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

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

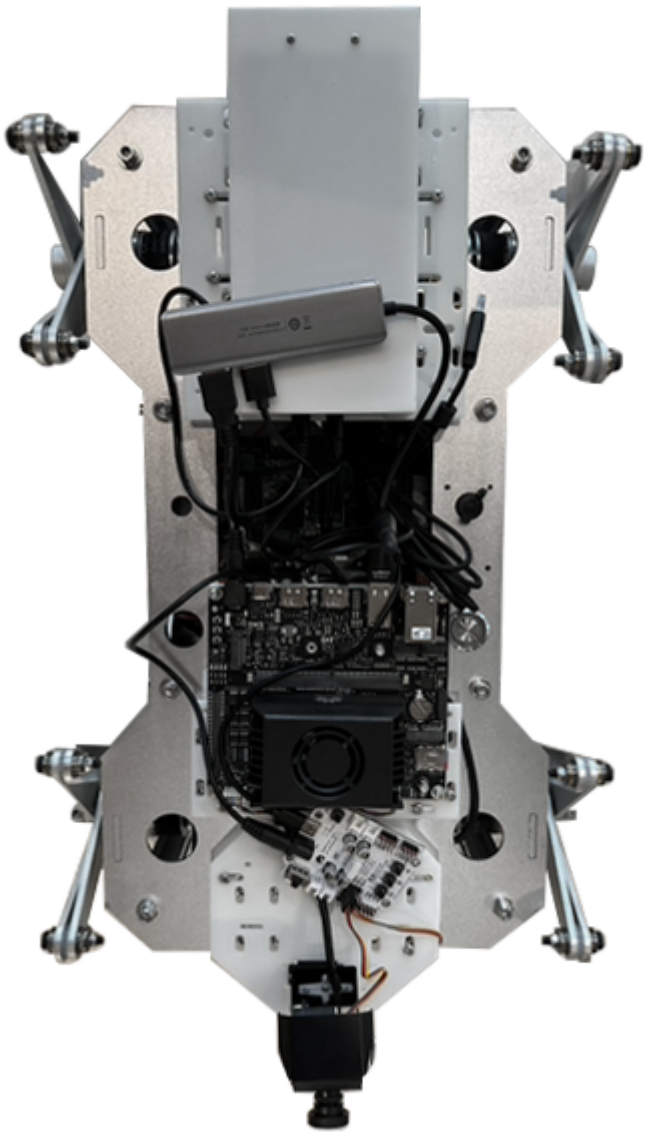
基于Atlas 200I DK A2开发者套件板（以下简称为开发板）的算力基础与接口丰富度，在实现Chatbot/智能小车/机械臂/语音台灯等样例之后，希望能够扩展兼容性和多领域适配度，实现另外一个复杂样例并且能够兼顾更加繁琐的下位机操作，所以基于开发板开发了本机械狗样例，能够实现站立坐下以及左右的姿态变化，能够自动循迹行走，另外能够追踪目标追随遛狗，并且能够在多人的情境下，仍然稳定追踪锁定的目标。

1.1 外观结构

机械狗运动部分是由以下部分组成：

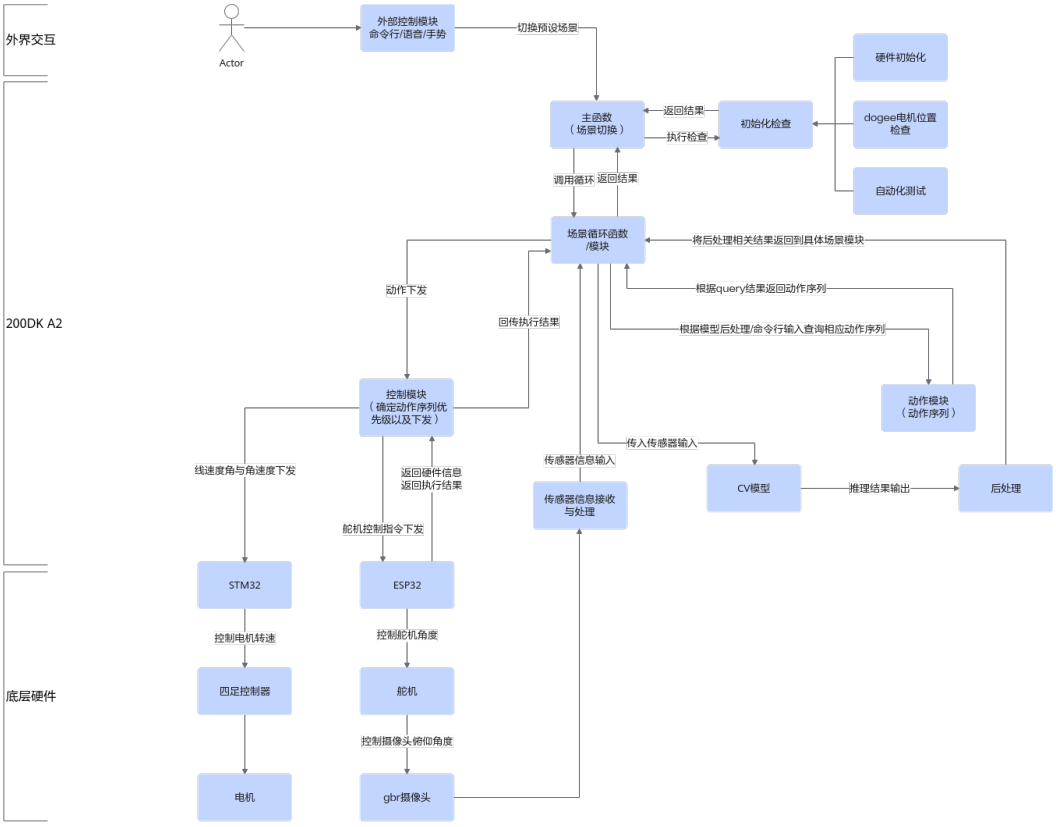
1. 四个具有2自由度的机械腿以及中间的主体部分实现的，机械腿的曲柄连杆结构保证了机械狗可以单独操作每一条腿的运动，每条腿上的两个关节由两个行星轮减速电机驱动，并放置在中间的主体部分。
2. 下位机由带有串口扩展板的两个STM32单片机开发板构成，分为主控F405和从控F103，位于机械狗身体内部。
3. 使用补充的3D连接固定件固定摄像头云台、开发板以及开发板电源
4. 摄像头云台连接一块ESP32下位机控制摄像头的移动，再通过USB扩展坞将所有的线连接到开发板上，再使用Wifi模块连接到开发板上。

图 1-1 机械狗外观结构图



1.2 功能与原理介绍

图 1-2 功能原理图

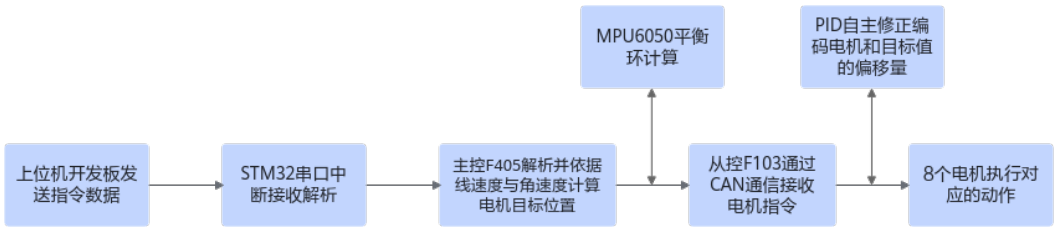


机械狗分为上位机即Atlas 200I DK A2开发者套件板，下位机即STM32和ESP32单片机，通过命令行以及手势语音等方式输入到机械狗上的各类传感器以及上位机上，在场景切换后进入到主函数，初始化各类硬件底层设备完成之后，就可以进入到各个场景对应的模块中进行信息拉取和推理进程了。以摄像头的为例，利用摄像头拉流到视频信息之后，输入到场景循环函数中，经过CV模型得到推理结果，再将后处理的结果回传到场景循环模块中，针对控制模块做动作下发，传递到STM32下位机上控制四足的运动，即可完成一个简单的流程。

1.3 控制与运动部分

机械狗的运动控制部分的实现原理图如图1-3所示：

图 1-3 机械狗运动控制原理图



- 在上位机（开发者套件）上部署离线推理模型后，在上位机上根据推理结果生成对应的运动控制指令。指令下发到STM32单片机上，由串口中断函数来进行串口数据传输的接收。在上位机端和STM32使用相同的校验码的生成和校验方式，如果串口处接收到是上位机发出的指令，则会进入到对应的中断函数中进行运动状态的改变和操作。
- 在下位机（STM32）的初始化过程中：
  a. 会开启多个任务序列，包括初始化电机零点位置、陀螺仪初始化、显示屏显示、LED灯显示、蜂鸣器初始化及串口发送数据初始化中断函数标记归位等操作。
  b. 主进程在完整这些操作之后，进入获取控制端以及四足行动的部分，在收取到上位机发出的指令后，下位机进行解析，获取机器狗的姿态全局变量参数以及运动的两个方向的线速度和角速度。
  c. 进入到机械狗的运动解析部分进行运算，解析接下来的运动需中每一个机械腿的运动位置，以及八个电机需要转到的目标位置，并下发指令到从控的STM32上，通过串口下发到电机处。其中的电机使用PID来编码，在整个姿态变化和运动的过程中，通过PID控制电机的目标位置和实际运动位置误差以及自我纠正误差。
  d. 同时，为了防止运动过程中机械狗摔倒，加入了陀螺仪的运算，且可以在显示屏上显示实际数据。MPU6050陀螺仪会计算平衡环的数值大小用来修正不同方向上的非水平位置变化，然后加入到机械腿的运动控制中。得到了控制机械腿的目标位置后，可以通过机械腿路径规划算法实现实时计算机械腿机械腿的运动位置，使机械狗能够稳步前进以及完成其他动作。

图 1-4 运动控制时序图 1

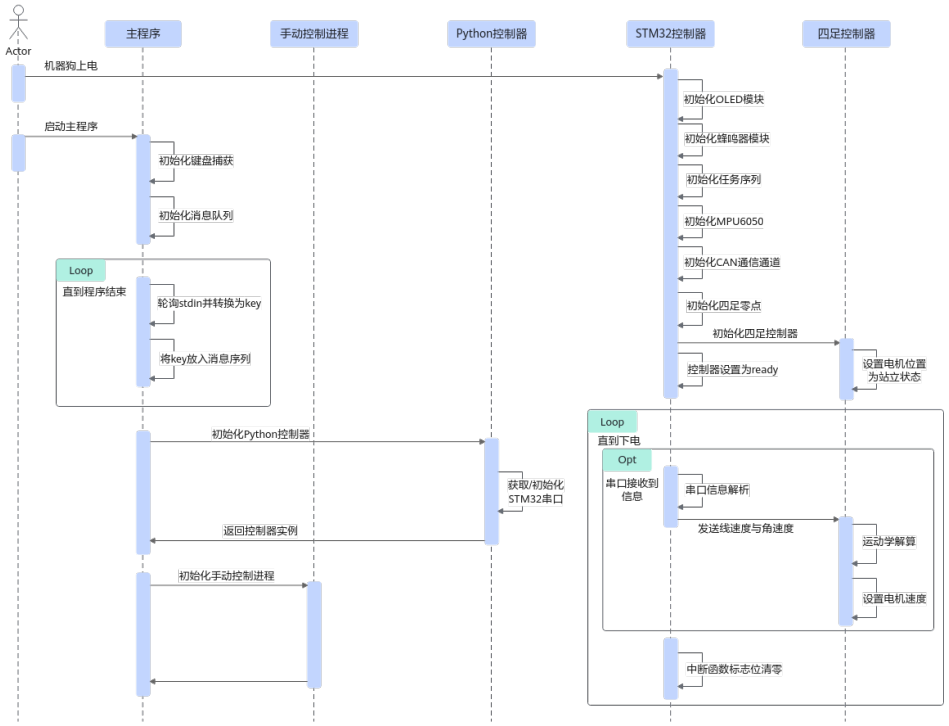
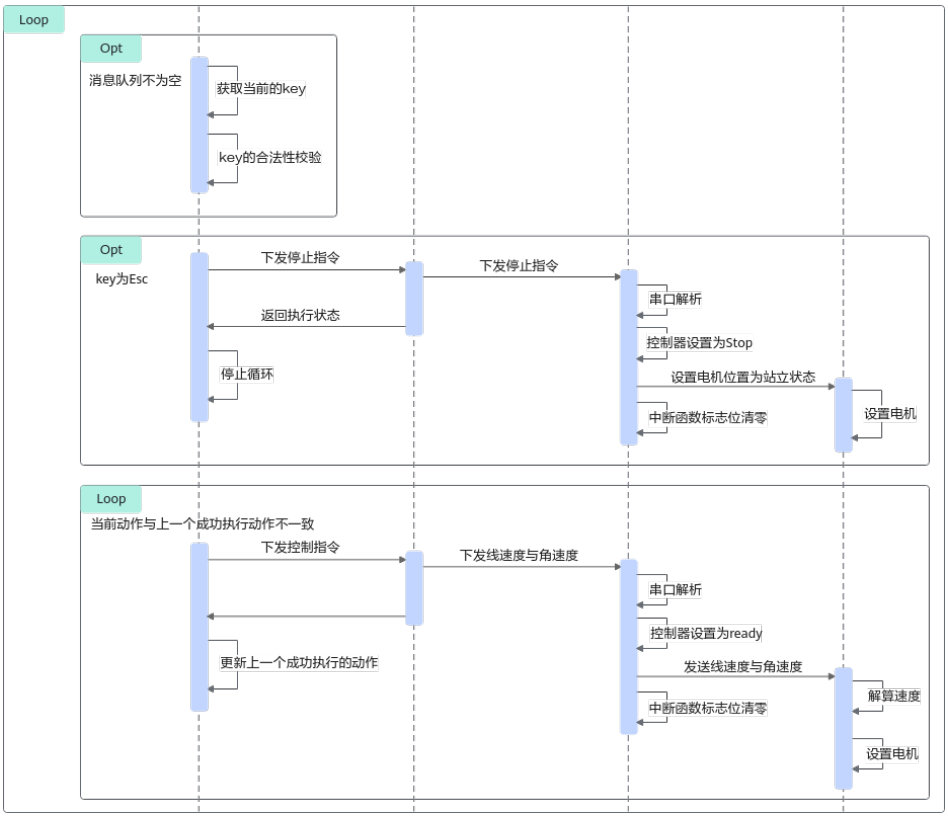


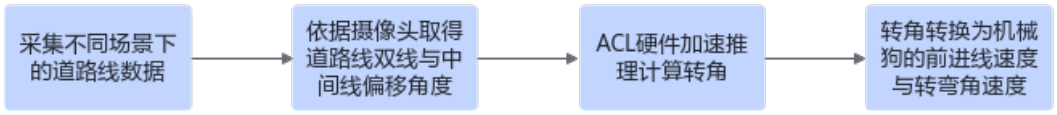
图 1-5 运动控制时序图 2



1.4 循迹行走部分

机械狗的循迹行走部分的原理图如图1-6所示：

图 1-6 机械狗循迹行走原理图



此部分功能的实现原理与智能车相似均基于偏移纠正模型和转向辅助模型。

- 偏移纠正模型：通过回归计算左右两条引导线，计算出两条引导线的中线，并与摄像头的底部求交点以及偏转角。理论角度偏转角为90°时说明机械狗在引导线内直线行走，不会走出引导线。若偏转角是钝角或者锐角，则说明机械狗需要依据实际的角度进行左转和右转的修正，如此循环就能够使机械狗在引导线内行走。
- 转向辅助模型：识别转弯标识以及调头标识。为了避免摄像头抖动导致的问题，判断的条件为：相邻的几帧均识别到相同的转弯标识才进行对应的动作。

通过两个模型的交替进行，再针对转向角和机械狗的速度做映射和微调，即可实现机械狗的循迹行走功能。

1.5 锁定追踪部分

机械狗的锁定追踪部分的时序图如图1-7、图1-8和图1-9所示：

图 1-7 机械狗锁定追踪时序图 1

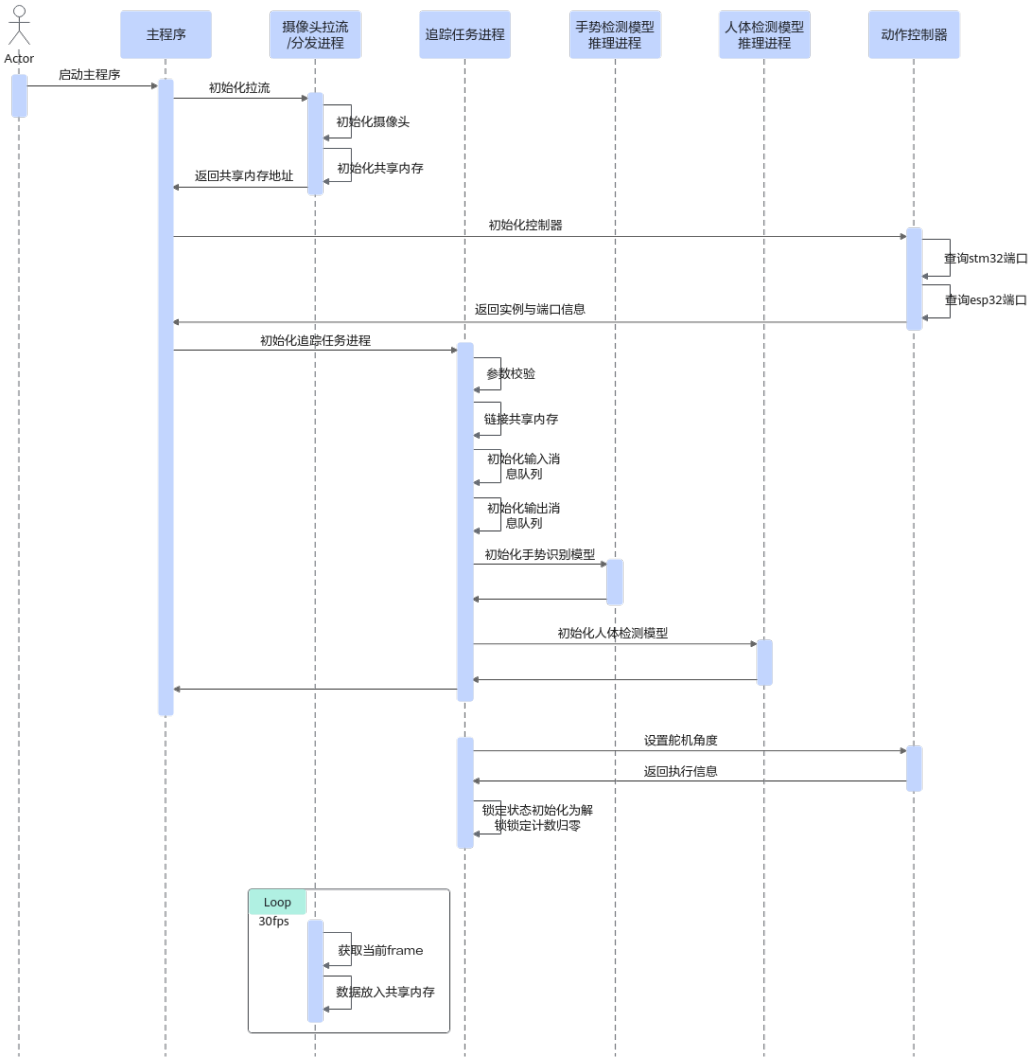


图 1-8 机械狗锁定追踪时序图 2

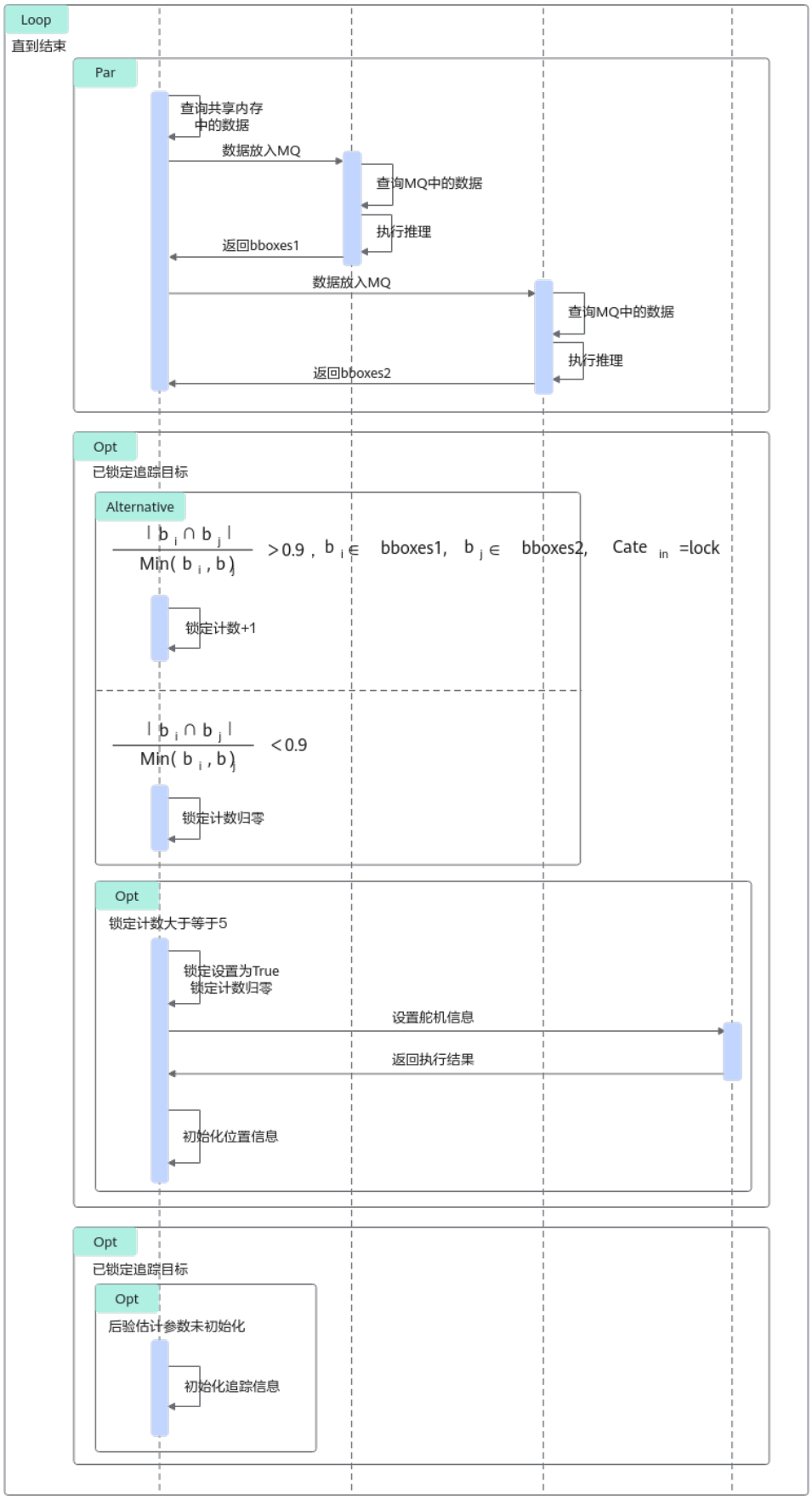
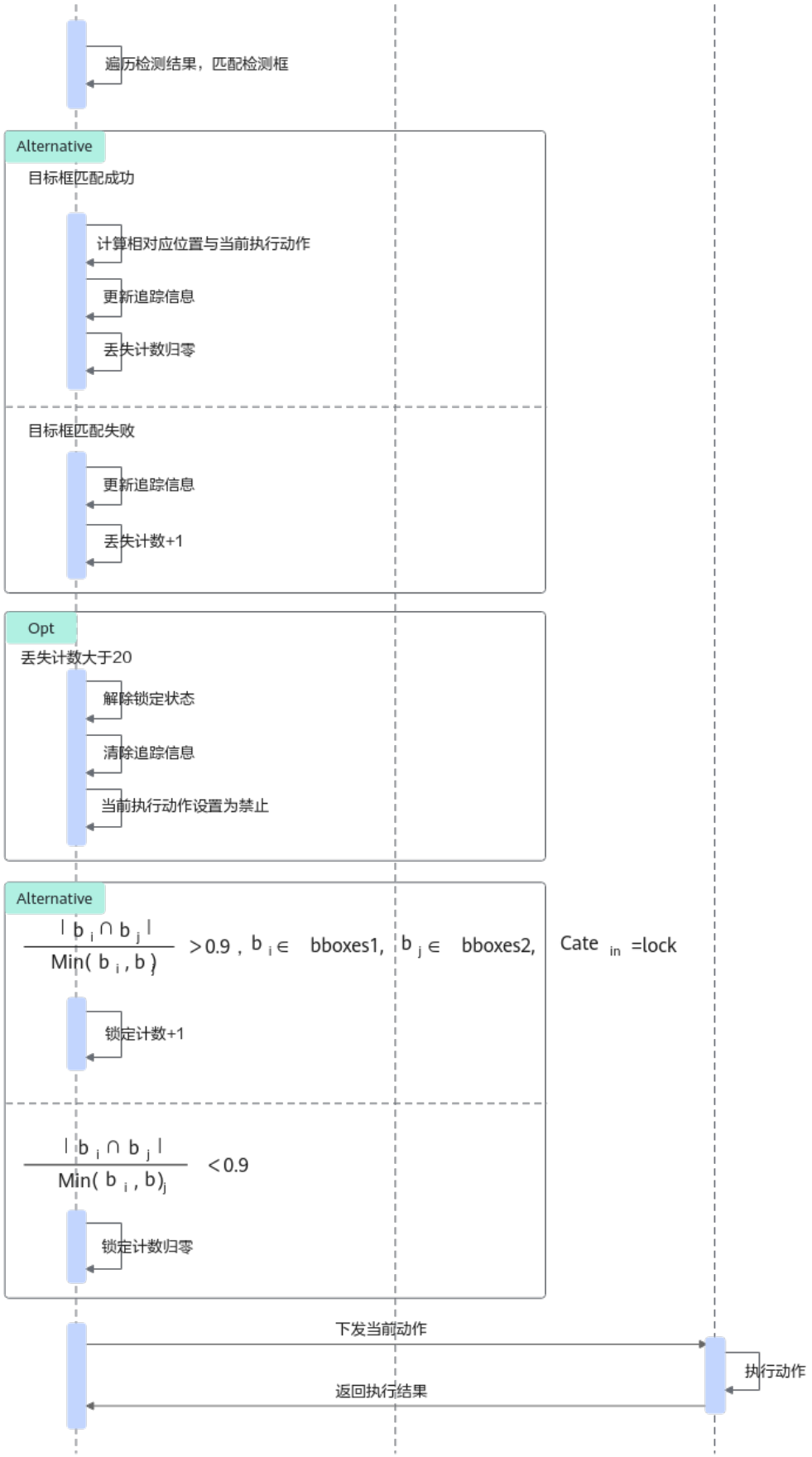


图 1-9 机械狗锁定追踪时序图 3



1. 在启动主程序后，首先进行摄像头和共享内存的初始化，拉流后在运动控制器端初始化，获取STM32和ESP32的端口信息。
2. 其中，外设的连接端口号ttyUSB*中具体的号码是由拔插的先后顺序决定的，所以需要提前查询端口信息并返回到主程序中。而后在追踪任务的进程中初始化，再开启两个手势识别和人体检测的模型进程，分别进行初始化，然后在控制端下位机设置舵机的角度，调整到方便机械狗追踪的角度，将识别锁定追踪目标的状态初始化为解锁，锁定计数归零。
3. 在拉流分发的进程中开启循环获取frame并放入共享内存中，然后再启动循环并开始并行的手势识别和人体识别的推理进程。此处使用两个消息队列来防止手势识别和人体识别出现异帧不同步的情况，将两个过程同步之后返回两个bboxes，分别是手势和人体的识别框存储数据，再利用如下公式来判断锁定的目标框：
   a. 在出现比值大于0.9的帧时，计数器值加1，在连续五帧都超过这个比值后，将锁定目标的FLAG置为True，然后设置舵机的信息，进行目标的锁定追踪；若在已经锁定目标的状态下就需要锁定到该框并且使用卡尔曼滤波的方式进行目标框的匹配，进而推算出要追随的目标，即使在有多人存在的场景仍然可以准确追踪到目标。
   b. 在超过20帧没有识别到目标之后以及出现解锁的手势超过5帧之后就会将机械狗的下位机中的运动FLAG设定为停止，然后上位机上的解锁标志回归为解锁，并且下发动作，然后等待下一个锁定的手势出现，继续追踪新的目标。

2 样例组装

2.1 准备组件

机械狗所需配件如**表2-1**所示：

须知

当前样例仅在Ubuntu OS适配验证过，未在openEuler OS适配验证，推荐烧录镜像时，烧录Ubuntu OS镜像。

表 2-1 机械狗配件表

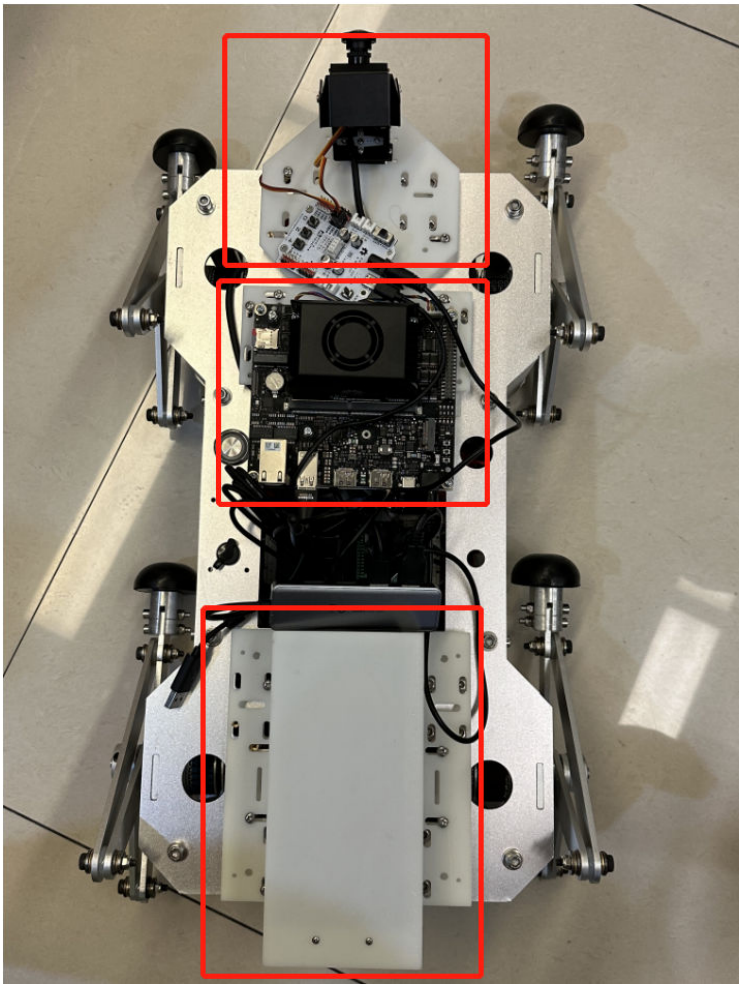
名称	数量/个	是否需要单独购买	规格
开发者套件及电源适配器	1	否	唯一
机械狗裸机	1	是	裸机
摄像头两轴云台	1	是	两轴云台精简版
广角摄像头模组	1	是	118°广角无畸变模组
ESP32机器人开发板	1	是	主板+Type-C数据线
38400mah移动电源	1	是	12V9V5V三输出
单头六角铜柱	40	是	M2.5*7+6
单头六角铜柱	40	是	M4*7+6

名称	数量/个	是否需要单独购买	规格
防松螺母	40	是	M4
防松螺母	40	是	M2.5
圆头螺母	40	是	M4*5
圆头螺母	40	是	M2.5*5
3D打印支撑板（下载文件后通过3D打印）	6	是	唯一规格
USB扩展坞	1	是	单USB转4口USB3.0
Micro USB数据传输线	1	是	一定要有数据传输功能，不可仅为充电线
USB Wifi模块	1	是	-

2.2 组装步骤

步骤1 首先将机械狗裸机平放在地面上，然后将3D打印好的三个支撑板用单头六角铜柱和防松螺母固定在机械狗的裸机上平面上，具体的位置如**图1 安装固定**所示：

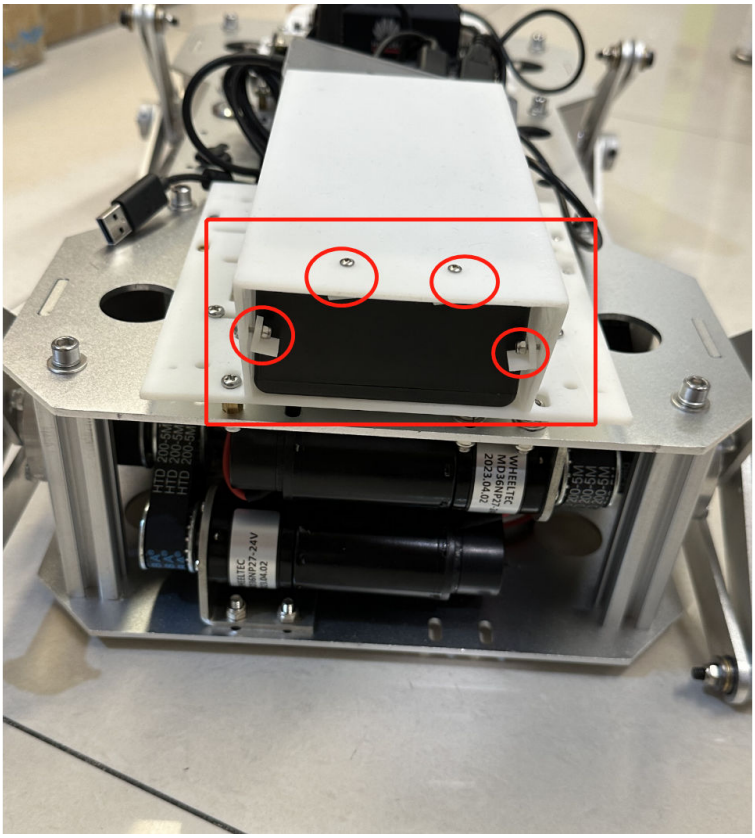
图 2-1 安装固定



从上到下依次为摄像头承载板、开发板承载板以及电池仓，按照图中的螺丝与单头螺柱位置固定即可。

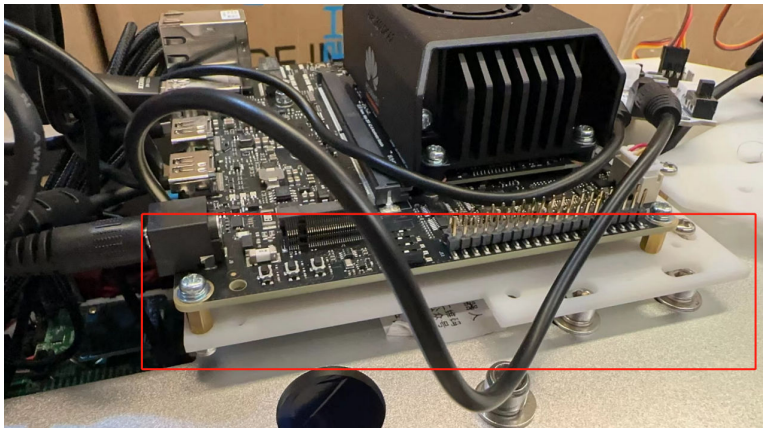
步骤2 将电源放入到电池仓中，使用电池挡板固定，如图2-2所示：

图 2-2 安装电源



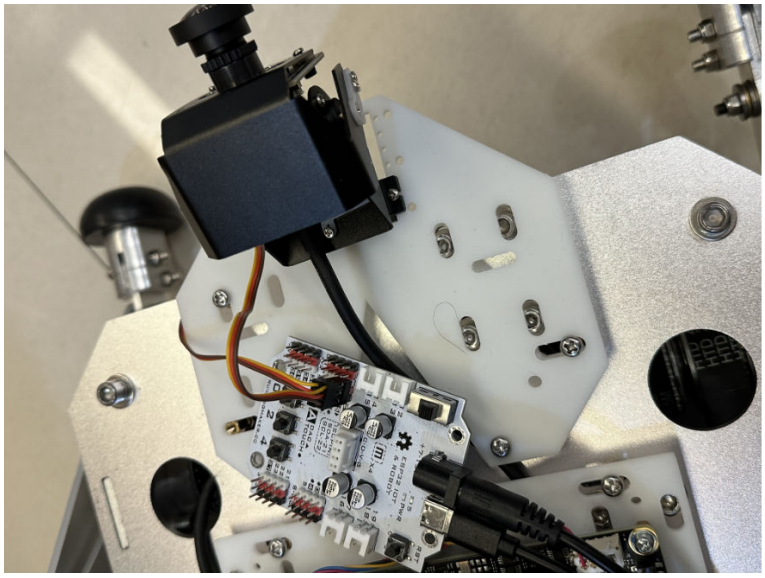
步骤3 将开发者套件从底板上拆下并固定到3D打印出来的开发板承载板上，如**图3 固定开发板**所示：

图 2-3 固定开发板



步骤4 将两轴云台上原有的普通摄像头拆下，替换上无畸变的广角摄像头模组，再将云台安装到摄像头承载板上，如**图4 安装广角摄像头模组**所示：

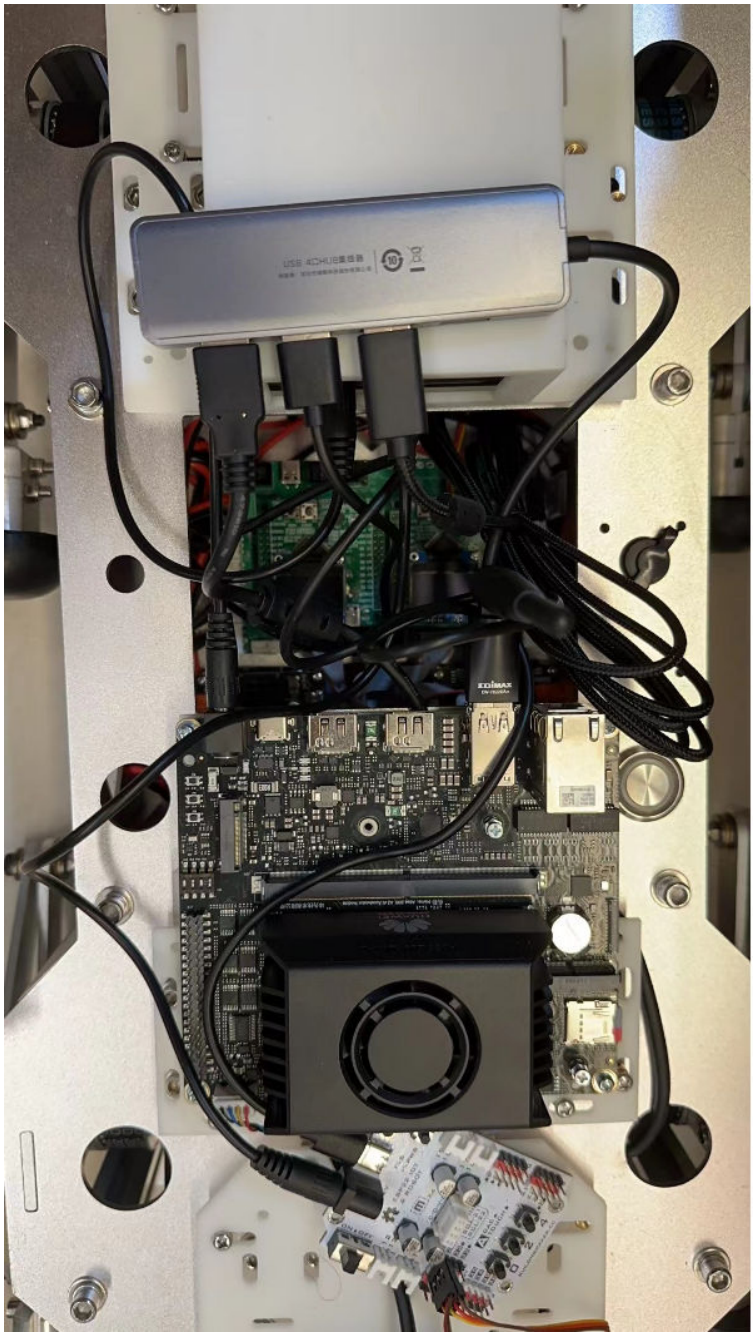
图 2-4 安装广角摄像头模组



其中上半部分的舵机连接到ESP32上的26号接口，棕色线接地靠近5V一侧，下半部分的舵机连接到ESP32上的25号接口，棕色线接地靠近5V一侧。

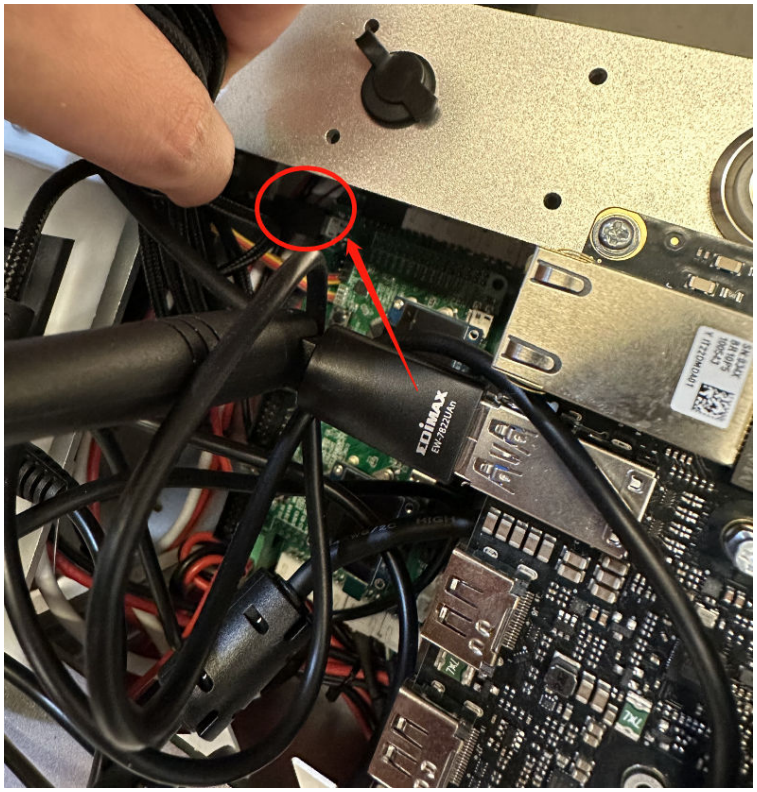
步骤5 总体的接线方式如图2-5所示：

图 2-5 接线图



步骤6 将各个电源线连接好后，需要使用一根Micro USB数据线连接到机械狗内部的主控F405一侧的MicroUSB端口，另一端连接到插到开发板上的USB扩展坞上，再将USBWifi模块插到开发板的USB接口上，将USB摄像头和ESP32接到USB扩展坞上，即可完成接线。

图 2-6 接线图



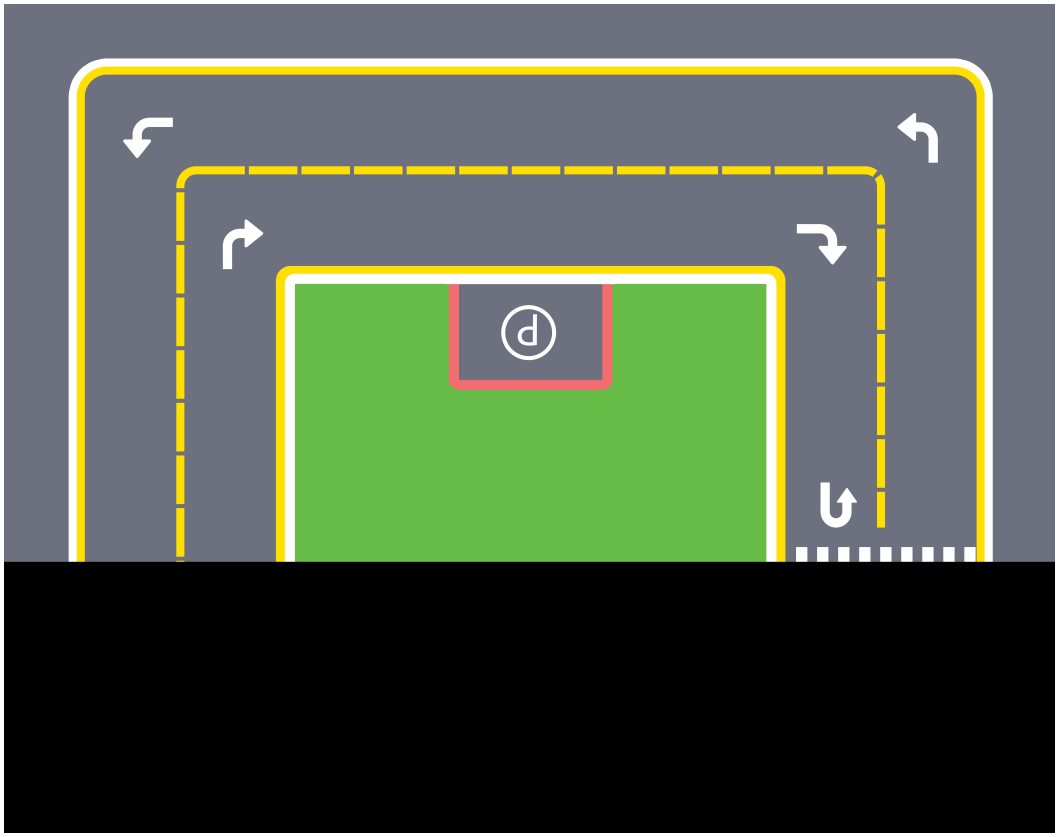
----结束

3 运行环境准备

3.1 地图绘制

机械狗的自动行走依赖于模型的训练，因此需要设计一张地图，用于自动行走的模型训练过程和推理结果验证。此处提供地图设计方案，开发者可以直接使用此地图，也可以自行设计其他版本的地图。

图 3-1 样例地图



须知

地图规格

- 道路宽度：60cm。
- 完整地图的尺寸：约3m * 4m。
- 停车位：30cm * 40cm。

制作建议

- 使用Photoshop等画图软件制作地图。
- 建议使用表面不太光滑的材质，如果地图表面过于光滑则容易使机械狗腿部打滑。另外光滑的表面会产生强烈的反光，影响机械狗通过摄像头采集的画面质量，进而干扰图像识别的任务。

3.2 配置无线 Wifi 模块

步骤1 确保路由器和开发者套件已按**步骤6**中所述连接。

步骤2 参见**登录开发者套件**选择任意一种登录方式使用**root**用户（默认密码为**Mind@123**）远程登录开发者套件。

步骤3 参见**通过USB WiFi网卡联网**配置USB WiFi设置。

----结束

3.3 配置 ESP32 开发板烧录软件

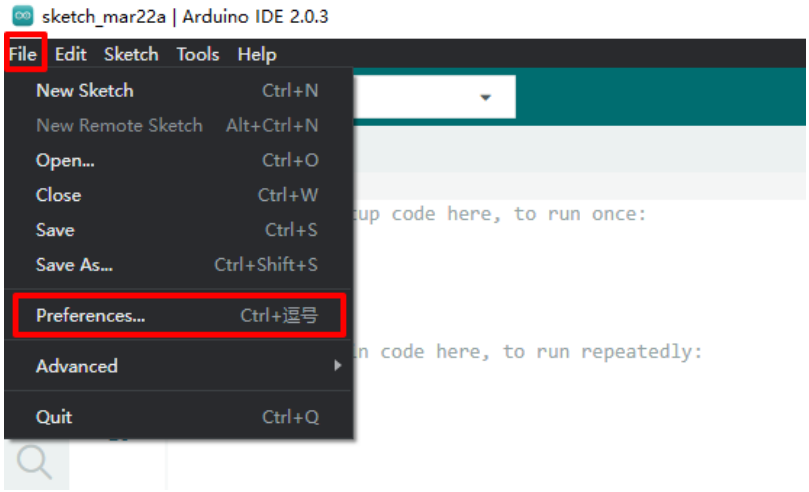
Arduino是一套便捷、灵活、容易上手的硬件开发平台，它包括多种型号的Arduino控制电路板和专用编程开发软件，能帮助用户快速的开发出智能硬件原型。

步骤1 进入**Arduino官网**下载程序安装包“arduino-ide_*version*_Windows_64bit.exe”，并按照默认配置安装。

步骤2 双击打开软件，修改语言为中文。

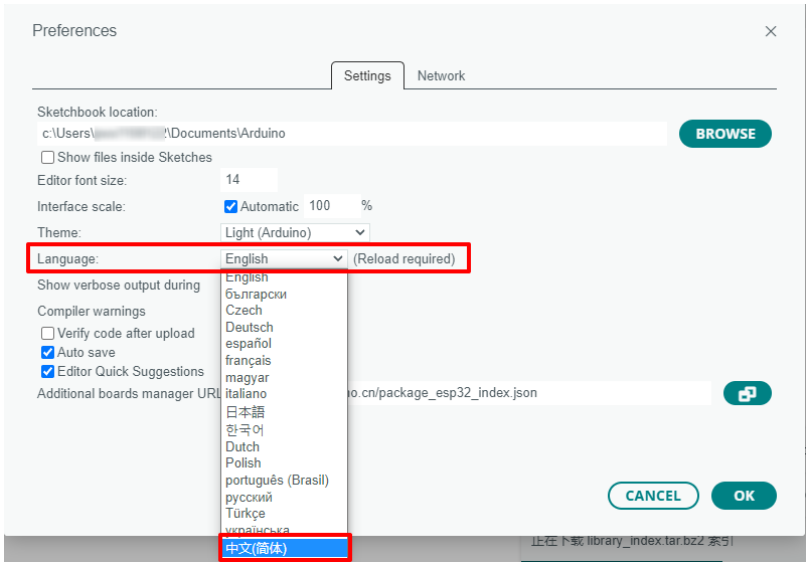
1. 单击“File > Preferences”，如**图3-2**所示。

图 3-2 软件菜单



2. 单击“Language”下拉菜单，选择“中文(简体)”，如**图3-3**所示。单击OK保存后工具会自动重启，重启后为中文界面。

图 3-3 修改语言



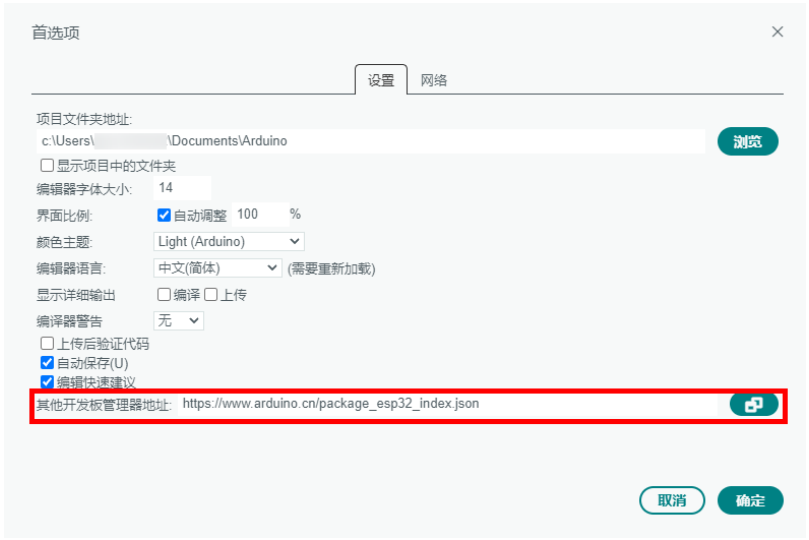
步骤3 安装ESP32开发板。

1. 单击“文件 > 首选项”，如图3-4所示。

图 3-4 中文软件菜单

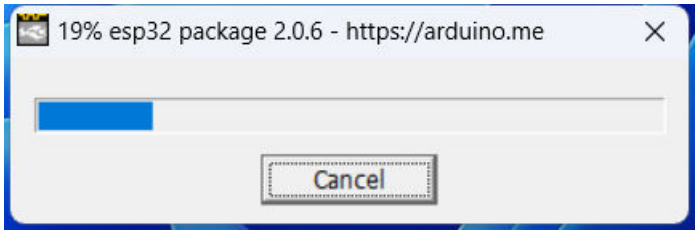
2. 在“其他开发板管理器地址”中输入“https://www.arduino.cn/package_esp32_index.json”与“https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json”或单击  图标在文本框中输入开发板管理器地址，单击确定保存，如图3-5所示。

图 3-5 添加开发板管理器地址



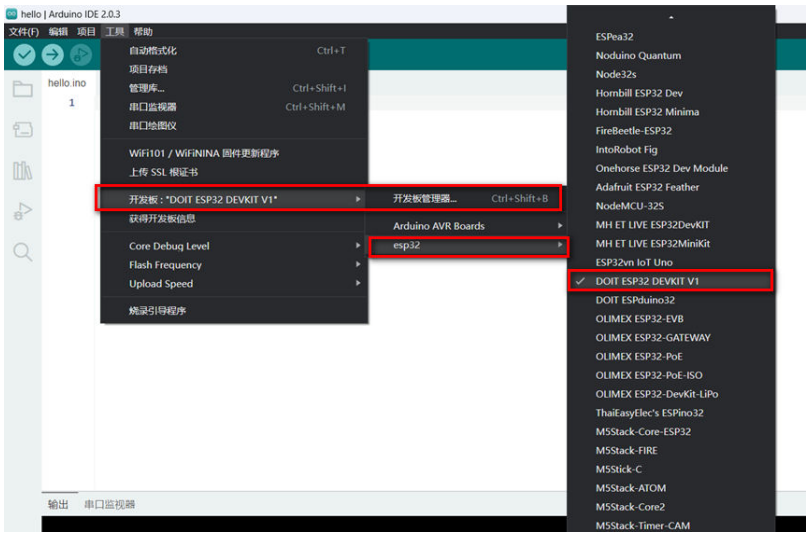
3. 离线安装ESP32开发板，单击**链接**下载Arduino的ESP32开发板安装包，下载完成后双击安装文件进行默认安装，安装完成后重启Arduino IDE。

图 3-6 通过离线包安装 ESP32 开发板



4. 在“开发板”中选择ESP32开发板，如图3-7所示。

图 3-7 选择开发板

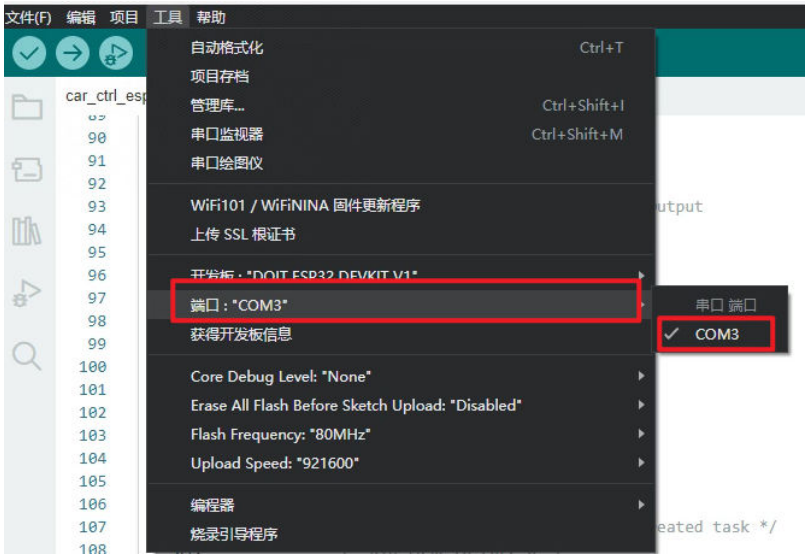


说明

此步骤中安装ESP32组件，推荐使用离线安装的方式，用户也可以自行搜索在线安装的方法，但由于国内网络等问题，在线安装过程很可能会非常慢甚至安装失败。

步骤4 用USB数据线连接PC和ESP32开发板，单击“工具 > 端口”，选择新增的COM串行端口作为Arduino与ESP32开发板传输数据的通道，如图3-8所示。

图 3-8 选择串口



说明

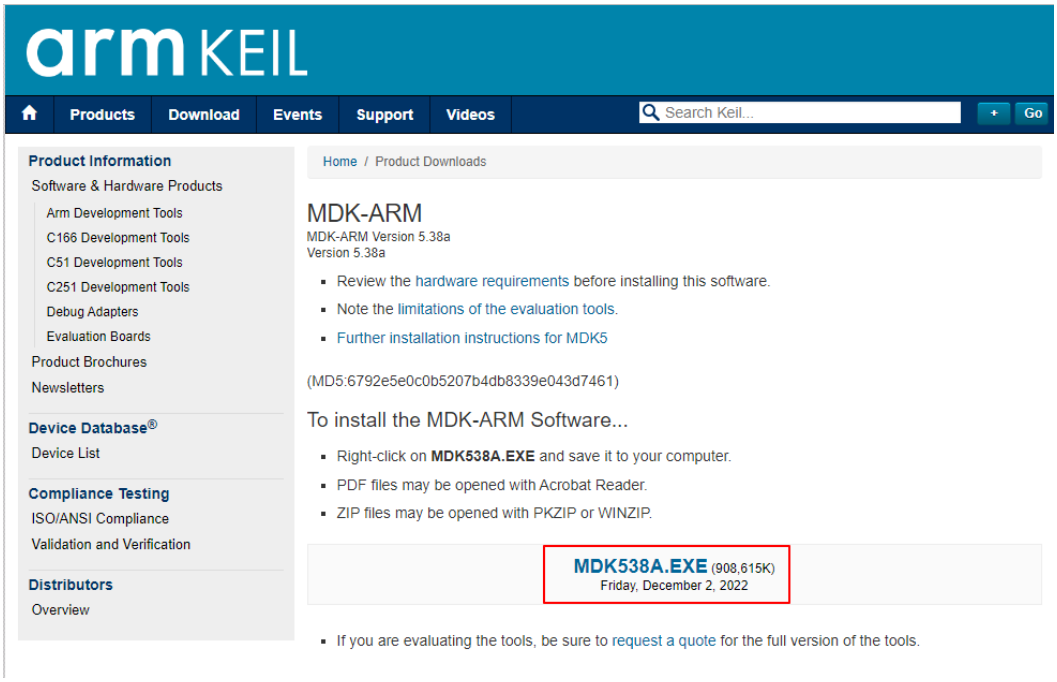
若没有出现COM串行端口，可能是由于电脑没有安装USB串口驱动，需自行下载并安装CH340驱动工具。

----结束

3.4 配置 STM32 开发板烧录软件

步骤1 请用户进入官方网站完成信息注册，单击如图3-9所示图标，下载注册机软件。

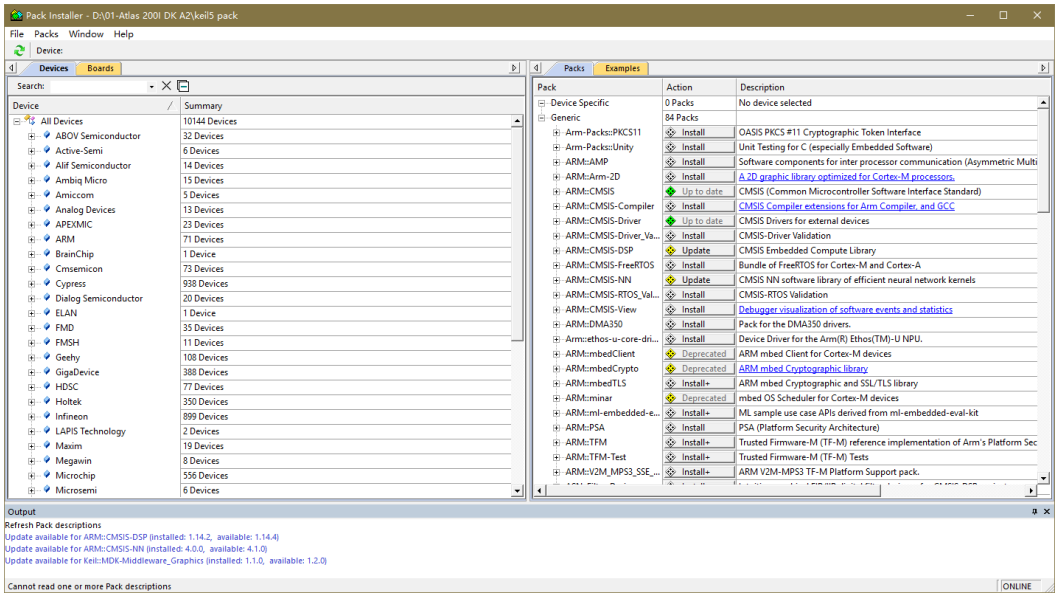
图 3-9 获取软件



步骤2 按照默认配置通过注册机安装Keil uVision5软件及工具包。

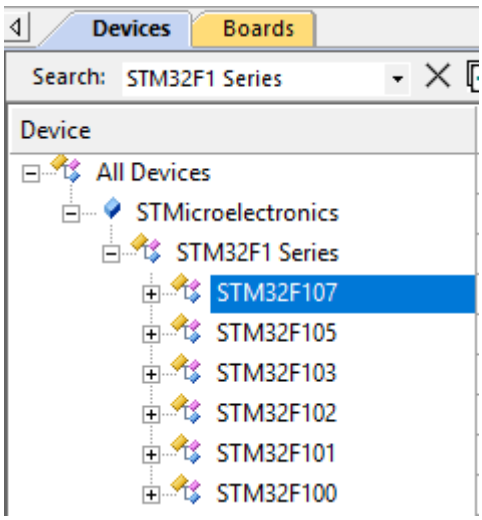
步骤3 打开Keil uVision5软件，单击工具栏  按钮，进入包管理工具。

图 3-10 包管理工具界面

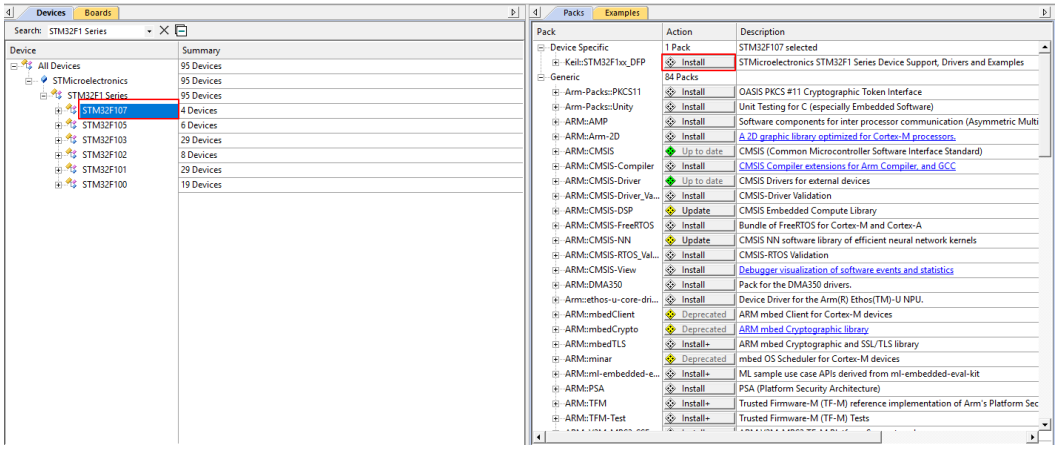


步骤4 在搜索框中搜索STM32F1 Series与STM32F4 Series并安装所有包。

图 3-11 搜索包



步骤5 左键单击待下载的包，单击右侧工具框中的“Install”按钮安装。



----结束

3.5 获取代码

3.5.1 ESP32 代码

获取代码

获取链接，单击链接下载样例代码压缩包“ascend-devkit-master.zip”到PC并解压，获得ESP32开发板代码目录“ascend-devkit-master\src\E2E-Sample\dogee\dogee_control\esp32_code”。

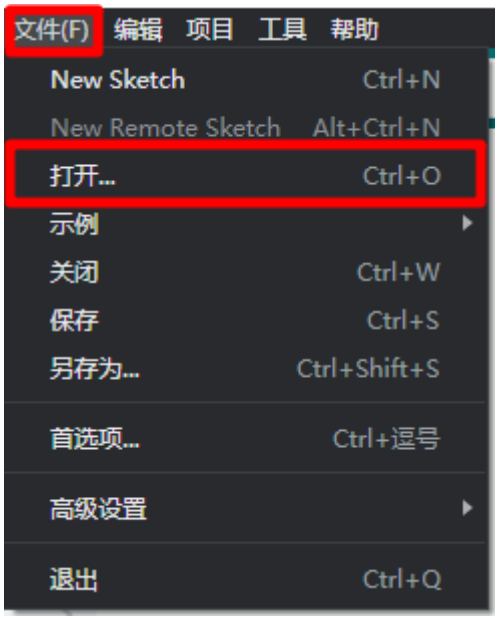
配置功能库文件

将dogee_control\esp32_code下的“ESP32_Servo.cpp”、“ESP32_Servo.h”、文件复制到Arduino库文件路径下，例如：“C:\Users\10459\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.6\cores\esp32”。

烧录代码到 ESP32 开发板

步骤1 在PC使用Arduino工具单击“文件>打开”按钮，选择“dogee_control\esp32_code\dogee_ctrl_esp32.ino”文件打开。

图 3-12 打开脚本



步骤2 在Arduino工具中单击按钮烧录控制代码至ESP32单片机，如图3-13和图3-14所示。

图 3-13 烧录代码

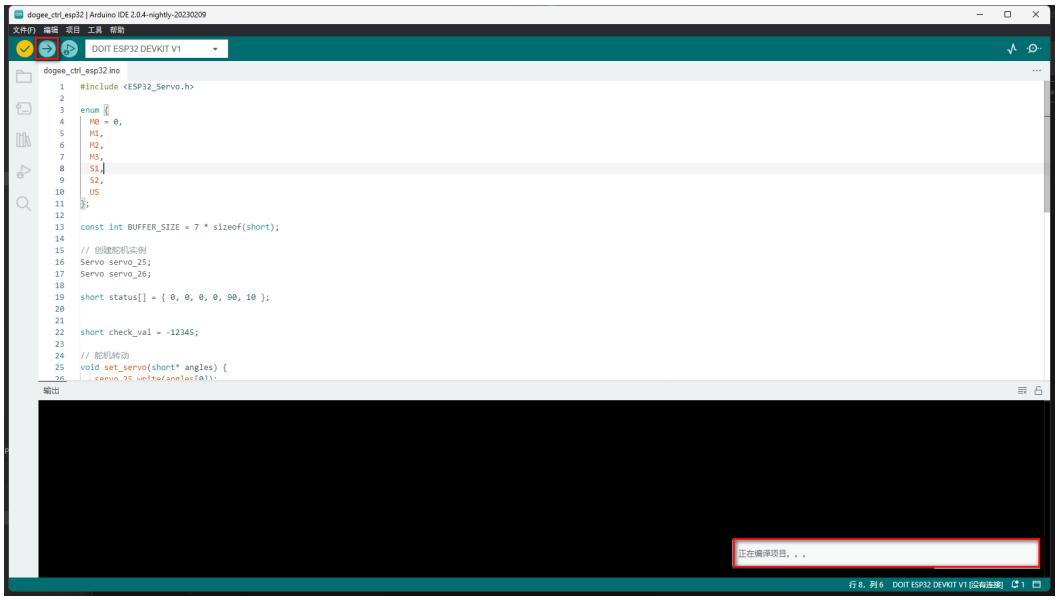
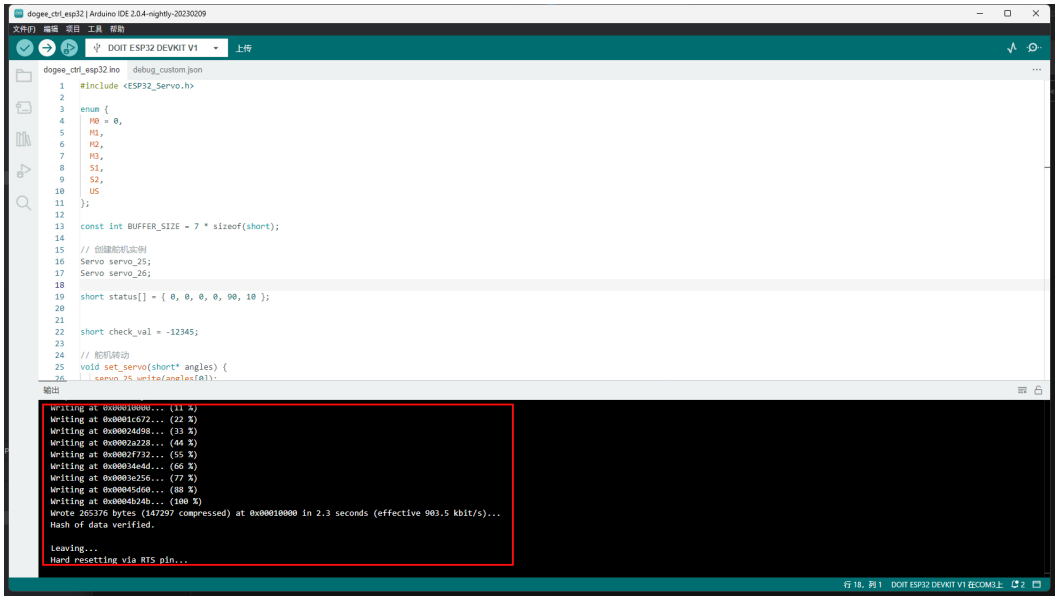


图 3-14 烧录回显



----结束

3.5.2 STM32 代码

获取代码

获取链接，单击链接下载样例代码压缩包“ascend-devkit-master.zip”到PC并解压，获得STM32开发板代码目录“ascend-devkit-master\src\E2E-Sample\dogee\dogee_control\stm32_code”将stm32_code中的代码复制到原始产品提供的代码中，选择替换对应文件。

烧录代码到 ESP32 开发板

步骤1 在PC使用Keil uVision5选择工具栏“File > Open”，打开替换文件后“\QuadrupedF405\QuadrupedF405\USER”目录下的“QuadrupedF405.uvprojx”项目文件，即可进入到对应的界面，如**图3-16**所示。

图 3-15 打开文件

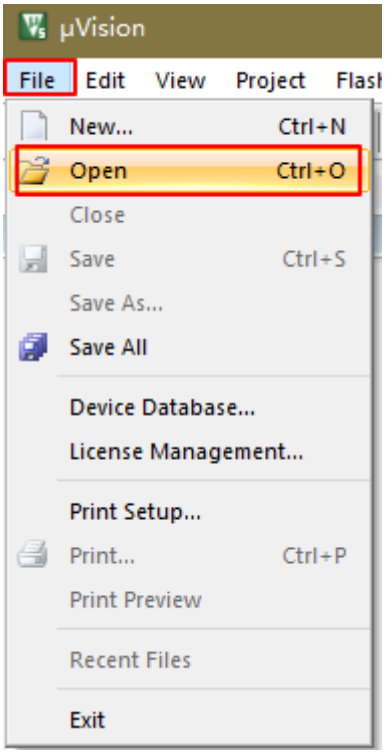
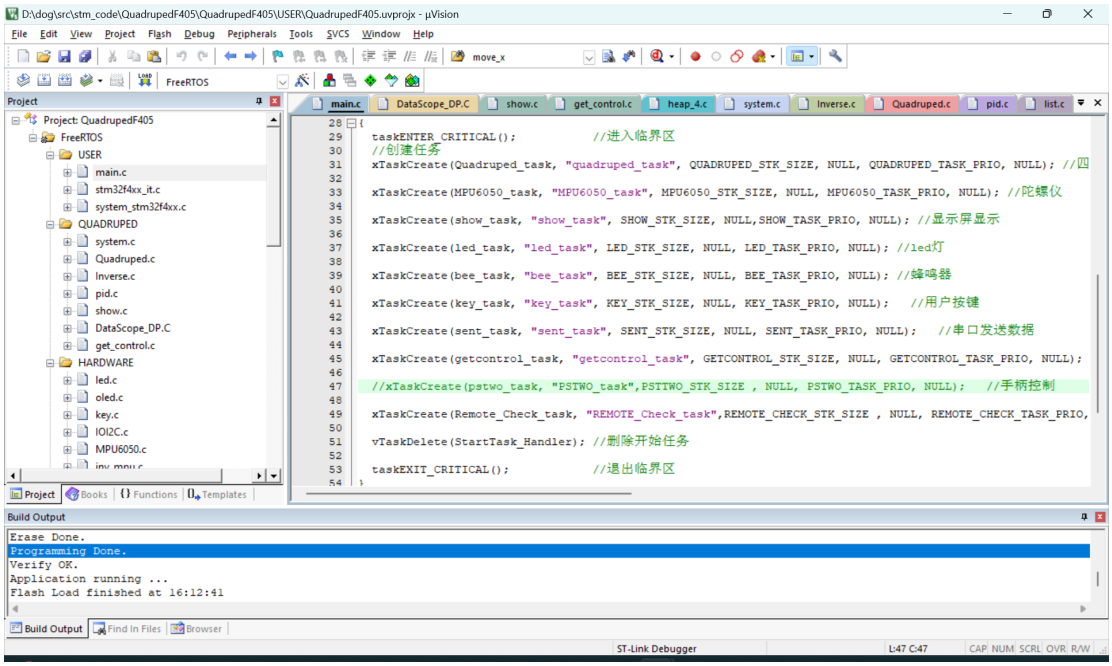
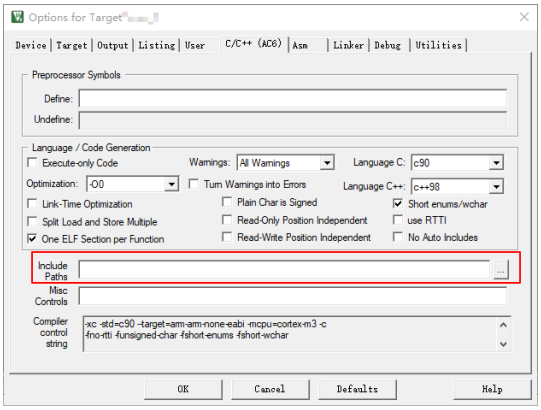


图 3-16 界面图

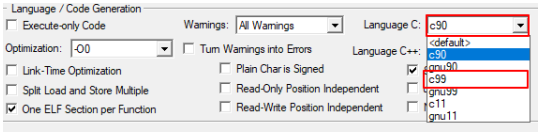


说明

- 出现缺少头文件提示时，选择工具栏“Project > Options for Target ...”，请选择C/C++，为Project添加头文件的include path路径（路径不能出现中文字符）

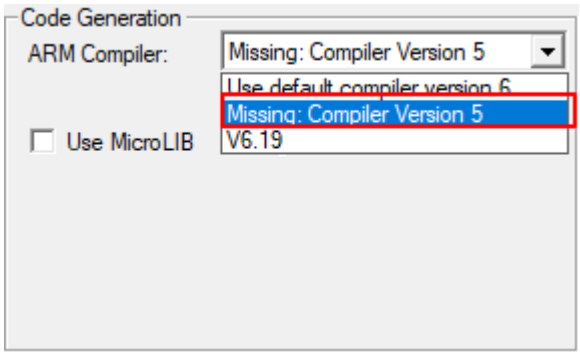


- 出现未识别include提示需要修改LanguageC从C90改为C99。



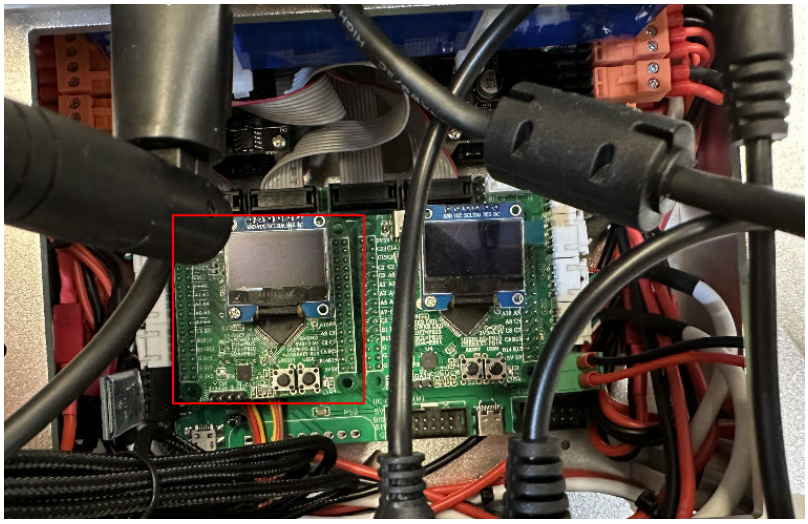
步骤2 当前Keil uVision5软件已不提供Version5的编译器，需用户自行查找并安装。

图 3-17 编译器丢失



步骤3 由于STM扩展板的限制，SWD扩展板上的外部接口未接芯片，需要将机械狗身体中的STM32F405主控板拆下，主控板位置如图3-18所示。

图 3-18 405 主控板



1. 由于STLINK的USB端串口位置固定，与STM32无法一一对应，所以需要使用杜邦线连接STLINK的USB端的串口，使其能够与开发板的串口对应，STM32串口位置如图3-19所示，STLINK的USB端串口位置如图3-20所示。

图 3-19 STM32 串口引脚

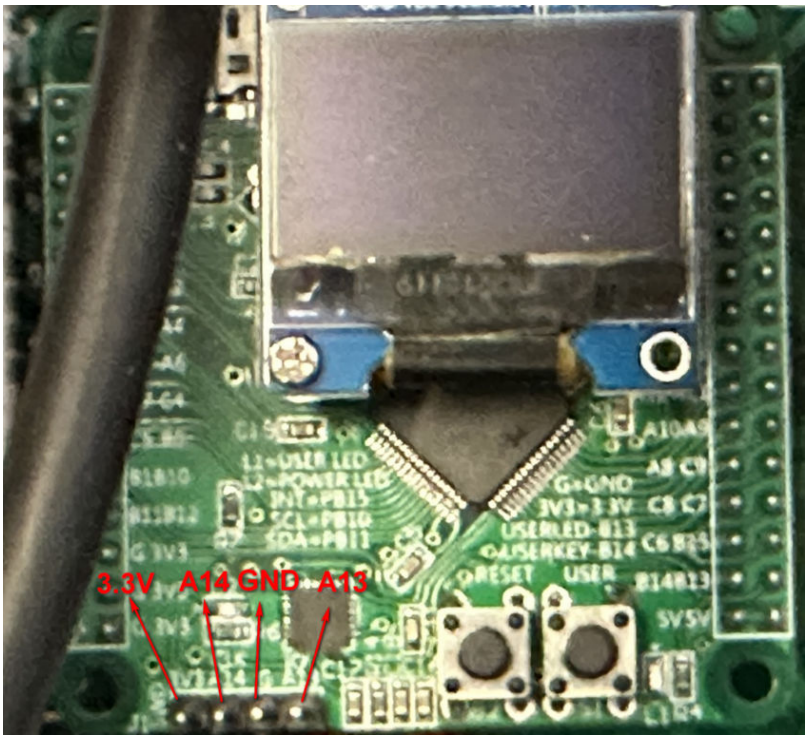
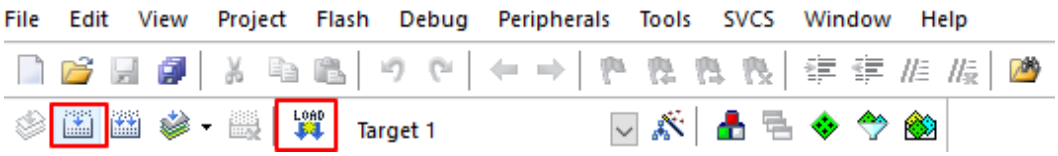


图 3-20 STLINK 的 USB 端串口位置

2. STLINK线的SWCLK串口与STM32开发板的A14连接，STLINK线的SWDIO串口与STM32开发板的A13连接，电源3.3V对应3.3V，GND串口对应GND。

步骤4 在keil客户端准备完成后，单击 编译按钮，等待代码编译无报错后，单击 烧录按钮即可将控制代码烧录到STM32的主控中。

图 3-21 界面图



----结束

3.5.3 机械狗代码

步骤1 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2 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/dogee/
```

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/dogee/”目录。

----结束

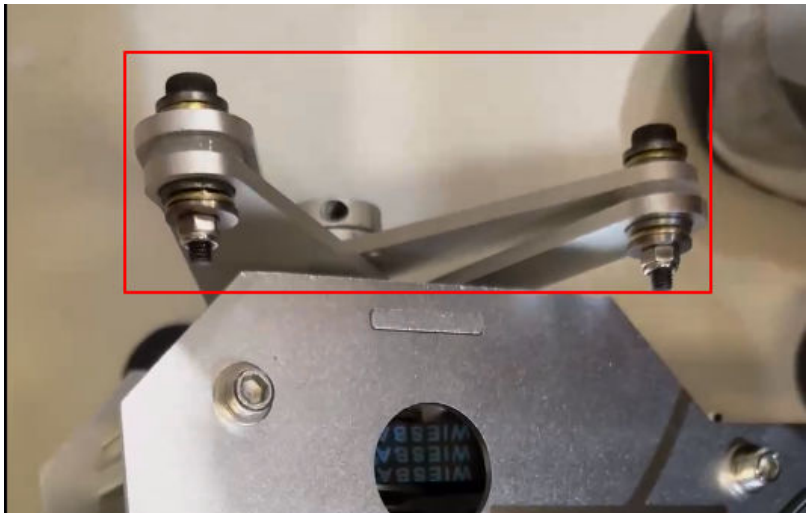
4 快速体验

4.1 手动控制机械狗姿态变换

步骤1 在机械狗上电启动之前，需要手动将机械狗的四个腿置于初始位置，便于机械狗的八个电机获取开机处的零点位置。

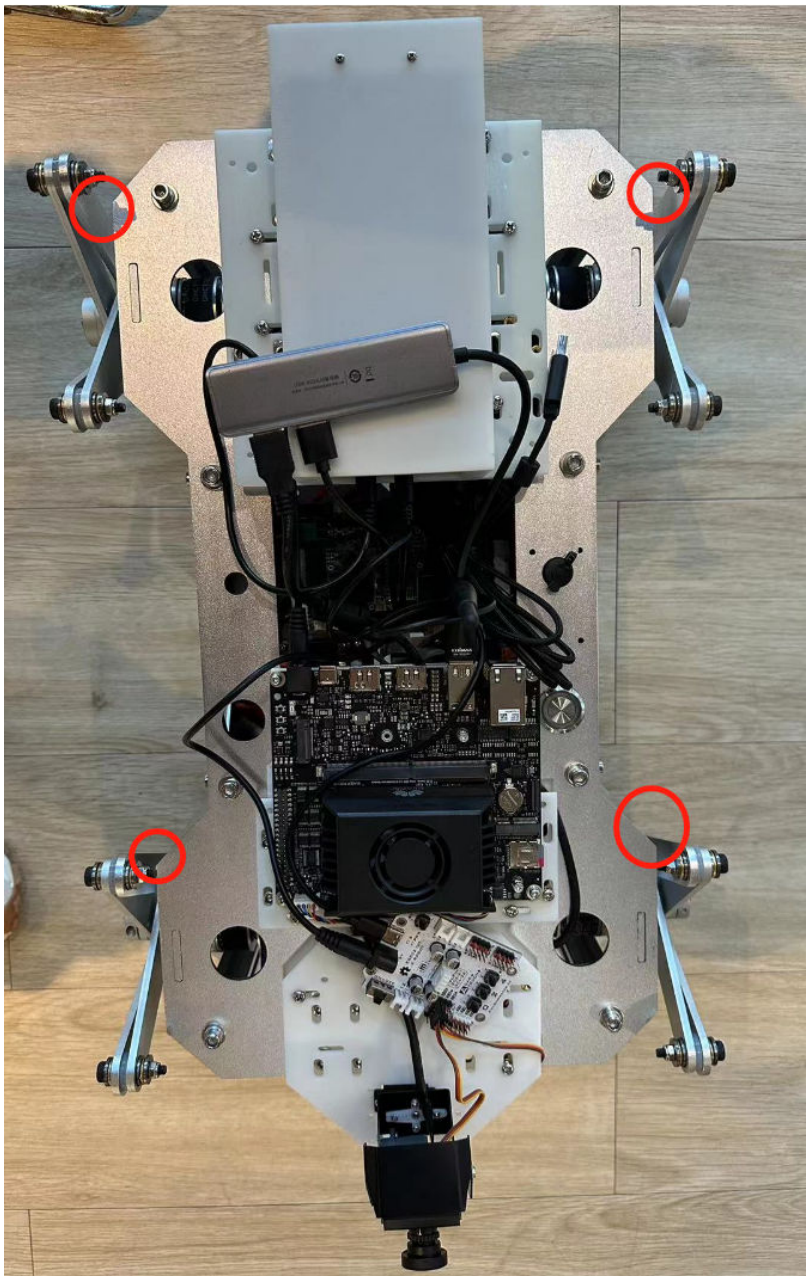
1. 以单条腿为例，扶起整个腿部，使底部黑色支持脚立于地面，呈现“Y”字型。
2. 沿金属腿外侧捏紧腿部，如图4-1所示。

图 4-1 腿部实物图



3. 以摄像头方向为头部，将每条腿部的后侧金属腿与机械狗外壳突出部分的角对齐，如图4-2所示。

图 4-2 对齐金属腿



4. 按下银色圆形的金属电源键，等待蓝色灯圈出现，中间的STM32开发板显示屏幕亮起，等待15s左右听到蜂鸣器出现一次蜂鸣后，机械狗就会自动站起。

步骤2 执行以下命令，进入到开发板上机械狗的代码目录下。

```
cd /home/HwHiAiUser/E2ESample/ascend-devkit-master/src/E2E-Sample/dogee/demo
```

步骤3 执行以下命令，一键安装所需要的依赖。

```
pip3 install -r requirements.txt
```

步骤4 执行以下命令，运行机械狗初始化脚本。

```
python3 dogee_pose.py
```

出现回显如图4-3所示，即表示完成了机械狗的初始化，等待输入指令进行姿态控制。

图 4-3 命令回显

```
----------------------------------------
STM32 is open, waiting for the cmd
----------------------------------------
```

键盘对应的控制表如表4-1所示。

表 4-1 键盘对应的控制表

按键	机械狗动作
i	前倾
k	后仰
j	左倾
l	右倾
Space	回正姿态位置

----结束

4.2 手动控制机械狗运动

说明

在手动控制机械狗姿态变化之后，需要重新启动机械狗的电源，再进行运动相关的操作。

步骤1 执行以下命令，进入到开发板上机械狗的代码目录下。

```
cd /home/HwHiAiUser/E2ESample/ascend-devkit-master/src/E2E-Sample/dogee/demo
```

步骤2 执行以下命令启动脚本，即可手动控制机械狗进行移动。

```
python3 main.py
```

相对应的键盘控制表如表4-2所示。

表 4-2 键盘对应的控制表

按键	机械狗动作
W	前进
S	后退
A	左转
D	右转
Space	停止

----结束

4.3 机械狗自动行走

将机械狗放置于地图上的车道引导线中，请参见3.1 地图绘制制作地图。

步骤1 执行以下命令，进入到开发板上机械狗的代码目录下。

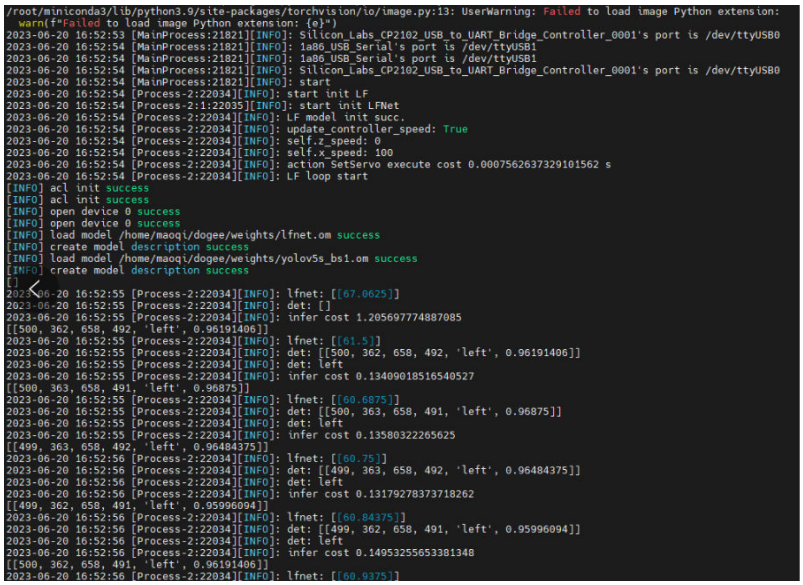
```
cd /home/HwHiAiUser/E2ESample/ascend-devkit-master/src/E2E-Sample/dogee/demo
```

步骤2 执行以下命令启动脚本，等待机械狗初始化完成。

```
python3 main.py --mode=easy
```

初始化完成回显如下所示：

图 4-4 命令回显



完成后，机械狗就会自动在引导线内行走。

----结束

4.4 机械狗锁定目标追踪

步骤1 将机械狗放置在距离追踪目标人2-3m距离。

步骤2 执行以下命令，进入到开发板上机械狗的代码目录下：

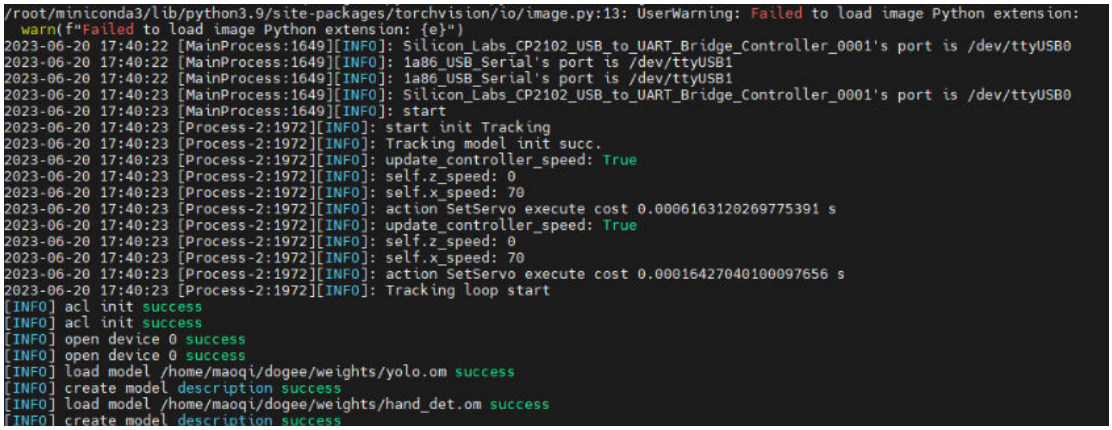
```
cd /home/HwHiAiUser/E2ESample/ascend-devkit-master/src/E2E-Sample/dogee/demo
```

步骤3 执行以下命令启动脚本，等待机械狗初始化。

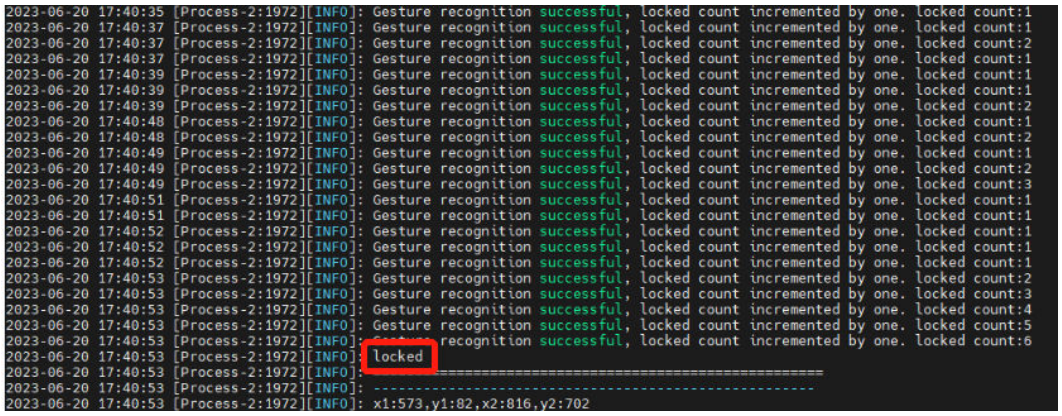
```
python3 main.py --mode=tracking
```

初始化完成回显如下所示：

图 4-5 命令回显

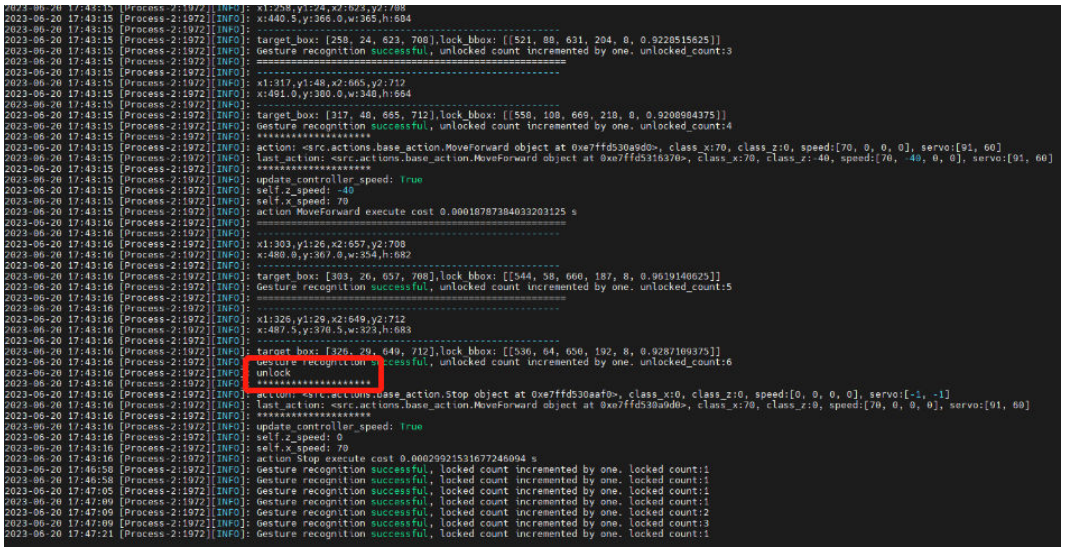


步骤4 在机械狗面前摆出“OK”的手势，等待开发者套件远程登录界面回显中出现**locked**，即表示已经锁定了追踪的目标，接下来就可以缓慢移动，机械狗就会跟随目标了。



解除追踪锁定，可参见以下两种方式：

- 面对机械狗，手掌对向摄像头，五指张开，持续3s左右，开发者套件远程登录界面回显中出现**unlock**字样，即表示解除了锁定，机械狗会保持静止状态。
- 快速脱离机械狗的视线超过10s左右，机械狗会认为丢失目标，开发者套件远程登录界面回显中出现**unlock**字样，即表示解除了锁定，机械狗会保持静止状态。



----结束

5 代码实现

5.1 主要代码文件介绍

表 5-1 文件介绍

文件（夹）名称	说明
demo/main.py	机械狗demo运行入口文件。
demo/requirements.txt	机械狗样例代码运行所需依赖。
demo/src/actions	机械狗基础与复杂运动代码。
demo/src/utils	工具类python文件包含OpenCV、acl工具等。
demo/src/scenes	机械狗预设场景相关代码。
demo/src/models	推理模型相关代码。

5.2 机械狗控制逻辑入口

“main.py”是机械狗控制代码的主入口，定义了控制模式及对应模式的后续调用实现。

首先导入一系列与参数有关的模块。

```
import os
from argparse import ArgumentParser
```



```
from multiprocessing import Process, Queue

from src.actions import Stop
from src.scenes import Manual, scene_initiator
from src.utils import getkey, log, CameraBroadcaster, SystemInfo, Controller, get_port, STM32_NAME,
ESP32_NAME
```

定义出可选参数更改小车的控制模式，可选命令行cmd控制（预留接口），手动控制，简易模式（自动巡线行走），声音控制（预留接口）、手动控制行走等。

```
def parse_args():
    parser = ArgumentParser()
    parser.add_argument('--mode', type=str, required=False, default='manual',
                        choices=['cmd', 'voice', 'tracking', 'easy', 'manual'])
    return parser.parse_args()
```

遵循简单性的原则，在程序的主入口需判断模式的设定后，进入到对应模式的实现中。

```
if __name__ == '__main__':

    # 获取STM32和ESP32设备的端口
    stm32_port = get_port(STM32_NAME)
    esp32_port = get_port(ESP32_NAME)

    # 使用获得的端口创建SystemInfo实例
    system_info = SystemInfo(stm32_port=stm32_port, esp32_port=esp32_port)

    # 初始化Controller
    ctrl = Controller()

    # 解析命令行参数
    args = parse_args()
    log.info('start')

    # 创建一个最大大小为1的消息队列
    msg_queue = Queue(maxsize=1)
    camera = CameraBroadcaster(system_info)
    shared_memory_name = camera.memory_name

    # 在后台启动摄像头进程
    camera_process = Process(target=camera.run)
    camera_process.start()

    # 检查选择的模式并执行相应的操作
    if args.mode == 'manual':
        task = Manual(shared_memory_name, system_info, msg_queue)
        process = Process(target=task.loop)
        process.start()
        try:
            while True:
                key = getkey()
                if key == 'esc':
                    process.join()
                    camera.stop_sign.value = True
                    camera_process.join()
                    break
                else:
                    msg_queue.put(key)
        except (KeyboardInterrupt, SystemExit):
            camera.stop_sign.value = True
            camera_process.join()
            os.system('stty sane')
            log.info('stopping.')
    elif args.mode == 'cmd':
        process_list = []
        record_map = {}
        try:
            log.info(f'start reading cmd')
            while True:
```



```
            command = input().strip()
            if command == 'stop':
                # 如果命令是'stop'，终止所有进程，包括摄像头，并退出循环
                for p in process_list:
                    p.kill()
                log.info(f'start put stop sign')

                ctrl.execute(Stop())
                camera.stop_sign.value = True
                camera_process.join()
                break
            elif command == 'clear':
                # 如果命令是'clear'，清空进程列表，重置控制器，并继续
                for p in process_list:
                    p.kill()
                process_list.clear()
                ctrl = Controller()
                ctrl.execute(Stop())
                log.info(f'clear succ')
                continue
            elif command == 'Manual':
                log.error(f'Does not support switching from cmd mode to manual mode')
                continue
            # 根据命令构建场景，并在单独的进程中启动它
            log.info(f'building scene {command}')
            scene = scene_initiator(command)
            log.info(f'{scene}')
            if scene is not None:
                scene_obj = scene(shared_memory_name, system_info, msg_queue)
                process = Process(target=scene_obj.loop)
                process.start()
                process_list.append(process)

    # 处理键盘中断或系统退出，停止摄像头进程和所有场景进程，并记录事件
    except (KeyboardInterrupt, SystemExit):
        camera.stop_sign.value = True
        camera_process.join()
        for process in process_list:
            process.kill()
        log.info('stopping.')

elif args.mode == 'voice':
    raise NotImplementedError('voice control is not currently supported.')
elif args.mode == 'easy':
    process_list = []
    task2 = scene_initiator('LF')(shared_memory_name, system_info, msg_queue)
    process_list.append(Process(target=task2.loop))

    for process in process_list:
        process.start()
    try:
        while True:
            key = getkey()
            if key == 'esc':
                for process in process_list:
                    process.kill()
                camera.stop_sign.value = True
                camera_process.join()
                break
            else:
                # 将按键放入消息队列供场景处理
                msg_queue.put(key)
    except (KeyboardInterrupt, SystemExit):
        camera.stop_sign.value = True
        camera_process.join()
        os.system('stty sane')
        log.info('stopping.')
elif args.mode == 'tracking':
    # 对于跟踪模式，启动对应场景（Tracking）的单独进程
```



```
    process_list = []
    task1 = scene_initiator('Tracking')(shared_memory_name, system_info, msg_queue)
    process_list.append(Process(target=task1.loop))

    for process in process_list:
        process.start()
    try:
        while True:
            key = getkey()
            if key == 'esc':
                for process in process_list:
                    process.kill()
                camera.stop_sign.value = True
                camera_process.join()
                break
            else:
                msg_queue.put(key)
    except (KeyboardInterrupt, SystemExit):
        # 处理键盘中断或系统退出，停止摄像头进程，并记录事件
        camera.stop_sign.value = True
        camera_process.join()
        os.system('stty sane')
        log.info('stopping.')
```

5.3 手动控制机械狗行走

导入一系列与参数有关的模块。

```
import datetime
import os

import cv2
import numpy as np

from src.actions import MoveForward, SetServo, Stop, MoveBack, BaseAction
from src.scenes.base_scene import BaseScene
from src.utils import log
```

手动控制行走模式功能定义。

```
class Manual(BaseScene):
    def __init__(self, memory_name, camera_info, msg_queue):
        super().__init__(memory_name, camera_info, msg_queue)
        # 初始速度设置为100
        self.speed = 100
        self.save_dir = os.path.join(os.getcwd(), 'capture')
        if not os.path.exists(self.save_dir):
            os.makedirs(self.save_dir, exist_ok=True)

    def init_state(self):
        # 初始化状态，执行设置Servo的动作
        self.ctrl.execute(SetServo(servo=[91, 10]))

    def loop(self):
        # 循环执行该场景
        ret = self.init_state()
        if ret:
            log.error(f'{self.__class__.__name__} init failed.')
            return
        frame = np.ndarray((self.height, self.width, 3), dtype=np.uint8, buffer=self.broadcaster.buf)
        log.info(f'{self.__class__.__name__} loop start')
        # 设置摄像头的角度为91，10
        last_action = SetServo(servo=[91, 10])

        while True:
            try:
                if not self.msg_queue.empty():
                    key = self.msg_queue.get()
```



```
            else:
                continue
        except KeyboardInterrupt:
            self.ctrl.execute(Stop())
            break

        # degree为z轴的角度数，正数为左转，负数为右转
        degree = 0
        if key == 'up':
            self.speed = min(self.speed + 1, 100)
        elif key == 'down':
            self.speed = max(self.speed - 1, 100)
        elif key == 'right':
            last_action = SetServo(servo=[91, 10])
        elif key == 'left':
            last_action = SetServo(servo=[91, 90])
        elif key == 'w':
            last_action = MoveForward(x=self.speed)
        elif key == 'a':
            last_action = MoveForward()
            degree = 40
        elif key == 's':
            last_action = MoveBack(x=self.speed)
        elif key == 'd':
            last_action = MoveForward()
            degree = -40
        elif key == 'space':
            last_action = Stop()
        elif key == 'esc':
            self.ctrl.execute(Stop())
            break
        elif key == 'c':
            save_img = frame.copy()
            cv2.imwrite(os.path.join(self.save_dir, f'{datetime.datetime.now()}.jpg'), save_img)
            log.info(f'image saved.')
        else:
            continue

        if isinstance(last_action, BaseAction):
            last_action.update_z_speed = False
            last_action.update_x_speed = False
            # 设置动作中z轴的角度变化
            last_action.z_speed = degree
            last_action.speed_setting = last_action.generate_speed_setting(self.speed, degree)
        self.ctrl.execute(last_action)
```

5.4 机械狗巡引导线行走

导入一系列与参数有关的模块。

```
import os
import time

import numpy as np

from src.actions import SetServo, Stop
from src.models import LFNet
from src.scenes.base_scene import BaseScene
from src.utils import log
```

自动巡线行走功能定义。

```
class LF(BaseScene):
    def __init__(self, memory_name, camera_info, msg_queue):
        super().__init__(memory_name, camera_info, msg_queue)
        self.net = None
        self.forward_spd = 22
```



```
    def init_state(self):
        log.info(f'start init {self.__class__.__name__}')
        lfnet_path = os.path.join(os.getcwd(), 'weights', 'lfnet.om')
        if not os.path.exists(lfnet_path):
            log.error(f'Cannot find the offline inference model(.om) file needed for {self.__class__.__name__}
scene.')
            return True
        self.net = LFNet(lfnet_path)
        log.info(f'{self.__class__.__name__} model init succ.')
        # 设置舵机水平角度90度，垂直角度10度
        self.ctrl.execute(SetServo(servo=[90, 10]))
        return False

    def loop(self):
        ret = self.init_state()
        if ret:
            log.error(f'{self.__class__.__name__} init failed.')
            return
        frame = np.ndarray((self.height, self.width, 3), dtype=np.uint8, buffer=self.broadcaster.buf)
        log.info(f'{self.__class__.__name__} loop start')
        try:
            while True:
                if self.stop_sign.value:
                    break
                if self.pause_sign.value:
                    continue
                start = time.time()
                img_bgr = frame.copy()
                curr_steering_val = float(self.net.infer(img_bgr)[0])
                log.info(f'lfnet: {curr_steering_val}')

                log.info(f'infer cost {time.time() - start}')
        except KeyboardInterrupt:
            self.ctrl.execute(Stop())
```

5.5 机械狗锁定目标追踪

导入一系列与参数有关的模块。

```
import os
from multiprocessing import Process, Queue

import numpy as np
from src.actions import SetServo, Stop, MoveForward
from src.models import YoloV5
from src.models.yolov7 import YoloV7
from src.scenes.base_scene import BaseScene
from src.utils import log
from src.utils.cv_utils import cal_iou, xyxy_to_xywh, xywh_to_xyxy, cal_inter_small
from src.utils.constant import STATE_OBSERVATION_MATRIX, STATE_TRANSITION_MATRIX,
PROCESS_NOISE_COVARIANCE_MATRIX, \
    OBSERVATION_NOISE_COVARIANCE_MATRIX
```

锁定目标追踪功能定义。

```
class Tracking(BaseScene):
    def __init__(self, memory_name, camera_info, msg_queue):
        super().__init__(memory_name, camera_info, msg_queue)
        self.in_queue1 = Queue(1)
        self.in_queue2 = Queue(1)
        self.out_queue1 = Queue(1)
        self.out_queue2 = Queue(1)

    def init_state(self):
        log.info(f'start init {self.__class__.__name__}')
        yolov5_model_path = os.path.join(os.getcwd(), 'weights', 'yolo.om')
        yolov7_model_path = os.path.join(os.getcwd(), 'weights', 'hand_det.om')
```



```
        if not os.path.exists(yolov5_model_path) or not os.path.exists(yolov7_model_path):
            log.error(f'Cannot find the offline inference model(.om) file needed for {self.__class__.__name__}
scene.')
            return True
        lock_process = Process(target=YoloV7(yolov7_model_path).infer, args=(self.in_queue2,
self.out_queue2))
        tracking_process = Process(target=YoloV5(yolov5_model_path).infer, args=(self.in_queue1,
self.out_queue1))
        lock_process.start()
        tracking_process.start()
        log.info(f'{self.__class__.__name__} model init succ.')
        self.ctrl.execute(SetServo(servo=[91, 60]))
        return False

    def get_start_bbox(self, tracking_bboxes, lock_bboxes):
        """
        判断当前是否存在锁定目标，判断条件为：人体的检测框与手势的检测框的相交面积 / 手势检测框的面积
大于0.9 且手势为锁定手势
        @param tracking_bboxes: 所有的人体检测框
        @param lock_bboxes: 所有的手势检测框
        @return:
        """
        fives = []
        for i, (x1, y1, x2, y2, cls, conf) in enumerate(lock_bboxes):
            if cls != 6:
                continue
            fives.append([x1, y1, x2, y2])

        for i, (x1, y1, x2, y2, cls, conf) in enumerate(tracking_bboxes):
            human_box = [x1, y1, x2, y2]
            for i, (xx1, yy1, xx2, yy2) in enumerate(fives):
                hand_box = [xx1, yy1, xx2, yy2]
                iou = cal_inter_small(hand_box, human_box)
                if iou > 0.9:
                    return [x1, y1, x2, y2]
        return None

    def get_count(self, target_box, lock_bboxes):
        """
        判断当前锁定目标是否解锁，判断条件为：人体的检测框与手势的检测框的相交面积 / 手势检测框的面积
大于0.9 且手势为解锁手势
        @param target_box:当前锁定目标的检测框
        @param lock_bboxes:所有的手势检测框
        @return:
        """
        for i, (x1, y1, x2, y2, cls, conf) in enumerate(lock_bboxes):
            if cls != 8:
                continue
            hand_box = [x1, y1, x2, y2]
            iou = cal_inter_small(hand_box, target_box)
            if iou > 0.9:
                return 1
        return 0

    def loop(self):
        # 初始化循环与模型
        ret = self.init_state()
        if ret:
            log.error(f'{self.__class__.__name__} init failed.')
            return
        # 初始化舵机角度
        self.ctrl.execute(SetServo(servo=[91, 60]))
        # 创建np数组，绑定共享内存
        frame = np.ndarray((self.height, self.width, 3), dtype=np.uint8, buffer=self.broadcaster.buf)
        log.info(f'{self.__class__.__name__} loop start')
        locked = False
        locked_count = 0
        unlocked_count = 0
        lost_count = 0
```

```
        iou_threshold = 0.3  # 匹配时的阈值

        x_posterior = None
        x_speed = 70
        z_speed = 40
        last_action = SetServo(servo=[91, 60])
        while True:
            action = Stop()
            if self.stop_sign.value:
                break
            if self.pause_sign.value:
                continue
            # 获取图像
            img_bgr = frame.copy()
            # 图像通过消息队列传递给模型
            self.in_queue1.put(img_bgr)
            self.in_queue2.put(img_bgr)

            # 获取模型的输出
            tracking_bbox = self.out_queue1.get()
            lock_bbox = self.out_queue2.get()

            if not locked:
                # 获取锁定目标的检测框
                start_bbox = self.get_start_bbox(tracking_bbox, lock_bbox)
                # 当前没有找到锁定目标
                if start_bbox is None:
                    locked_count = 0
                    continue
                else:
                    locked_count += 1
                    log.info(
                        f'Gesture recognition successful, locked count incremented by one. locked count:
{locked_count}')

                if locked_count > 4:
                    locked = True
                    locked_count = 0
                    action = SetServo(servo=[91, 60])
                    log.info(f'locked')

                    # 将检测框转换为中心点坐标和宽高，初始化状态
                    initial_box_state = xyxy_to_xywh(start_bbox)
                    initial_state = np.array(
                        [[initial_box_state[0], initial_box_state[1], initial_box_state[2], initial_box_state[3],
                          0, 0]]).T  # [中心x,中心y,宽w,高h,dx,dy]

            if locked:
                log.info(f'=========================================================')
                # 初始化卡尔曼滤波器
                if x_posterior is None:
                    x_posterior = np.array(initial_state)
                    p_posterior = np.array(np.eye(6))
                    z = np.array(initial_state)
                max_iou = iou_threshold
                max_iou_matched = False

                # 使用最大IOU来寻找观测值
                for i, (x1, y1, x2, y2, cls, conf) in enumerate(tracking_bbox):
                    xyxy = [x1, y1, x2, y2]
                    iou = cal_iou(xyxy, xywh_to_xyxy(x_posterior[0:4]))
                    if iou > max_iou:
                        target_box = xyxy
                        max_iou = iou
                        max_iou_matched = True
                if max_iou_matched:
                    log.info(f'---------------------------------------------------------')
                    # 如果找到了最大IOU BOX,则认为该框为观测值
```



```
            xywh = xyxy_to_xywh(target_box)
            x1, y1, x2, y2 = target_box
            x, y, w, h = xywh
            log.info(f'x1:{x1},y1:{y1},x2:{x2},y2:{y2}')
            log.info(f'x:{x},y:{y},w:{w},h:{h}')
            log.info(f'----------------------------------------------------------')

            # 运动控制
            while True:
                if h >= 700 and 450 < x < 750:
                    action = Stop(servo=[91, 60])
                    break
                if x <= 450:
                    action = MoveForward(x=x_speed, z=-z_speed, servo=[91, 60])
                elif x >= 750:
                    action = MoveForward(x=x_speed, z=z_speed, servo=[91, 60])
                else:
                    action = MoveForward(x=x_speed, z=0, servo=[91, 60])
                break

            # 计算dx,dy
            dx = xywh[0] - x_posterior[0]
            dy = xywh[1] - x_posterior[1]

            z[0:4] = np.array([xywh]).T
            z[4::] = np.array([dx, dy])
            lost_count = 0
        else:
            lost_count += 1

        if max_iou_matched:
            # 进行先验估计
            x_prior = np.dot(STATE_TRANSITION_MATRIX, x_posterior)

            # 计算状态估计协方差矩阵P
            p_prior = np.dot(np.dot(STATE_TRANSITION_MATRIX, p_posterior),
                             STATE_TRANSITION_MATRIX.T) + PROCESS_NOISE_COVARIANCE_MATRIX
            # 计算卡尔曼增益
            k1 = np.dot(p_prior, STATE_OBSERVATION_MATRIX.T)
            k2 = np.dot(np.dot(STATE_OBSERVATION_MATRIX, p_prior),
                        STATE_OBSERVATION_MATRIX.T) + OBSERVATION_NOISE_COVARIANCE_MATRIX
            kalman_gain = np.dot(k1, np.linalg.inv(k2))
            # 后验估计
            x_posterior_1 = z - np.dot(STATE_OBSERVATION_MATRIX, x_prior)
            x_posterior = x_prior + np.dot(kalman_gain, x_posterior_1)

            # 更新状态估计协方差矩阵P
            P_posterior_1 = np.eye(6) - np.dot(kalman_gain, STATE_OBSERVATION_MATRIX)
            p_posterior = np.dot(P_posterior_1, p_prior)
        else:
            # 如果IOU匹配失败，此时失去观测值，那么直接使用上一次的最优估计作为先验估计
            # 此时直接迭代，不使用卡尔曼滤波
            x_posterior = np.dot(STATE_TRANSITION_MATRIX, x_posterior)
            action = Stop()

    if lost_count > 20:
        action = Stop()
        locked = False
        x_posterior = None
        log.info(f'target lost')

    if locked:
        log.info(f'target_box: {target_box},lock_bbox: {lock_bbox}')

        if self.get_count(target_box, lock_bbox):
            unlocked_count += 1
            log.info(f'Gesture recognition successful, unlocked count incremented by one. '
                     f'unlocked_count:{unlocked_count}')
        else:
```



```
                unlocked_count = 0

            if unlocked_count > 5:
                log.info(f'unlock')
                locked = False
                action = Stop()
                x_posterior = None

        # 执行动作，如果动作和上一次动作相同，则不执行
        if action.speed_setting == last_action.speed_setting:
            pass
        else:
            log.info('*' * 20)
            log.info(
                f'action: {action}, class_x:{action.x_speed}, class_z:{action.z_speed}, '
                f'speed:{action.speed_setting}, servo:{action.servo_angle}')
            log.info(
                f'last_action: {last_action}, class_x:{last_action.x_speed}, class_z:{last_action.z_speed}, '
                f'speed:{last_action.speed_setting}, servo:{last_action.servo_angle}')
            log.info('*' * 20)
            self.ctrl.execute(action)
            last_action = action
```

