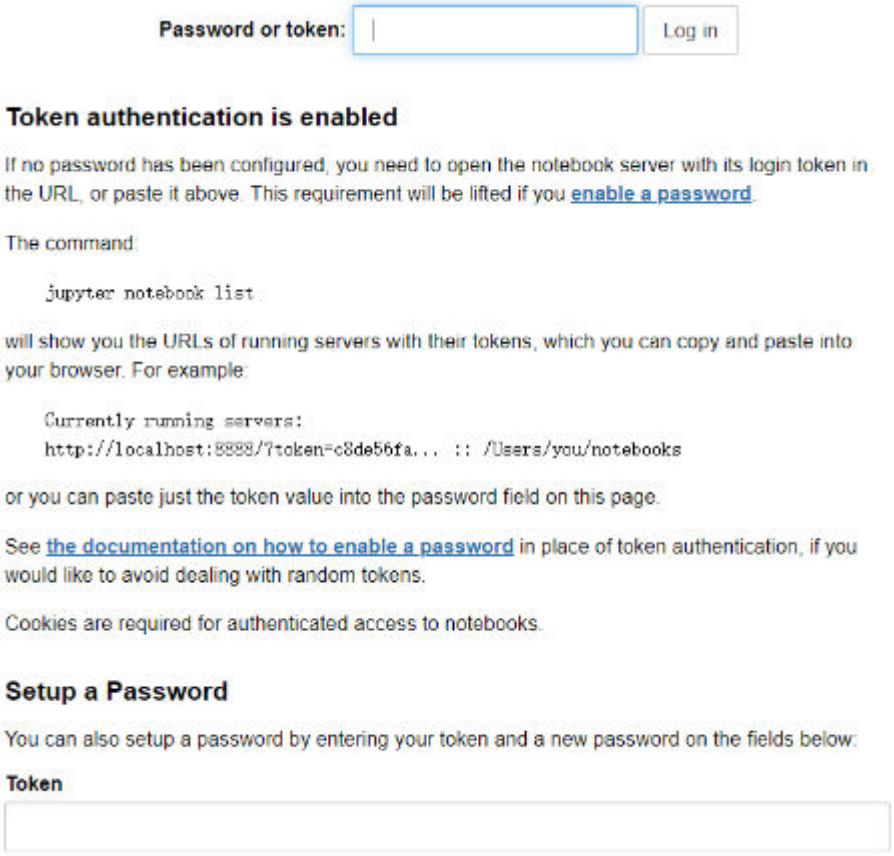
步骤1 打开新的终端窗口，执行命令进入jupyter notebook。

```
jupyter notebook --allow-root 192.168.137.100
```

--allow-root为当前开发者套件的IP，请根据实际IP修改。

首次进入时可能需要输入登录密码或token。

图 3-1 登录界面



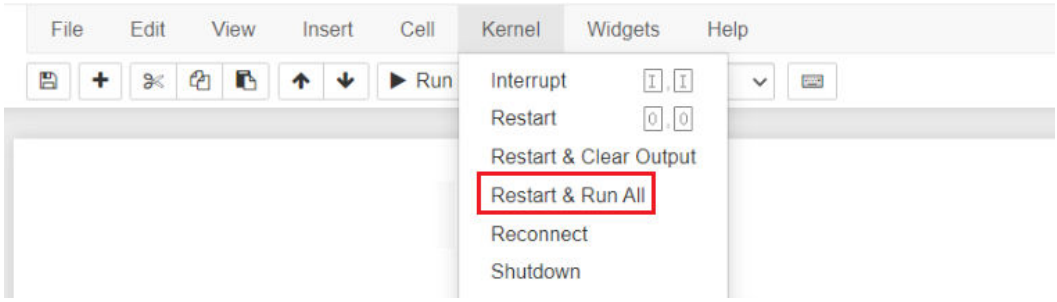
若首次登陆未设置密码，请根据实际回显中的token值登录。回显示例如下，token值为加粗部分。

```
[I 2023-05-09 08:02:59.683 ServerApp] nbclassic | extension was successfully loaded.
[I 2023-05-09 08:02:59.685 ServerApp] Serving notebooks from local directory: /home/HwHiAiUser/samples/notebooks
[I 2023-05-09 08:02:59.685 ServerApp] Jupyter Server 1.23.6 is running at:
[I 2023-05-09 08:02:59.685 ServerApp] http://192.168.137.100:8888/lab?token=a046a76dc21f1504f271c16278ed62ed7fb014aaf38ee807
[I 2023-05-09 08:02:59.685 ServerApp]  or http://127.0.0.1:8888/lab?token=a046a76dc21f1504f271c16278ed62ed7fb014aaf38ee807
[I 2023-05-09 08:02:59.685 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2023-05-09 08:02:59.701 ServerApp]
```

步骤2 在jupyter notebook中逐级进入目录“/root/ros2_robot_arm/ros2_ws/src/dofbot_garbage_yolov5/tools”，双击“相机校准.ipynb”进入代码界面。

步骤3 在代码界面选择“Kernel”选择“Restart & Run All”运行代码。

图 3-2 运行代码

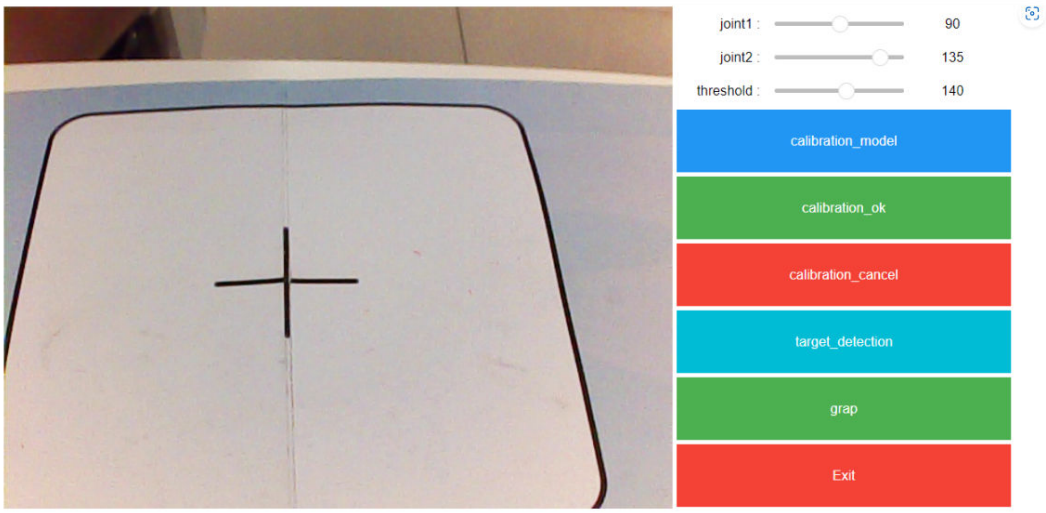


说明

若运行后，下方无回显，请多尝试几次。

成功运行后，画面如下所示。

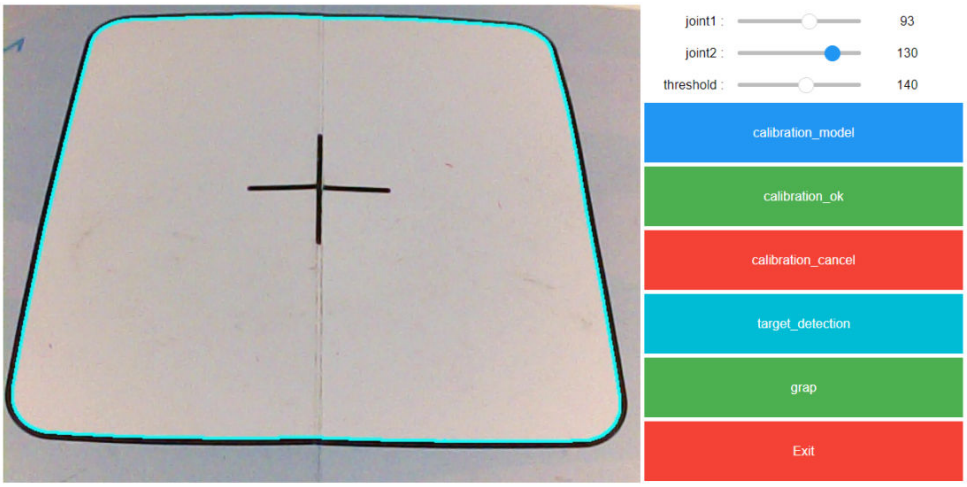
图 3-3 成功运行



步骤4 开始校准摄像头。

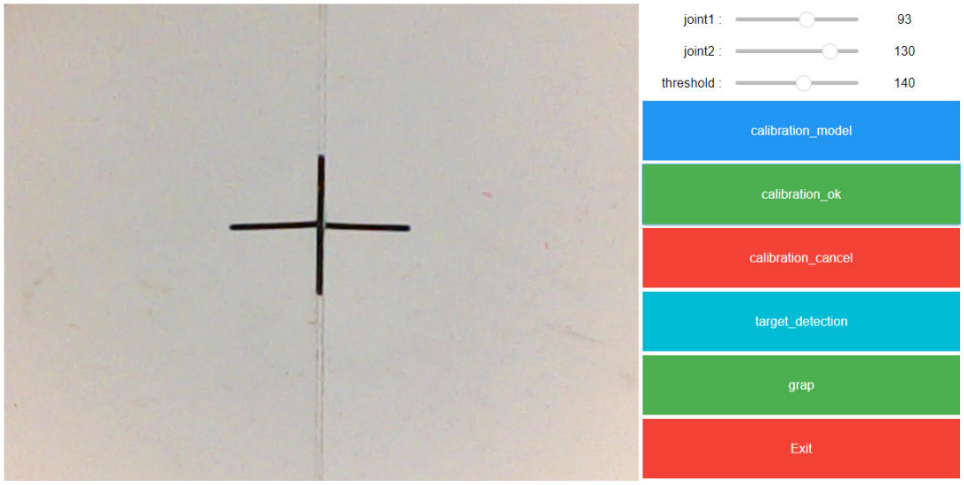
1. 单击“calibration_model”按钮，并调节“joint1”与“joint2”。请保证调试环境的光源充足，确保蓝色边框覆盖整个十字框，画面如下所示。

图 3-4 调节画面



2. 单击“calibration_ok”按钮，可视化界面进入方框内部，画面如下所示。

图 3-5 调节成功



步骤5 返回开发者套件终端，按下键盘“Ctrl”+“C”键，停止jupyter notebook程序。

----结束

3.4 任意位置积木分拣

步骤1 重新开启2个终端窗口，分别进入对应目录执行命令。

- 1号终端。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm
source setenv.sh
ros2 run dofbot_moveit dofbot_server
```

- 2号终端。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm
source setenv.sh
ros2 run dofbot_garbage_yolov5 block_cls
```

说明

任意位置抓取时，需要使用带有标定板框一面的地图，并将摄像头对准标定板框。

步骤2 查看机械臂分拣结果。

----结束

3.5 任意位置积木堆叠

步骤1 重新开启2个终端窗口，分别进入对应目录执行命令。

- 1号终端。
```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm
source setenv.sh
ros2 run dofbot_moveit dofbot_server
```
- 2号终端。
```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm
source setenv.sh
ros2 run robot_arm_color_stacking color_stacking
```

步骤2 查看机械臂分拣结果。

----结束

4 代码实现

4.1 代码文件介绍

机械臂代码文件目录如下所示：

```
RobotArm  源码仓
├── dofbot_color_stacking  色块堆叠功能包
│   ├── CMakeLists.txt
│   ├── package.xml
│   └── scripts  代码以及配置文件
│       ├── color_stacking.txt
│       ├── config 配置文件
│       │   ├── dp.bin  透视变换矩阵
│       │   ├── offset.txt 硬件误差配置
│       │   └── XYT_config.txt 相机标定配置
│       ├── det_utils.py  ACL推理功能文件
│       ├── dofbot_config.py 机械臂描述文件
│       ├── main.py 主函数入口
│       ├── model  模型文件
│       │   ├── coco_names.txt
│       │   └── yolov5s_bs1.om
│       ├── npu_utils.py  ACL推理功能文件
│       ├── stacking_grap.py  色块堆叠抓取功能文件
│       └── stacking_target.py 色块堆叠目标定位文件
├── dofbot_garbage_yolov5  色块分拣功能包
│   ├── dofbot_garbage_yolov5
│   │   ├── a200dk_camera_capture_picture.ipynb
│   │   ├── config  配置文件
│   │   │   ├── dp.bin  透视变换矩阵
│   │   │   ├── offset.txt  硬件误差配置
│   │   │   └── XYT_config.txt  相机标定配置
│   │   ├── det_utils.py  ACL推理功能文件
│   │   ├── dofbot_config.py  机械臂描述文件
│   │   ├── garbage_grap_bak.py
│   │   ├── garbage_grap.py  垃圾分拣抓取脚本
```

```
    ├── garbage_identify.py 垃圾分拣识别脚本
    ├── __init__.py
    ├── main.py  主函数入口
    ├── model  模型文件
    │   ├── coco_names.txt
    │   └── yolov5s_bs1.om
    ├── npu_utils.py  ACL推理功能文件
    ├── __pycache__
    │   ├── det_utils.cpython-310.pyc
    │   ├── dofbot_config.cpython-310.pyc
    │   ├── garbage_grap.cpython-310.pyc
    │   ├── garbage_identify.cpython-310.pyc
    │   ├── npu_utils.cpython-310.pyc
    │   └── single_garbage_identify.cpython-310.pyc
    ├── single_garbage_identify.py
    └── 相机校准.ipynb  相机校准脚本
  ├── package.xml
  ├── resource
  │   └── dofbot_garbage_yolov5_tiny1
  ├── setup.cfg
  ├── setup.py
  └── test
      ├── test_copyright.py
      ├── test_flake8.py
      └── test_pep257.py
├── dofbot_info  机械臂接口功能包
  ├── CMakeLists.txt
  ├── package.xml
  └── srv  服务信息文件
      └── Kinemarics.srv
├── dofbot_moveit  机械臂描述功能包
  ├── CMakeLists.txt
  ├── include
  │   └── dofbot_moveit
  │       └── dofbot_kinemarics.h  机械臂运动学头文件
  ├── launch  工程启动文件
  │   └── dofbot_moveit.launch.py
  ├── lib  坐标求解库文件
  │   └── libdofbot_kinemarics.so
  ├── meshes  三维结构文件
  │   ├── base_link.STL
  │   ├── link1.STL
  │   ├── link2.STL
  │   ├── link3.STL
  │   ├── link4.STL
  │   └── link5.STL
  ├── package.xml
  ├── rviz  RVIZ可视化文件
  │   ├── dofbot.rviz
  │   └── urdf.rviz
  ├── src
  │   └── dofbot_server.cpp  机械臂运动学服务器
  └── urdf
      └── dofbot.urdf  机械臂结构描述文件
├── imgs
  ├── 10.jpg
  ....
  └── 9.PNG
└── README.md
```

4.2 请求与响应配置文件

dofbot_info功能包包含了需要编译的消息文件。“dofbot_info/srv/Kinemarics.srv”文件中，横线上方表示客户端发送请求的消息格式，下方表示服务器回复响应的消息格式。

客户端发送的请求包括以下内容。

- 机械臂基坐标系下的目标坐标（tar_x, tar_y, tar_z）
- 目标姿态角（roll, pitch, yaw）
- 机械臂目前的六个关节信息（cur_joint1, cur_joint2, cur_joint3, cur_joint4, cur_joint5, cur_joint6）
- 用户请求的模式（kin_name）。

除用户请求的模式为字符串类型外，其他消息都应为浮点型。用户请求的模式分为正运动学解（fk）以及逆运动学解（ik）。

- 当模式为正运动学解时，只需填入机械臂目前的六个关节信息。
- 当模式为逆运动学解时，则无需填入机械臂目前的六个关节信息。

服务端响应的消息全为浮点型。

- 当模式为正运动学解时，服务器会响应相对于机械臂基坐标系的到达坐标（x, y, z）和到达姿态角（roll, pitch, yaw）。
- 当模式为逆运动学解时，服务器会响应机械臂的六个关节目标（joint1, joint2, joint3, joint4, joint5, joint6）。

```
float64 tar_x
float64 tar_y
float64 tar_z
float64 roll
float64 pitch
float64 yaw
float64 cur_joint1
float64 cur_joint2
float64 cur_joint3
float64 cur_joint4
float64 cur_joint5
float64 cur_joint6
string  kin_name
---
float64 joint1
float64 joint2
float64 joint3
float64 joint4
float64 joint5
float64 joint6
float64 x
float64 y
float64 z
float64 roll
float64 pitch
float64 yaw
```

4.3 ROS2 服务与机械臂仿真代码

dofbot_moveit功能包包含了ROS2正逆运动学解算服务器、服务器和仿真环境下可视化机械臂的启动文件（launch）、以及用于描述三维机械臂模型的URDF描述文件。

“dofbot_moveit/urdf/dofbot.urdf”文件为约定俗成的机器人三维描述文件（urdf全称为Unified Robot Description Format）。示例代码为机械臂的三维模型定义片段。

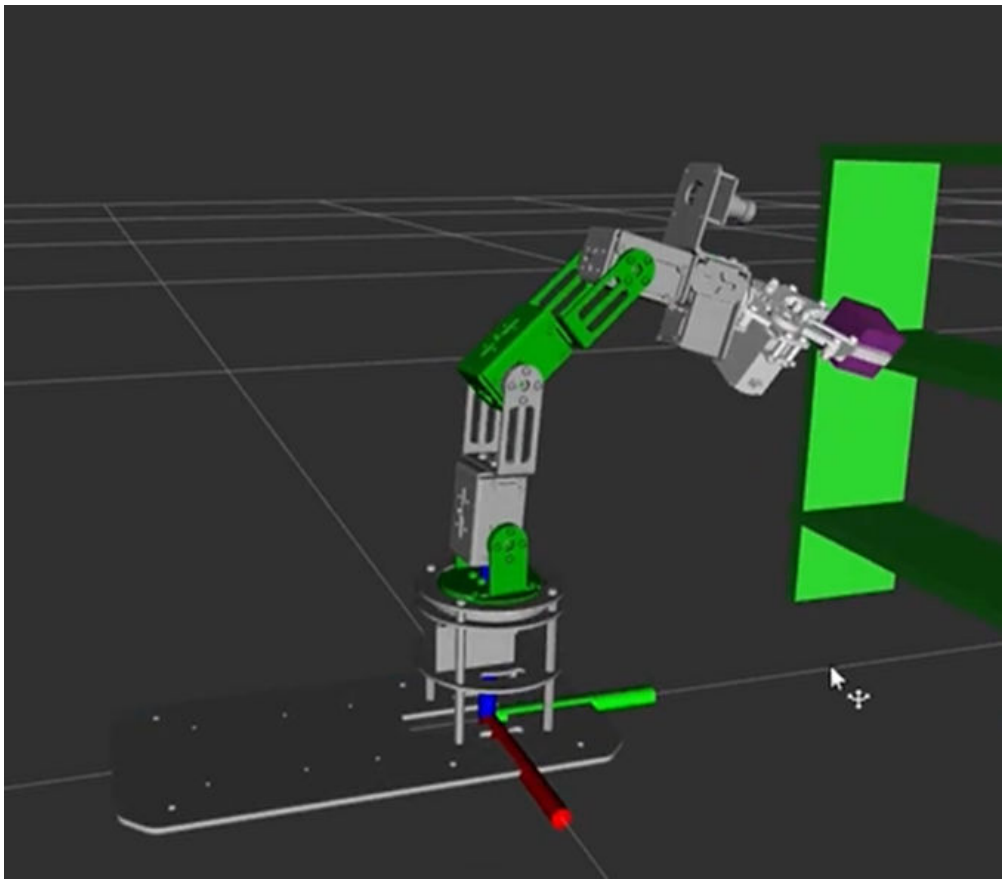
```
<robot name="dofbot">
   <!-- 实体描述 -->
   <link name="base_link">
      <!-- 视觉描述 -->
      <visual>
         <origin
               xyz="0 0 0"
               rpy="0 0 0"/>
```



```
        <geometry>
          <mesh
                filename="package://dofbot_moveit/meshes/base_link.STL"/>
        </geometry>
        <material
              name="">
          <color
                rgba="0.7 0.7 0.7 1"/>
        </material>
      </visual>
      <!-- 碰撞描述 -->
      <collision>
        <origin
              xyz="0 0 0"
              rpy="0 0 0"/>
        <geometry>
          <mesh
                filename="package://dofbot_moveit/meshes/base_link.STL"/>
        </geometry>
      </collision>
    </link>
```

robot标签定义了机器人的名称为dofbot，在此标签下有多个子标签，如link、joint等，每个子标签定义了当前机器人的组成部分。每个组成部分也拥有属于自己的子标签。这些子标签为该组成部分的属性，例如这个具体关节所在空间的位置和姿态，以及关节的颜色及材质等。通过urdf文件对机器人的描述，可以在仿真环境中可视化机械臂，如图4-1所示。

图 4-1 机械臂可视化模型



“dofbot_moveit/src/dofbot_server.cpp”文件定义了正逆运动学解的服务器，其核心代码如下。

srvicecallback()函数为正逆运动学解服务器的回调函数，两个参数分别为客户端发送的请求（request）和服务端回复（response）的响应。二者皆由前文中提到的srv消息类型文件定义。当客户端发送的请求模式为求解正运动学（fk），则调用响应函数，进行响应运算，获得目标位置的坐标和姿态角，存入响应变量（response）中，反之同理。

```
bool srvicecallback(const std::shared_ptr<dofbot_info::srv::Kinemarics::Request> request,
                    std::shared_ptr<dofbot_info::srv::Kinemarics::Response> response) {

    if (request->kin_name == "fk") {
        double joints[]{request->cur_joint1, request->cur_joint2, request->cur_joint3, request->cur_joint4,
                        request->cur_joint5};
        // 定义目标关节角容器
        vector<double> initjoints;
        // 定义位姿容器
        vector<double> initpos;
        // 目标关节角度单位转换,由角度转换成弧度
        for (int i = 0; i < 5; ++i) {
            initjoints.push_back((joints[i] - 90) * DE2RA);
        }
        // 调取正解函数,获取当前位姿
        dofbot.dofbot_getFK(urdf_file, initjoints, initpos);
        cout << "---------- Fk ----------" << a << "---------- Fk ----------" << endl;
        cout << "XYZ坐标：  " << initpos.at(0) << " ," << initpos.at(1) << " ," << initpos.at(2) << endl;
        cout << "Roll,Pitch,Yaw：  " << initpos.at(3) << " ," << initpos.at(4) << " ," << initpos.at(5) << endl;
        response->x = initpos.at(0);
        response->y = initpos.at(1);
        response->z = initpos.at(2);
        response->roll = initpos.at(3);
        response->pitch = initpos.at(4);
        response->yaw = initpos.at(5);
    }
```

4.4 任意位置积木分拣代码

dofbot_garbage_yolov5功能包包含了训练好的yolov5模型及功能脚本。主要为通过yolov5进行色块的识别与定位，根据位置控制机械臂进行积木分类的代码。

“dofbot_garbage_yolov5/dofbot_garbage_yolov5/data_collect.py”为摄像头采集数据的逻辑代码。

```
# 打开摄像头
capture = cv.VideoCapture(0)
# 当摄像头正常打开的情况下循环执行
while capture.isOpened():

    # 读取相机的每一帧
    _, img = capture.read()
    # 统一图像大小
    img = cv.resize(img, (640, 480))

    try:
        write_XYT(XYT_path, [93,130], 140)
    except Exception:
        print("File XYT_config Error !!!")

    dp = np.array([[86, 31], [18, 426], [609, 427], [537, 31]])
    img = calibration.Perspective_transform(dp, img)

    # ----------  Get Image Index ---------------
    img_idx = -1
    with open("/home/HwHiAiUser/RobotArmProject/ros2_dofbot_formal_ws/src/dofbot_garbage_yolov5/
dofbot_garbage_yolov5/config/data_collect_idx.txt", "r") as f:
        contents = f.readline()
        # print(type(contents))
        img_idx = int(contents)
```



```
    stored_path = "/home/HwHiAiUser/RobotArmProject/ros2_dofbot_formal_ws/src/dofbot_garbage_yolov5/dofbot_garbage_yolov5/data/" + str(img_idx) + ".png"
    cv.imwrite(stored_path, img)

    with open("/home/HwHiAiUser/RobotArmProject/ros2_dofbot_formal_ws/src/dofbot_garbage_yolov5/dofbot_garbage_yolov5/config/data_collect_idx.txt", "w") as f:
        img_idx += 1
        f.write(str(img_idx))
    # ---------- Done --------------------------
    break
```

“dofbot_garbage_yolov5/dofbot_garbage_yolov5/dofbot_config.py”为图像检测与机器视觉逻辑代码。

```
    def calibration_map(self, image,xy=None, threshold_num=130):
        '''
        放置方块区域检测函数
        :param image:输入图像
        :return:轮廓区域边点,处理后的图像
        '''
        if xy!=None:
            self.xy=xy
        # 机械臂初始位置角度
        joints_init = [self.xy[0], self.xy[1], 0, 0, 90, 0] # REVISE
        # 将机械臂移动到标定方框的状态
        self.arm.Arm_serial_servo_write6_array(joints_init, 500)
        self.image = image
        self.threshold_num = threshold_num
        # 创建边点容器
        dp = []
        h, w = self.image.shape[:2]
        # print("h",h)
        # 获取轮廓点集(坐标)
        contours = self.Morphological_processing()
        # 遍历点集
        for i, c in enumerate(contours):
            # 计算轮廓区域。
            area = cv.contourArea(c)

            low_bound = None
            high_bound = None
            if self.use_other_cam:
                low_bound = h * w / 2 - 40000
            else:
                low_bound = h * w / 2
            high_bound = h * w

            # 设置轮廓区域范围
            if low_bound < area < high_bound:
                # 计算多边形的矩
                mm = cv.moments(c)
                if mm['m00'] == 0:
                    continue
                cx = mm['m10'] / mm['m00']
                cy = mm['m01'] / mm['m00']
                # 绘制轮廓区域
                cv.drawContours(self.image, contours, i, (255, 255, 0), 2)
                # 获取轮廓区域边点
                dp = np.squeeze(cv.approxPolyDP(c, 100, True))
                # 绘制中心
                if not self.data_collect:
                    cv.circle(self.image, (np.int_(cx), np.int_(cy)), 5, (0, 0, 255), -1)
        return dp, self.image

    def Morphological_processing(self):
        '''
        形态学及去噪处理,并获取轮廓点集
        '''
        # 将图像转为灰度图
```

```
    gray = cv.cvtColor(self.image, cv.COLOR_BGR2GRAY)
    # 使用高斯滤镜模糊图像。
    gray = cv.GaussianBlur(gray, (5, 5), 1)
    # 图像二值化操作
    ref, threshold = cv.threshold(gray, self.threshold_num, 255, cv.THRESH_BINARY)
    # 获取不同形状的结构元素
    kernel = np.ones((3, 3), np.uint8)
    # 形态学开操作
    if self.use_other_cam:
        blur = cv.morphologyEx(threshold, cv.MORPH_OPEN, kernel, iterations=8)
    else:
        blur = cv.morphologyEx(threshold, cv.MORPH_OPEN, kernel, iterations=4)
    # 提取模式
    mode = cv.RETR_EXTERNAL
    # 提取方法
    method = cv.CHAIN_APPROX_NONE
    # 获取轮廓点集(坐标) python2和python3在此处略有不同
    # 层级关系 参数一：输入的二值图，参数二：提取模式，参数三：提取方法。
    contours, hierarchy = cv.findContours(blur, mode, method)
    return contours

def Perspective_transform(self, dp, image):
    '''
    透视变换
    :param dp: 方框边点(左上,左下,右下,右上)
    :param image: 原始图像
    :return: 透视变换后图像
    '''
    if len(dp) != 4:
        return
    upper_left = []
    lower_left = []
    lower_right = []
    upper_right = []
    for i in range(len(dp)):
        if dp[i][0]<320 and dp[i][1]<240:
            upper_left=dp[i]
        if dp[i][0]<320 and dp[i][1]>240:
            lower_left=dp[i]
        if dp[i][0]>320 and dp[i][1]>240:
            lower_right=dp[i]
        if dp[i][0]>320 and dp[i][1]<240:
            upper_right=dp[i]
    # 原图中的四个顶点
    pts1 = np.float32([upper_left, lower_left, lower_right, upper_right])
    # 变换后的四个顶点
    pts2 = np.float32([[0, 0], [0, 480], [640, 480], [640, 0]])
    # 根据四对对应点计算透视变换。
    M = cv.getPerspectiveTransform(pts1, pts2)
    # 将透视变换应用于图像。
    Transform_img = cv.warpPerspective(image, M, (640, 480))
    return Transform_img
```

“dofbot_garbage_yolov5/dofbot_garbage_yolov5/garbage_grap.py”为机械臂抓取与移动分拣的逻辑代码。

```
def move(self, joints, joints_down):
    '''
    移动过程
    :param joints: 移动到物体位置的各关节角度
    :param joints_down: 机械臂抬起各关节角度
    '''
    joints_uu = [90, 80, 50, 50, 265, self.grap_joint]
    # 抬起
    joints_up = [joints_down[0], 80, 50, 50, 265, 30]
    # 移动至物体位置上方
    self.arm.Arm_serial_servo_write6_array(joints_uu, 1000)
    sleep(1)
    # 开合夹爪
    # for i in range(5):
```



```
        #     self.arm.Arm_serial_servo_write(6, 180, 100)
        #     sleep(0.08)
        #     self.arm.Arm_serial_servo_write(6, 30, 100)
        #     sleep(0.08)
        # 松开夹爪
        self.arm.Arm_serial_servo_write(6, 0, 500)
        sleep(0.5)
        # 移动至物体位置
        self.arm.Arm_serial_servo_write6_array(joints, 500)
        sleep(0.5)
        # 进行抓取,夹紧夹爪
        self.arm.Arm_serial_servo_write(6, self.grap_joint, 500)
        sleep(0.5)
        # 架起
        self.arm.Arm_serial_servo_write6_array(joints_uu, 1000)
        sleep(1)
        # 抬起至对应位置上方
        self.arm.Arm_serial_servo_write(1, joints_down[0], 500)
        sleep(0.5)
        # 抬起至对应位置
        self.arm.Arm_serial_servo_write6_array(joints_down, 1000)
        sleep(1)
        # 释放物体,松开夹爪
        self.arm.Arm_serial_servo_write(6, 30, 500)
        sleep(0.5)
        # 抬起
        self.arm.Arm_serial_servo_write6_array(joints_up, 1000)
        sleep(1)

    def arm_run(self, name, joints):
        '''
        机械臂移动函数
        :param name:识别的垃圾名称
        :param joints: 反解求得的各关节角度
        '''
        # 有害垃圾--红色
        if name == "Syringe" or name == "Used_batteries" or name == "Expired_cosmetics" or name ==
"Expired_tablets" and self.move_status == True:
            # 此处设置,需执行完本次操作,才能向下运行
            self.move_status = False
            # print("有害垃圾")
            # print(joints[0], joints[1], joints[2], joints[3], joints[4])
            # 获得目标关节角
            joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
            # 移动到垃圾桶前对应姿态
            # joints_down = [45, 80, 35, 40, 265, self.grap_joint]
            # 移动到垃圾桶位置放下对应姿态
            joints_down = [45, 50, 20, 60, 265, self.grap_joint]
            # 移动
            self.move(joints, joints_down)
            # 移动完毕
            self.move_status = True
        # 可回收垃圾--蓝色
        if name == "Zip_top_can" or name == "Newspaper" or name == "Old_school_bag" or name == "Book"
and self.move_status == True:
            self.move_status = False
            # print("可回收垃圾")
            # print(joints[0], joints[1], joints[2], joints[3], joints[4])
            joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
            # joints_down = [27, 110, 0, 40, 265, self.grap_joint]
            joints_down = [27, 75, 0, 50, 265, self.grap_joint]
            self.move(joints, joints_down)
            self.move_status = True
        # 厨余垃圾--绿色
        if name == "Fish_bone" or name == "Watermelon_rind" or name == "Apple_core" or name ==
"Egg_shell" and self.move_status == True:
            self.move_status = False
            # print("厨余垃圾")
            # print(joints[0], joints[1], joints[2], joints[3], joints[4])
```



```
            joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
            # joints_down = [152, 110, 0, 40, 265, self.grap_joint]
            joints_down = [147, 75, 0, 50, 265, self.grap_joint]
            self.move(joints, joints_down)
            self.move_status = True

        # 其他垃圾--灰色
        if name == "Yellow" or name == "Cigarette_butts" or name == "Toilet_paper" or name == "Peach_pit"
or name == "Disposable_chopsticks" and self.move_status == True:
            self.move_status = False
            # print("其他垃圾")
            # print(joints[0], joints[1], joints[2], joints[3], joints[4])
            joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
            # joints_down = [137, 80, 35, 40, 265, self.grap_joint]
            joints_down = [133, 50, 20, 60, 265, self.grap_joint]
            self.move(joints, joints_down)
            self.move_status = True
```

4.5 任意位置积木堆叠代码

dofbot_color_stacking功能包包含了训练好的yolov5模型及功能脚本。主要为通过yolov5进行色块的识别与定位，根据位置控制机械臂进行积木堆叠的代码。

“dofbot_color_stacking/scripts/data_collect.py”为摄像头采集积木图像数据和处理的逻辑代码。

```
    def get_pos(self):
        '''
        获取识别信息
        :return: 名称,位置
        '''
        # 复制原始图像,避免处理过程中干扰
        img = self.image.copy()

        pred, names, drawed_res = infer_image(img, model, labels_dict, cfg)
        self.frame = drawed_res
        if len(pred) >= 1:
            cv.putText(self.frame, "Fix Infer Result, Waiting for Robot Arm Finish..", (30, 50),
cv.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)

        data = self.bridge.cv2_to_imgmsg(drawed_res, encoding="bgr8")
        self.image_pub.publish(data)

        pred = [pred]
        msg = {}
        gn = torch.tensor([640, 480, 640, 480])
        if pred:
            # Process detections
            cgz_ctr = 0
            for i, det in enumerate(pred):  # detections per image
                for *xyxy, conf, cls in reversed(det):
                    prediction_status = True
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                    label = '%s %.2f' % (names[int(cls)], conf)
                    # get name
                    name = names[int(cls)]
                    name = name + str(cgz_ctr)

                    if prediction_status:
                        point_x = np.int_(xywh[0] * 640)
                        point_y = np.int_(xywh[1] * 480)
                        cv.circle(self.image, (point_x, point_y), 5, (0, 0, 255), -1)
                        # plot_one_box(xyxy, self.image, label=label, color=colors[int(cls)], line_thickness=2)

                        # 计算方块在图像中的位置
                        (a, b) = (round(((point_x - 320) / 4000), 5), round(((480 - point_y) / 3000) * 0.8+0.19, 5))
```



```
                    msg[name] = (a, b)
                    cgz_ctr += 1
        return msg


    def get_Sqaure(self, color_hsv):
        '''
        颜色识别,获得方块的坐标
        '''
        (lowerb, upperb) = color_hsv
        # 复制原始图像,避免处理过程中干扰
        mask = self.image.copy()
        # 将图像转换为HSV。
        hsv = cv.cvtColor(self.image, cv.COLOR_BGR2HSV)
        # 筛选出位于两个数组之间的元素。
        img = cv.inRange(hsv, lowerb, upperb)
        # 设置非掩码检测部分全为黑色
        mask[img == 0] = [0, 0, 0]
        # 获取不同形状的结构元素
        kernel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))
        # 形态学闭操作
        dst_img = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
        # 将图像转为灰度图
        dst_img = cv.cvtColor(dst_img, cv.COLOR_RGB2GRAY)
        # 图像二值化操作
        ret, binary = cv.threshold(dst_img, 10, 255, cv.THRESH_BINARY)
        # 获取轮廓点集(坐标) python2和python3在此处略有不同
        contours, heriachy = cv.findContours(binary, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)  # 获取轮
廓点集(坐标)
        for i, cnt in enumerate(contours):
            # boundingRect函数计算边框值，x，y是坐标值，w，h是矩形的宽和高
            x, y, w, h = cv.boundingRect(cnt)
            # 计算轮廓的面积
            area = cv.contourArea(cnt)
            # 面积限制
            if area > 1000:
                # 中心坐标
                point_x = float(x + w / 2)
                point_y = float(y + h / 2)
                # 在img图像画出矩形，(x, y), (x + w, y + h)是矩形坐标，(0, 255, 0)设置通道颜色，2是设置线条粗
度
                cv.rectangle(self.image, (x, y), (x + w, y + h), (0, 255, 0), 2)
                # 绘制矩形中心
                cv.circle(self.image, (int(point_x), int(point_y)), 5, (0, 0, 255), -1)
                # # 在图片中绘制结果
                cv.putText(self.image, self.color_name, (int(x - 15), int(y - 15)),
                           cv.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 255), 2)
                # 计算方块在图像中的位置
                (a, b) = (round(((point_x - 320) / 4000), 5), round(((480 - point_y) / 3000) * 0.8+0.19, 5))
#                 print("------------------- identify up --------------------")
#                 print(a, b)
#                 print("------------------- identify down --------------------")
                return (a, b)

    def server_joint(self, posxy):
        '''
        发布位置请求,获取关节旋转角度
        :param posxy: 位置点x,y坐标
        :return: 每个关节旋转角度
        '''
        # 等待server端启动
        self.client.wait_for_service()
        # 创建消息包
        # request = kinemaricsRequest()
        request = Kinemarics.Request()
        request.tar_x = posxy[0]
        request.tar_y = posxy[1] + self.offset
        request.kin_name = "ik"
        try:
```

```
        # response = self.client.call(request)
        self.future = self.client.call_async(request)
        rclpy.spin_until_future_complete(self.node, self.future)
        response = self.future.result()
        if response:
            # 获得反解响应结果
            joints = [0.0, 0.0, 0.0, 0.0, 0.0]
            joints[0] = response.joint1
            joints[1] = response.joint2
            joints[2] = response.joint3
            joints[3] = response.joint4
            joints[4] = response.joint5
            # 当逆解越界,出现负值时,适当调节.
            if joints[2] < 0:
                joints[1] += joints[2] * 3 / 5
                joints[3] += joints[2] * 3 / 5
                joints[2] = 0
            # print joints
            return joints
    except Exception:
        # rospy.loginfo("arg error")
        print("arg error")
```

“dofbot_color_stacking/scripts/dofbot_config.py”为图像检测与机器视觉逻辑代码。

```
def calibration_map(self, image,xy=None, threshold_num=130):
    '''
    放置方块区域检测函数
    :param image:输入图像
    :return:轮廓区域边点,处理后的图像
    '''
    if xy!=None:
        self.xy=xy
    # 机械臂初始位置角度
    joints_init = [self.xy[0], self.xy[1], 0, 0, 90, 0] # REVISE
    # 将机械臂移动到标定方框的状态
    self.arm.Arm_serial_servo_write6_array(joints_init, 500)
    self.image = image
    self.threshold_num = threshold_num
    # 创建边点容器
    dp = []
    h, w = self.image.shape[:2]
    # print("h",h)
    # 获取轮廓点集(坐标)
    contours = self.Morphological_processing()
    # 遍历点集
    for i, c in enumerate(contours):
        # 计算轮廓区域。
        area = cv.contourArea(c)

        low_bound = None
        high_bound = None
        if self.use_other_cam:
            low_bound = h * w / 2 - 40000
        else:
            low_bound = h * w / 2
        high_bound = h * w

        # 设置轮廓区域范围
        if low_bound < area < high_bound:
            # 计算多边形的矩
            mm = cv.moments(c)
            if mm['m00'] == 0:
                continue
            cx = mm['m10'] / mm['m00']
            cy = mm['m01'] / mm['m00']
            # 绘制轮廓区域
            cv.drawContours(self.image, contours, i, (255, 255, 0), 2)
            # 获取轮廓区域边点
```

```
            dp = np.squeeze(cv.approxPolyDP(c, 100, True))
            # 绘制中心
            if not self.data_collect:
                cv.circle(self.image, (np.int_(cx), np.int_(cy)), 5, (0, 0, 255), -1)
    return dp, self.image

def Morphological_processing(self):
    '''
    形态学及去噪处理,并获取轮廓点集
    '''
    # 将图像转为灰度图
    gray = cv.cvtColor(self.image, cv.COLOR_BGR2GRAY)
    # 使用高斯滤镜模糊图像。
    gray = cv.GaussianBlur(gray, (5, 5), 1)
    # 图像二值化操作
    ref, threshold = cv.threshold(gray, self.threshold_num, 255, cv.THRESH_BINARY)
    # 获取不同形状的结构元素
    kernel = np.ones((3, 3), np.uint8)
    # 形态学开操作
    if self.use_other_cam:
        blur = cv.morphologyEx(threshold, cv.MORPH_OPEN, kernel, iterations=8)
    else:
        blur = cv.morphologyEx(threshold, cv.MORPH_OPEN, kernel, iterations=4)
    # 提取模式
    mode = cv.RETR_EXTERNAL
    # 提取方法
    method = cv.CHAIN_APPROX_NONE
    # 获取轮廓点集(坐标) python2和python3在此处略有不同
    # 层级关系 参数一：输入的二值图，参数二：提取模式，参数三：提取方法。
    contours, hierarchy = cv.findContours(blur, mode, method)
    return contours

def Perspective_transform(self, dp, image):
    '''
    透视变换
    :param dp: 方框边点(左上,左下,右下,右上)
    :param image: 原始图像
    :return: 透视变换后图像
    '''
    if len(dp) != 4:
        return
    upper_left = []
    lower_left = []
    lower_right = []
    upper_right = []
    for i in range(len(dp)):
        if dp[i][0]<320 and dp[i][1]<240:
            upper_left=dp[i]
        if dp[i][0]<320 and dp[i][1]>240:
            lower_left=dp[i]
        if dp[i][0]>320 and dp[i][1]>240:
            lower_right=dp[i]
        if dp[i][0]>320 and dp[i][1]<240:
            upper_right=dp[i]
    # 原图中的四个顶点
    pts1 = np.float32([upper_left, lower_left, lower_right, upper_right])
    # 变换后的四个顶点
    pts2 = np.float32([[0, 0], [0, 480], [640, 480], [640, 0]])
    # 根据四对对应点计算透视变换。
    M = cv.getPerspectiveTransform(pts1, pts2)
    # 将透视变换应用于图像。
    Transform_img = cv.warpPerspective(image, M, (640, 480))
    return Transform_img
```

“dofbot_color_stacking/scripts/garbage_grap.py”为机械臂抓取与移动堆叠的逻辑代码。

```
def move(self, joints, joints_down):
    '''
    移动过程
```



```
    :param joints: 移动到物体位置的各关节角度
    :param joints_down: 机械臂堆叠各关节角度
    '''
    # 架起角度
    joints_00 = [90, 80, 50, 50, 265, self.grap_joint]
    # 抬起
    joints_up = [135, 80, 50, 50, 265, 30]
    # 架起
    self.arm.Arm_serial_servo_write6_array(joints_00, 1000)
    sleep(1)
    # 开合夹爪
    # for i in range(5):
    #     self.arm.Arm_serial_servo_write(6, 180, 100)
    #     sleep(0.08)
    #     self.arm.Arm_serial_servo_write(6, 30, 100)
    #     sleep(0.08)
    # 松开夹爪
    self.arm.Arm_serial_servo_write(6, 0, 500)
    sleep(0.5)
    # 移动至物体位置
    self.arm.Arm_serial_servo_write6_array(joints, 1000)
    sleep(1)
    # 夹紧夹爪
    self.arm.Arm_serial_servo_write(6, self.grap_joint, 500)
    sleep(0.5)
    # 架起
    self.arm.Arm_serial_servo_write6_array(joints_00, 1000)
    sleep(1)
    # 移动至对应位置上方
    self.arm.Arm_serial_servo_write(1, joints_down[0], 1000)
    sleep(1)
    # 移动至目标位置
    self.arm.Arm_serial_servo_write6_array(joints_down, 1000)
    sleep(1.5)
    # 松开夹爪
    self.arm.Arm_serial_servo_write(6, 0, 500)
    sleep(0.5)
    # 抬起
    self.arm.Arm_serial_servo_write6_array(joints_up, 1000)
    sleep(1)

def arm_run(self, move_num, joints):
    '''
    机械臂移动函数
    :param move_num: 抓取次数
    :param joints: 反解求得的各关节角度
    '''
    # a=[132, 50, 20, 60, 265, 100]
    # b=[132, 55, 38, 38, 265, 100]
    # c=[132, 60, 45, 30, 265, 100]
    # d=[132, 65, 55, 20, 265, 100]
    if move_num == '1' and self.move_status == True:
        # 此处设置,需执行完本次操作,才能向下运行
        self.move_status = False  # 将设置机械臂的状态,以保证运行完此过程在执行其它
        # print(joints[0], joints[1], joints[2], joints[3], joints[4])
        # 获得目标关节角
        joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
        # 移动到目标前的姿态
        joints_down = [135, 50, 20, 60, 265, self.grap_joint]
        # 调用移动函数
        self.move(joints, joints_down)
        # 执行完此过程
        self.move_status = True
    if move_num == '2' and self.move_status == True:
        self.move_status = False
        # print joints[0], joints[1], joints[2], joints[3], joints[4]
        joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
        joints_down = [135, 55, 38, 38, 265, self.grap_joint]
        self.move(joints, joints_down)
```



```
            self.move_status = True
        if move_num == '3' and self.move_status == True:
            self.move_status = False
            # print joints[0], joints[1], joints[2], joints[3], joints[4]
            joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
            joints_down = [135, 60, 45, 30, 265, self.grap_joint]
            self.move(joints, joints_down)
            self.move_status = True
        if move_num == '4' and self.move_status == True:
            self.move_status = False
            # print joints[0], joints[1], joints[2], joints[3], joints[4]
            joints = [joints[0], joints[1], joints[2], joints[3], 265, 30]
            joints_down = [135, 65, 55, 20, 265, self.grap_joint]
            self.move(joints, joints_down)
            self.move_status = True
```

4.6 机械臂控制逻辑入口

“dofbot_color_stacking/scripts/main.py”与“dofbot_garbage_yolov5/dofbot_garbage_yolov5/main.py”均为机械臂的运行控制入口，启动后调用各个功能包模块代码，实现机械臂的分拣与堆叠。核心代码片段示例如下。

```
# 当摄像头正常打开的情况下循环执行
while capture.isOpened():

    # 读取相机的每一帧
    ret, img = capture.read()
    print("read image from camera successfully:", ret)
    # 统一图像大小
    img = cv.resize(img, (640, 480))

    dp = np.fromfile(dp_cfg_path, dtype=np.int32)
    if DP_PRINT:
        print("dp has dtype:", dp.dtype)
        print("dp has shape:", dp.shape)
    dp = dp.reshape(4,2)
    if DP_PRINT:
        print("After reshape, dp has shape:", dp.shape)
        print("dp now is:", dp)

    img = calibration.Perspective_transform(dp, img)

    img, msg = target.garbage_run(img)
    print("Model is warming up at stage:", warm_up_count)
    if warm_up_count != 0 and last_num == warm_up_count:
        last_count += 1
        if last_count > 5:
            warm_up_count = 0
            last_count = 0
    last_num = warm_up_count
```


A 故障处理

A.1 机械臂在完成相机校准后精准抓取色块仍有误差

问题描述

机械臂在完成相机校准后，进行精准抓取色块仍会产生一点距离误差。

原因分析

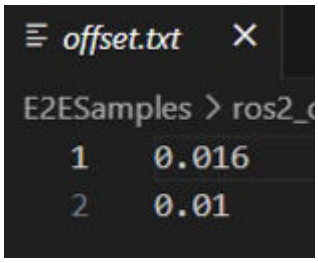
商家在生产机械臂时，难免会由于规格问题导致每台机械臂硬件误差不同。

解决方案

为了弥补硬件误差，需在程序中手工校准。并将手工校准参数作为配置文件放入config目录中。

在“/home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm/ros2_ws/src/dofbot_garbage_yolov5/dofbot_garbage_yolov5/config/offset.txt”文件中，对两个浮点型参数进行修改，如图A-1所示。

图 A-1 修改参数



步骤1 修改配置。

1. 如发现机械臂抓取略微靠后（色块后方），导致无法抓取，则需要适当加大该参数（如从0.008修正到0.01）
2. 如发现机械臂抓取略靠前（色块前方），导致无法抓取，则需要适当减少该参数（如从0.008修正到0.006）

步骤2 将该配置目录下的配置文件XYT_config.txt、dp.bin、offset.txt复制粘贴到如下目录“/home/HwHiAiUser/E2ESamples/src/E2E-Sample/ros2_robot_arm/ros2_ws/src/robot_arm_color_stacking/robot_arm_color_stacking/config”中。使得下一步骤的堆叠功能，共享当前的配置

----结束

