Atlas 200I DK A2 开发者套件
23.0.RC3
智能车应用开发指南

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

1.1 外观结构

小车的主要组成模块包括：

- 前中后外壳结构支撑模块。
- TT减速电机与麦克纳姆轮的运动模块。
- 电源供电模块。
- ESP32控制模块。
- 广角摄像头视觉感知模块。
- 激光雷达点云感知模块。
- Atlas 200I DK A2开发者套件。

图 1-1 小车外观结构图

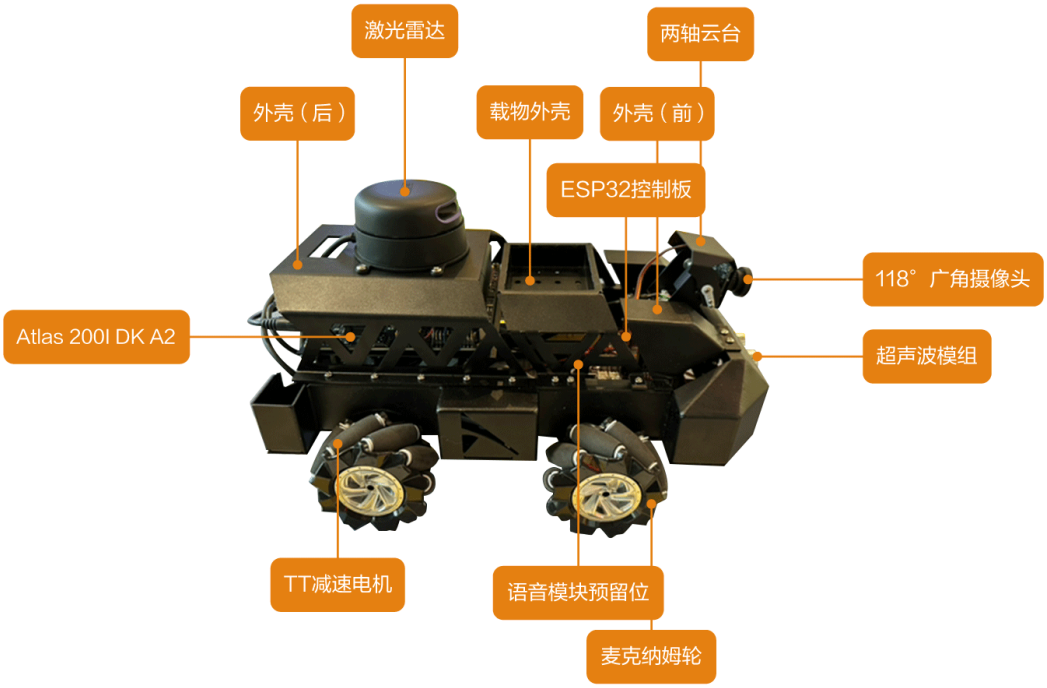


图 1-2 正面图

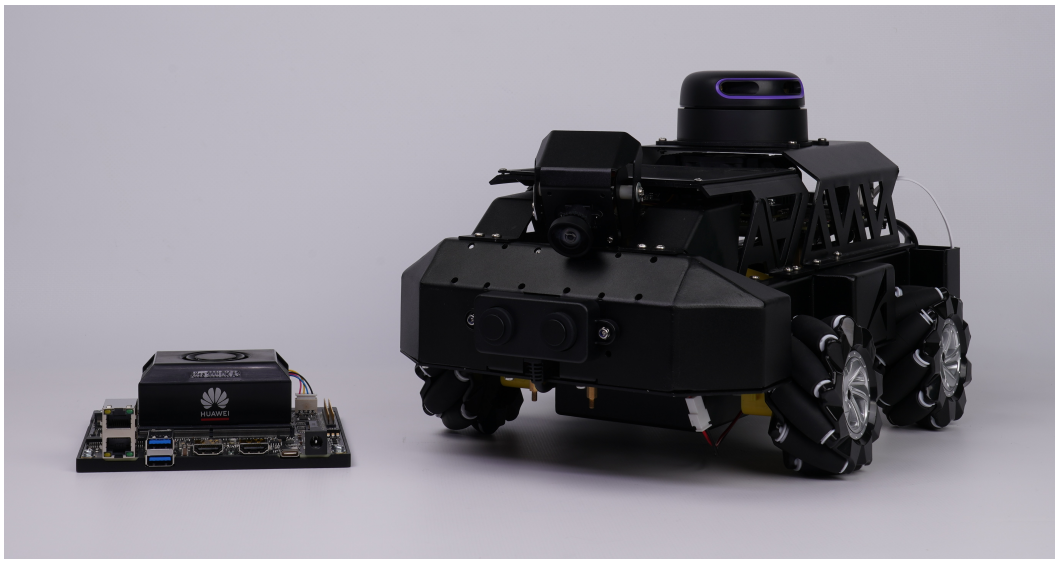
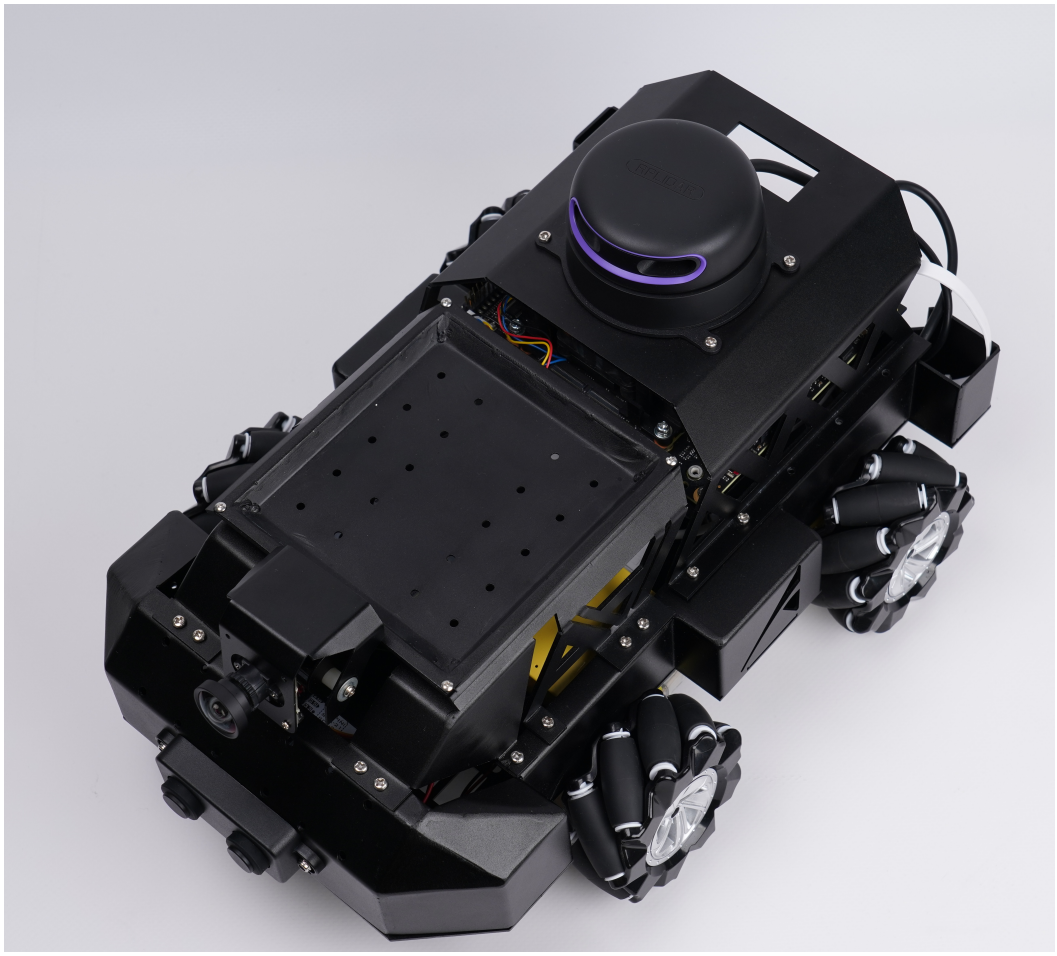


图 1-3 俯视图



1.2 功能与原理介绍

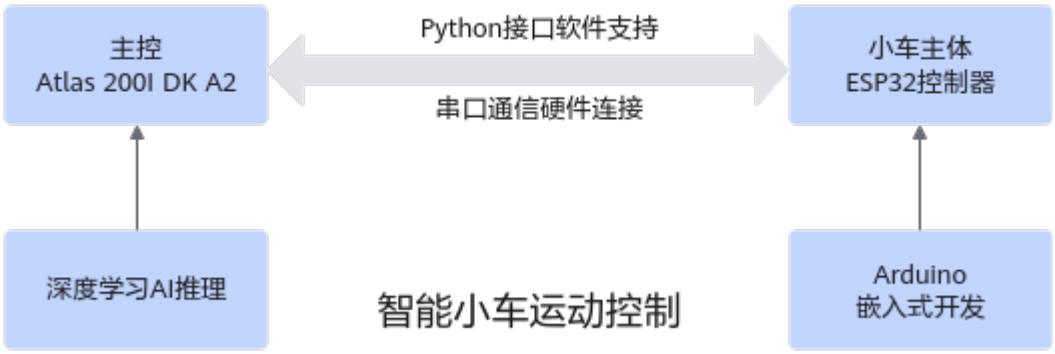
智能小车通过摄像头感知周围环境自主地进行运动控制，采集环境数据后在开发者套件上进行AI推理，根据推理结果发出指令控制小车的运动状态。小车运动状态的控制需要借助ESP32微控制器，使用Arduino平台可以对其进行嵌入式开发。主控与小车主体间控制指令的发出和数据的返回，需要通过串口协议进行双向通信。

智能小车从底层硬件到上层AI应用需要完成的任务。

- 了解智能小车的组成及部件原理
- 实现基于Arduino的硬件控制功能。
- 串口通信协议定义。
- ESP32硬件主程序设计。
- 开发者套件与ESP32进行指令通信的接口程序开发。
- AI推理应用开发。

本章节将对这些主要任务进行详细介绍。

图 1-4 实现原理图



功能介绍

须知

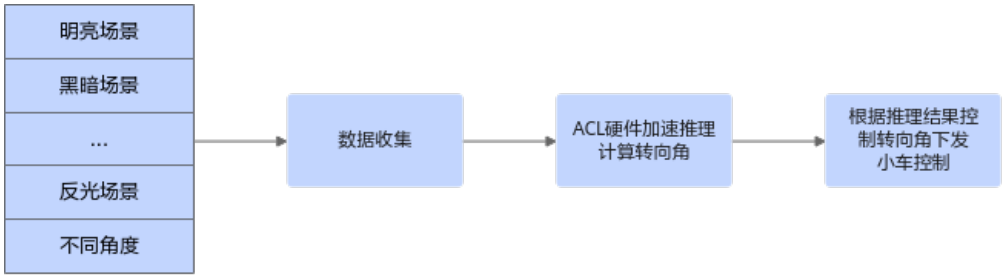
当前功能所用模型，均为已训练并进行模型转换的“.om”文件，如果用户需要使用自己的场景或地图，需参考《**使用模型适配工具生成推理应用**》，使用模型适配工具自行训练生成模型，并上传至开发者套件。

- 自动驾驶

  基于开发者套件的内置的YoloV5和轻量级直线行驶矫正神经网络模型推理结果引导智能小车在规定赛道上的自动循迹行驶。

  a. 通过摄像头收集不同场景下的路况信息数据，判断当前位置并识别转弯标识。
  b. 使用模型适配工具训练转换的om模型进行推理，计算转向角，最后根据推理结果下发指令控制小车转向。

  图 1-5 自动驾驶功能原理

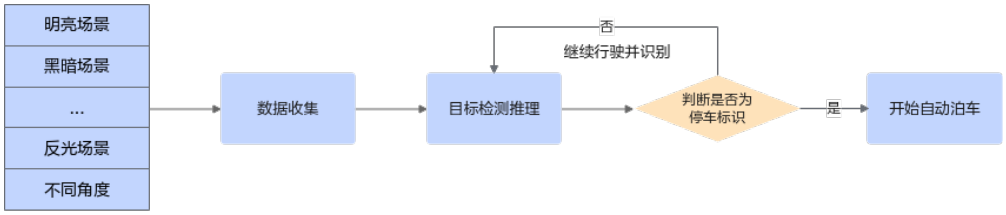
  

- 自动泊车

  基于开发者套件的内置YoloV5神经网络模型的推理结果引导智能小车在模拟的复杂交通场景和更贴近实际的道路上行驶。

  a. 通过摄像头收集不同场景下的停车标志信息数据。
  b. 使用模型适配工具训练转换的om模型进行推理，判断当前标识是否为停车标志，“是”则开始自动泊车，“否”则继续行驶并收集数据。
  c. 识别到目标后，判断目标停车标识和智能小车当前位置之间的距离，根据提前设置的阈值（开发者可自定义）判断到达停车距离，并开始横向移动进停车位。

图 1-6 自动泊车功能原理



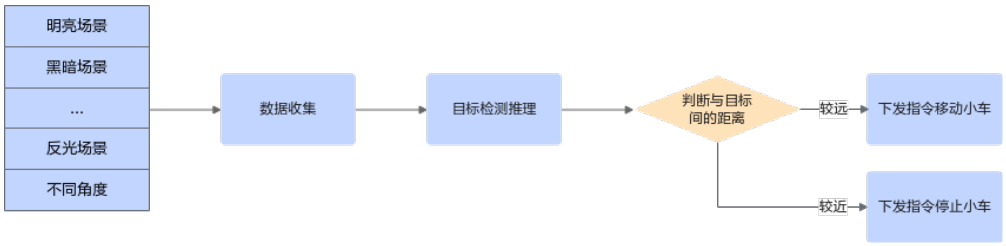
- 目标跟踪

  基于开发者套件的内置YoloV5神经网络模型的推理结果引导智能小车在多车行驶的模拟道路上行驶。

  a. 通过摄像头收集不同场景下需要跟随的目标，识别前方目标并跟随目标行驶。

  b. 使用模型适配工具训练转换的om模型进行推理，根据识别到物体的大小来推算小车和追踪目标之间的距离，下发指令到控制系统，控制小车的移动和停止，从而达到跟随效果（该功能也可以移植到追踪其他物体的使用场景中，待开发者二次开发）。

图 1-7 目标跟踪功能原理



1.3 小车组成

1.3.1 主要部件

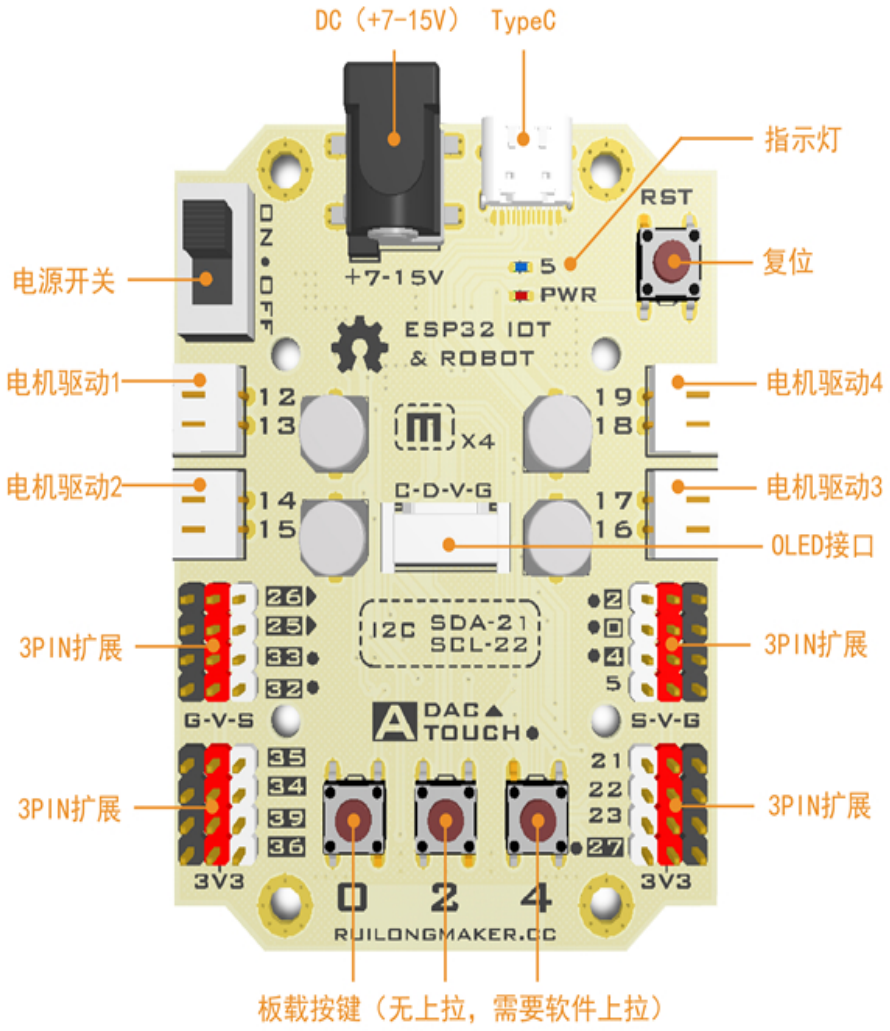
小车硬件主要包含以下部分组件：

- 开发者套件：根据摄像头收集的数据进行AI推理，将推理结果发送给主控模块控制小车运行。
- 主控ESP32开发板：信号接收与发送，执行并调动其他部分完成指令。
- 直流电机：为小车提供动力，可调节旋转方向和速度。
- 舵机：装载摄像头并调节拍摄角度，可控角度0~180°。
- 超声波传感器：可实现2~400cm的非接触距离感测功能，测距精度可以高达3mm。
- 小车载板及电池：载板提供支架，电池提供电源。

1.3.2 硬件主控 ESP32

ESP32 IOT&Robot Board （4Motor）物联网机器人开发板（4 电机），是一款以ESP-WROOM-32单片机为核心的物联网开发板。

图 1-8 ESP32 主控板接口



ESP32开发板能够处理数据和进行运算，最终通过引脚传递不同的电信号完成指令。通过不同引脚接入其他外设可以实现丰富的功能，更多详情及引脚定义可查看**官方文档**。

通过烧录控制程序可实现对外接传感器的感知和控制，常见的ESP32程序编译工具有Arduino、Mixly、MicroPython等，本文以Arduino工具为例。

表 1-1 技术参数

参数	说明
主控板	ESP32单片机
处理器	Tensilica LX6双核处理器
主频	240Mhz时钟频率
SRAM	520KB
Flash	8MB

参数	说明
Wi-Fi标准	FCC/CE/TELEC/KCC
Wi-Fi协议	8022.11 b/g/n/d/e/i/k/r，A-MPDU与A-MSDU聚合，支持0.4us防护问题
频率范围	2.4-2.5GHz
蓝牙协议	符合蓝牙v4.2BR/EDR/BLE标准
供电方式	CVSD和SBC音频低功耗 10uA
工作电压	Micro USB供电
最大工作电流	200mA

1.3.3 直流电机及其控制

电机的转动原理：1个电机对接ESP32开发板2个引脚，当两个引脚分别接高低电平则开始以一定方向转动，电平相反则反方向转动，同为低或者高均不转动。

电机转速控制：控制电信号PWM波的占空比，占空比越大，说明周期内高电平持续时间越长，电机转速越快。

说明

一般引脚的高电平为3.3V，通常不够驱动电机转动，需要5V左右电压可正常驱动轮子转动，所以需要使用大电池给小车供电。

图 1-9 电机位置

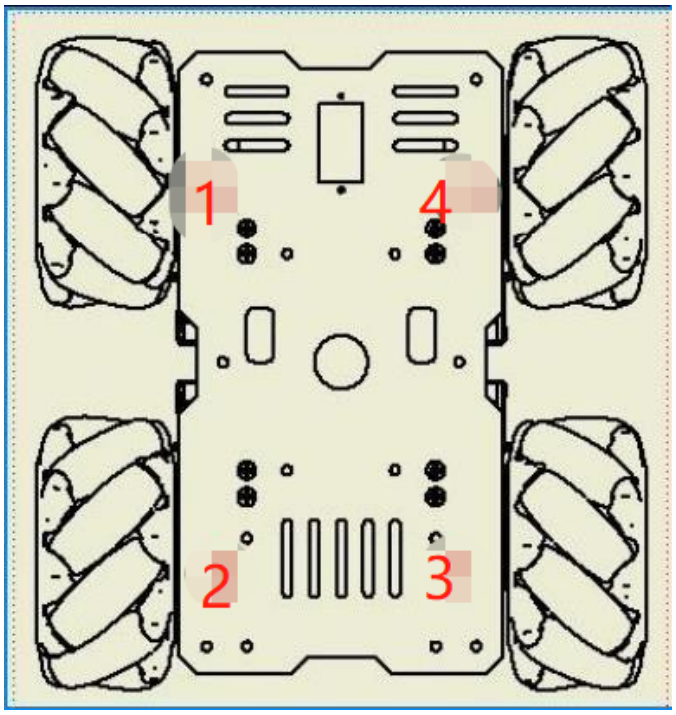


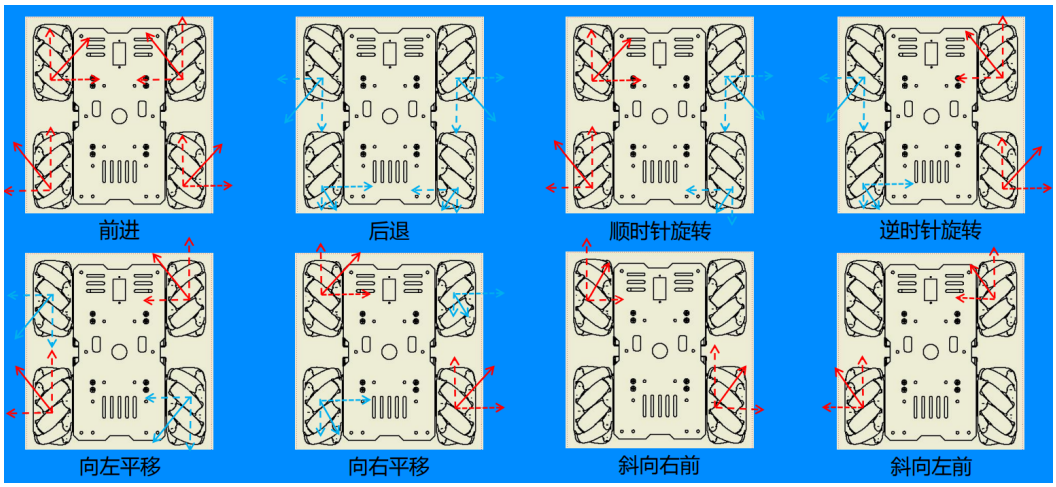
表 1-2 电机与引脚序号

电机序号	对应引脚
1	12/13
2	14/15
3	16/17
4	18/19

小车车轮配置麦克纳姆轮，车轮原理及方向控制如下图所示，只需调节4个电机的转动方向及速度即可实现小车的各种运动方式。

车轮原理：依靠各自机轮的方向和速度，这些力的最终合成在任何要求的方向上产生一个合力矢量，从而保证了这个平台在最终的合力矢量的方向上能自由地移动，而不改变机轮自身的方向。在它的轮缘上斜向分布着许多小滚子，故轮子可以横向滑移。小滚子的母线很特殊，当轮子绕着固定的轮心轴转动时，各个小滚子的包络线为圆柱面，所以改轮能够连续地向前滚动。麦克纳姆轮结构紧凑，运动灵活，是很成功的一种全方位轮。有4个这种新型轮子进行组合，可以更灵活方便的实现全方位移动功能。

图 1-10 麦克纳姆轮方向控制



1.3.4 舵机及其控制

舵机是由直流电机、减速齿轮组、传感器(可变电阻 ）和控制电路组成的一套自动控制系统。

图 1-11 TS90A 9D 舵机

舵机只能在一定角度内转动（有最大旋转角度比如：180度），支持反馈转动的角度信息（如果带编码器就可以反馈角度），支持控制某物体转动一定角度。

舵机的控制原理：舵机的输入线共有三条，红色是电源线，棕色是地线，这两根线给舵机提供最基本的能源保证，主要供能是电机的转动消耗。另外一根线是控制信号线，一般为桔黄色。舵机的信号线作为输入线用于接收PWM信号（定时器产生）。一般PWM的周期是20ms，那么对应的频率是50hz。那么改变不同的占空比就可以控制转动的角度。其中占空比从0.5到180度，呈线性变化。

图 1-12 舵机角度

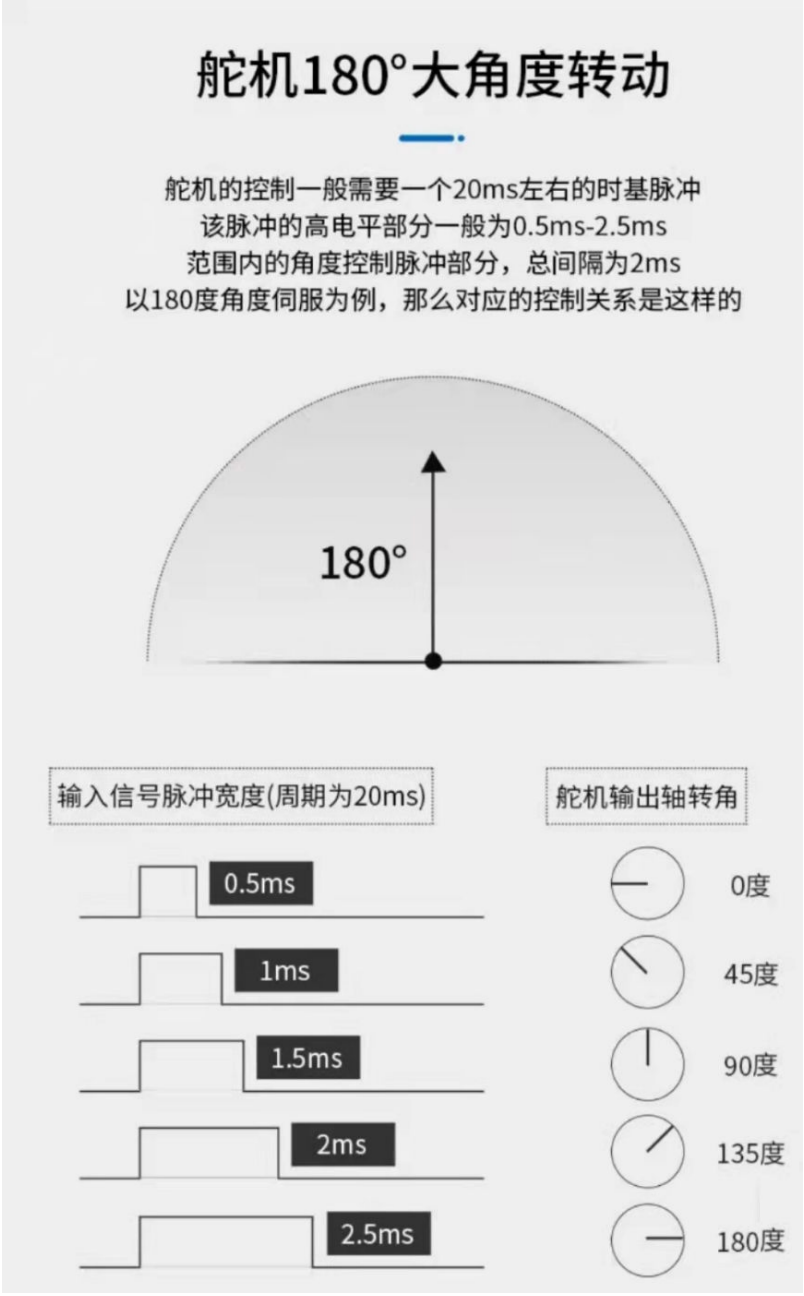


图3 两轴云台

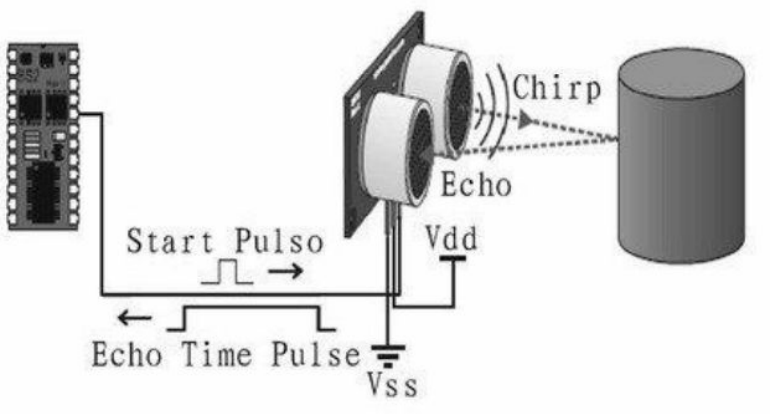
1.3.5 超声波传感器

超声波发射探头发出的超声波脉冲，传到物体表面，反射后传到接收探头，测定出超声脉冲从发射到接收所需的时间，根据传输媒质中的声速，计算从探头到物体表面之间的距离，工作原理如下：

- 采用IO口TRIG触发测距，给至少10us的高电平信号；
- 模块自动发送8个40khz的方波，自动检测是否有信号返回；
- 有信号返回，通过IO口ECHO输出一个高电平，高电平持续的时间就是超声波从发射到返回的时间；
- 测试距离 = (高电平时间 * 声速(340M/S)) / 2。

图 1-13 超声波传感器

图 1-14 工作原理



2 样例组装

2.1 组件设计

智能小车外壳用户可直接点击链接获取模型，按当前模型规格制作组件，也可自行DIY设计模块与功能，建模软件需自行准备。

- 使用已完成的建模模型。

  获取链接，用户可下载已设计完成的模型，提供给供应商直接进行组件制作，快速搭建智能小车样例。

- DIY设计模型。

  若用户自行设计智能小车的外观框架、功能模块等，可使用SoildWorks建模软件，根据以下约束与规格制作组件，避免模块之间互相影响。软件请用户自行获取准备。

须知

小车外壳DIY设计与生产时需遵循以下原则约束。

- 功能模块的使用层不可有其他功能与模块，否则会出现功能冲突。
- 所有的紧固件开孔需要符合国标尺寸，需考虑可接受误差范围，具体尺寸可查阅**表2-1**。
- 设计初期需要获取到所有零部件的固定孔径、固定孔距以及边缘最大尺寸，其中边缘最大尺寸需要包含运动极限尺寸，否则会出现运动干涉，影响正常运行。具体尺寸可查阅**表2-1**。
- 设计需要考虑实际组装流程、零件到装配体的组装过程、紧固件的固定顺序。另外在零件的设计绘制图纸过程中也需准备好建图拉伸和切除的顺序。
- 小车所需要的传感器较多，大多数传感器出厂设置的线长和接口方式固定，需要提前预留走线的位置和孔洞，使大部分连接线隐藏化。
- 小车3D结构的选材和生产，可供选择的材质为铝合金、ABS3D打印、塑料注塑成型。其中铝合金材质在重量上要重于另外两种材质，所以在二次开发时要结合质量和外观选择不同的材质，不同材质会导致小车车轮承重不同，需要在控制层面调试不同的电机运行速度，以上三种材料均在普通TT减速电机的适用范围内。

表 2-1 小车部件尺寸表

部件名称	部件关键尺寸参数/mm		
ESP32单片机	四孔中心距32*64。	孔径3.4。	最外边缘最大尺寸尺寸54*72*30。
ESP32电池盒	四孔中心距32*55.5。	孔径4。	最外边缘最大尺寸40*80*30。
广角摄像头模组	四孔中心距27*27~29*29。	孔径1。	最外边缘最大尺寸32.6*32.6*26.8。
深度相机	四孔中心距99*78。	孔径3。	最外边缘最大尺寸160*90*80。
TT电机	双孔中心距8。	孔径3。	-
6路麦克风模组	四孔中心距101*49。	孔径4。	最外边缘最大尺寸109*83*19。
激光雷达	四孔中心距80*48。	孔径4.3。	最外边缘最大尺寸88*80*42。
Atlas 200I DK A2开发者套件	四孔中心距112*127。	孔径3。	最外边缘最大尺寸135*120*50。
舵机云台	两孔中心距28，距离中心线左9右19。	孔径2。	-

部件名称	部件关键尺寸参数/mm		
USB扬声器	四孔中心距63.5*24.5。	孔径3.5。	最外边缘最大尺寸69*30*16。
扬声器I2C语音合成	四孔中心距38.5*22.5。	孔径3.5。	最外边缘最大尺寸44*28*8。

2.2 准备组件

智能小车所需组件见表2-2。

表 2-2 智能小车组件表

模块名称	数量/个	规格	是否需要单独购买
昇腾开发者套件	1	Atlas 200I DK A2	否
麦克纳姆轮	4	直径80mm，适用TT减速电机	是
TT直流减速电机	4	TT电机带插头+立体支架	是
ESP32单片机	1	单主板	是
超声波传感器	1	PWM受控，超声波测距模块防水传感器探头 3cm小盲区	选购
无线路由器	1	唯一规格，MT300N-V2迷你路由器	是
12V、9V、5V电源	1	12V9V5V(38400mah)	是
广角摄像头	1	模组HF867_2.7mm118度无畸变（备注选择可插拔接线）	是
舵机云台	1	精巧版两轴云台	是
激光雷达	1	A2雷达+转接板支架	选购
2pinXH2.54公母延长线	4	P2	是
杜邦线母对公	5	长度>10cm	选购
套筒扳手	1	十字套筒，适用2mm-7mm螺母	是
外壳与结构件3D打印	1	-	根据小车三维结构而定，选用DIY材料依开发者实际需求决定。
铝合金定制加工	1	-	
USB扩展坞	1	1USB口转4USB口	是

模块名称	数量/个	规格	是否需要单独购买
停车标志	1	停车标志类似下图 	是
小车所需紧固件 参见表2-3	10	-	是

表 2-3 紧固件数量表

紧固件名称	数量/个
M3*9螺钉/防松螺母	10
M2*7螺钉/防松螺母	10
M3*6螺钉/防松螺母	80
M4*5螺钉/防松螺母	20
M3*15+4单头六角铜柱	8
M4*50+6单头六角铜柱	8
M4*25+6单头六角铜柱	8
M4*35+6单头六角铜柱	8
M4*55+6单头六角铜柱	8
M2/M3/M4国标螺母/垫圈	20

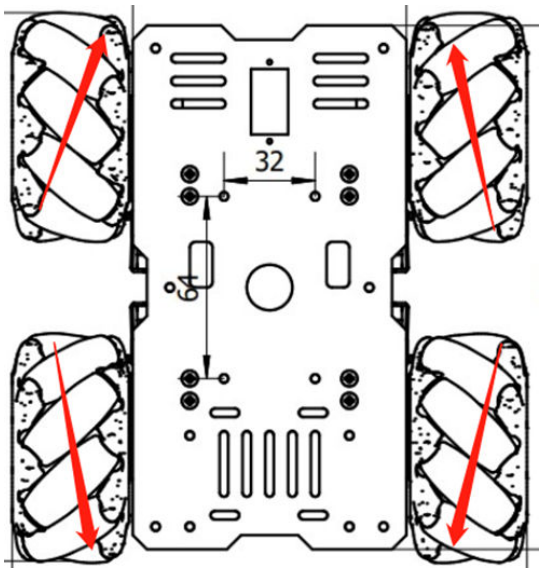
2.3 组装注意事项

1. 注意螺钉和螺栓的固定顺序，第一次固定不要完全固定，以防有顺序错误需要拆卸的情况。
2. 为了便于插拔ESP32和开发者套件上的各类接口，最上层的外壳未设计为完全固定的形式，可随时取下，所以移动小车时建议手托到底盘。
3. 为了防止小车前端的外壳遮挡广角摄像头的视线，也为了提高摄像头位置的自由度，广角摄像头载板设置了多挡位调节，开发者可以选择合适的摄像头前后位置。
4. 注意麦克纳姆轮的安装顺序和前后轮位置，否则可能会影响小车的平移和原地旋转效果，左右麦克纳姆轮上的扁状滚轮上的摩擦力方向需要均指向小车的中轴线方向，如所示。

说明

麦克纳姆轮转速差异较大请参见**智能车手动控制时车轮转向错误或四个轮子转速差异较大**解决。

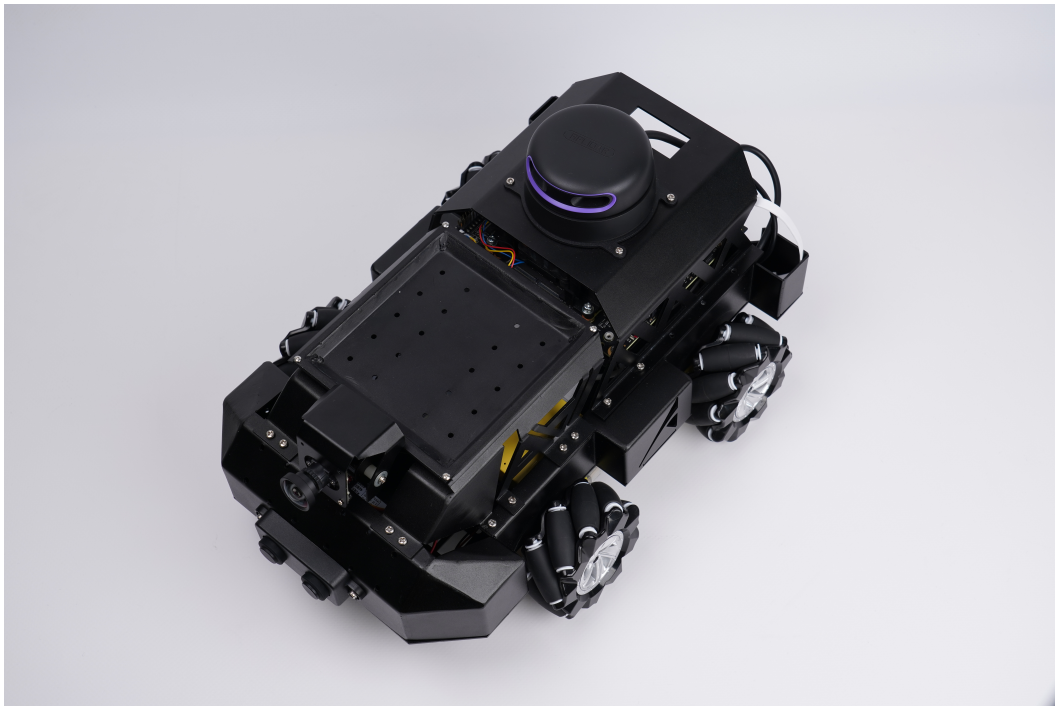
图 2-1 麦克纳姆轮摩擦力方向



2.4 组装步骤

依照从下至上的顺序组装小车，组装底盘上的连接件，按照“电池 > TT直流电机 > 麦克纳姆轮 > 广角摄像头模块 >ESP32单片机 > 第一层外壳 > 超声波模块 > 语音麦克风阵列 > Atlas 200I DK A2开发者套件 > 激光雷达连接到后段外壳 > 前中段外壳顺”序组装，实物图如**图2-2**所示。

图 2-2 实物图

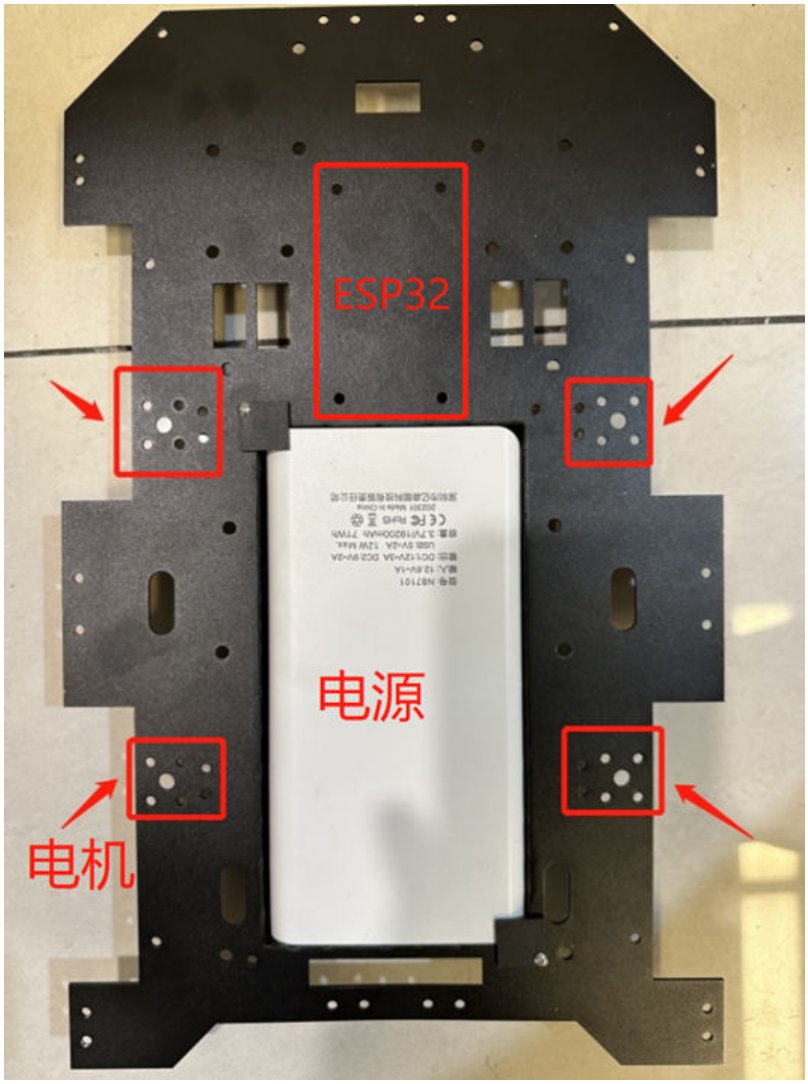


步骤1 准备3D打印或铝合金加工的小车底盘，并确定好对应的开孔与位置和设计图中相同，并安装电源，直接将电池放到电池槽里即可，电池槽后方的矩形空洞可用于将全部电源线埋藏在小车内部。如图2-3所示。

图 2-3 安装电源

步骤2 将4个TT减速电机安装在如图2-4中所示的“电机”位置。

图 2-4 底盘结构图



说明

TT减速电机主体与固定架安装请参见减速机安装说明书。

步骤3 安装麦克纳姆轮，利用TT减速电机自带的黑色或银色固定架固定好四个电机后，即可安装四个麦克纳姆轮。如图2-5和图2-6所示，对准轮子中心位置拧入自攻螺丝并对准TT电机的白色传动轴，拧紧即可完成电机和车轮的固定。

图 2-5 安装麦克纳姆轮

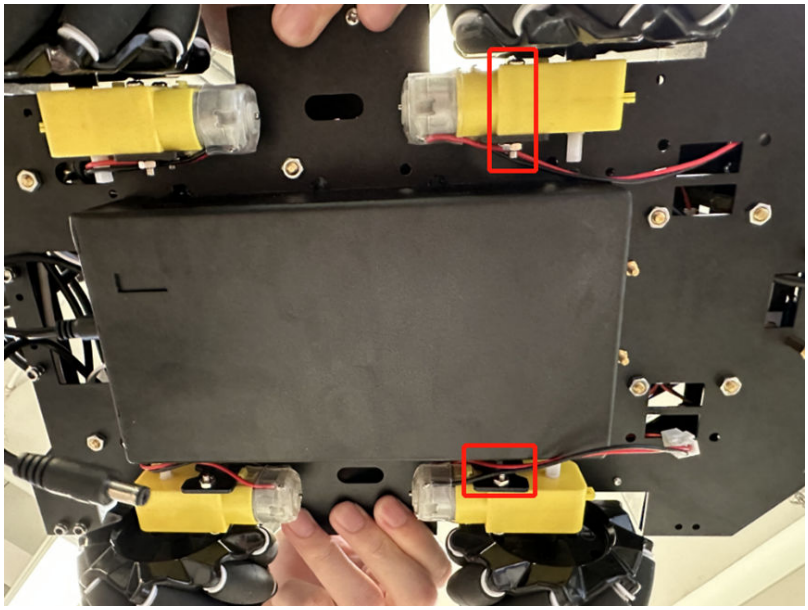
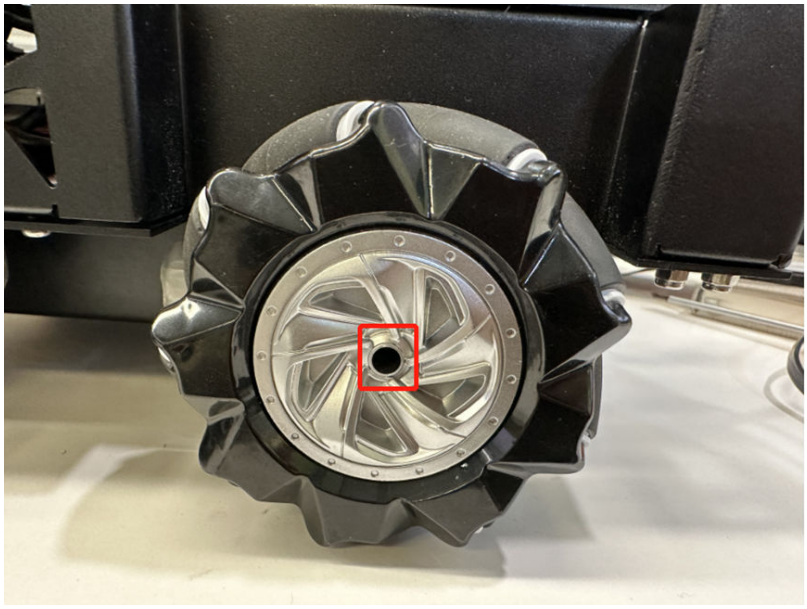


图 2-6 麦克纳姆轮示例图



说明

图中底盘靠近车头部分的四个矩形孔洞为预留出的电机走线口，需使用XH2.542pin线连接到ESP32单片机。

步骤4 将云台的自带摄像头拆下，替换为118度广角摄像头模组，并拧入固定螺丝和防松螺母，如图2-7和图2-8所示。

图 2-7 安装摄像头模组

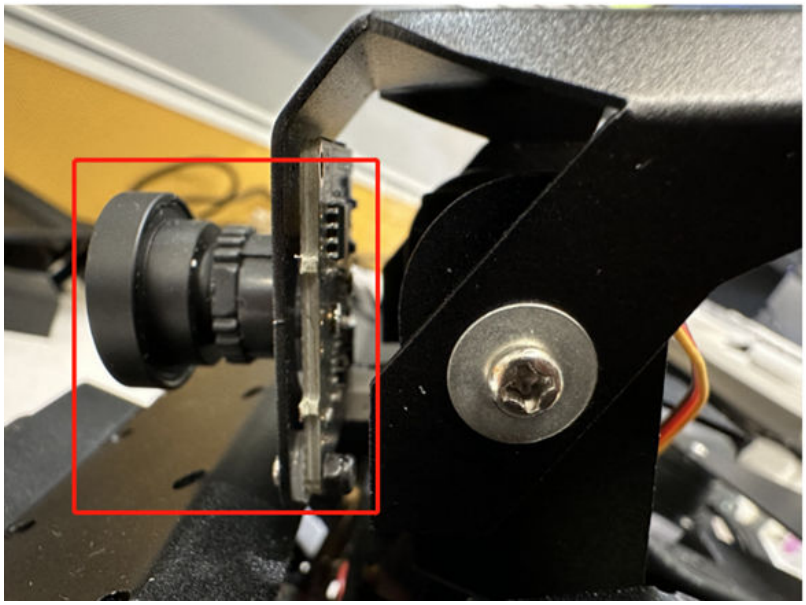
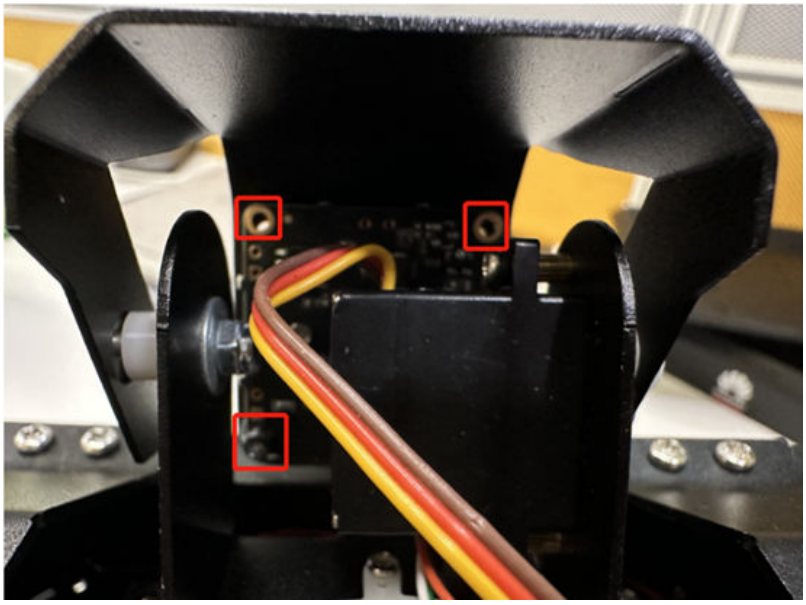


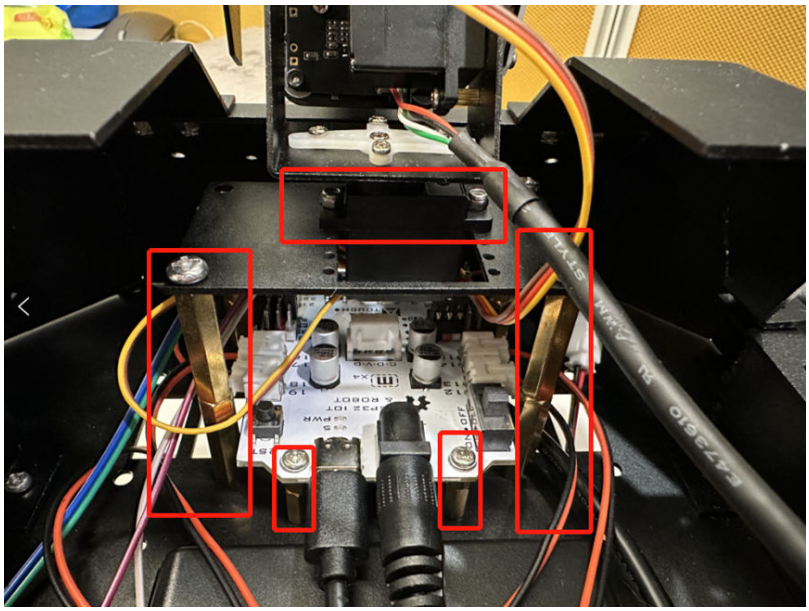
图 2-8 固定摄像头模组



步骤5 安装ESP32单片机。

1. 将ESP32单片机用4根铜柱固定在如图2-4和中所示位置，将步骤2中的四个电机的连接线连接到ESP对应的接口上，具体接口号和位置见表1-2。
2. 将两轴云台上部分舵机控制线连接到ESP32的26号引脚接口上，棕色线接地，靠近5V电源一侧。下部分舵机控制线连接到ESP32的25号引脚接口上，棕色线接地，靠近5V电源一侧。
3. 将组装好的云台与铜柱固定，如图2-9所示。

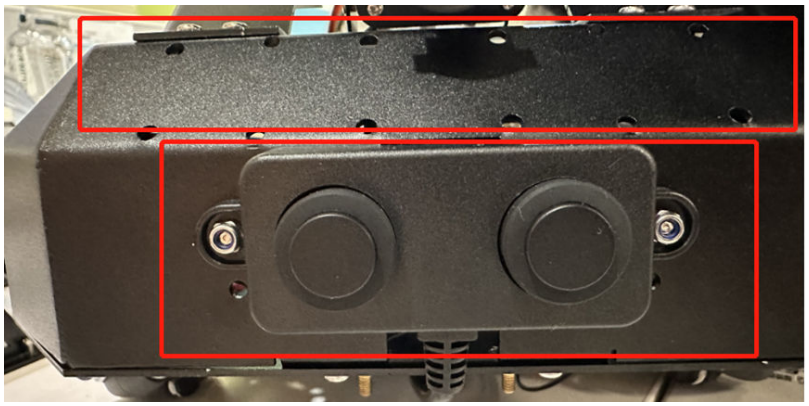
图 2-9 固定云台与 ESP32 单片机



步骤6 安装超声波模块/语音模块。

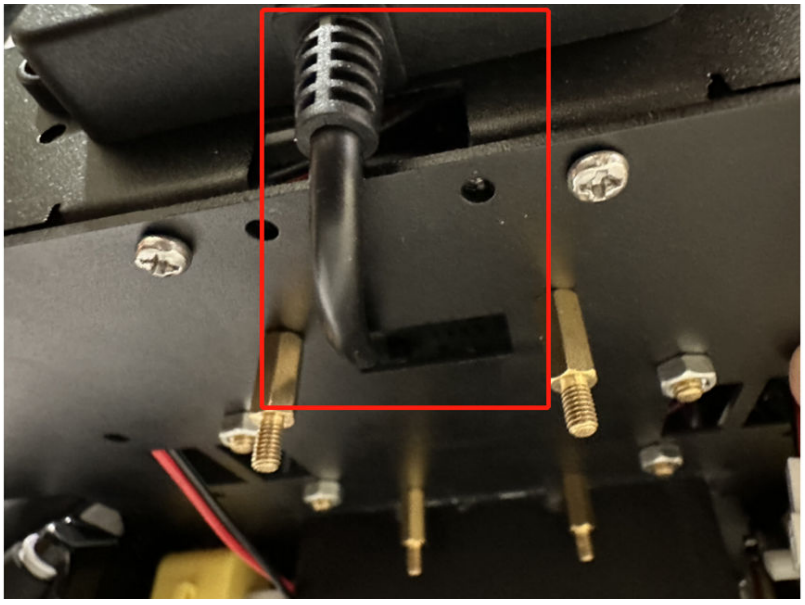
1. 将超声波模块/语音模块固定至第一层外壳，如图2-10所示，红框中部分的矩形孔洞为超声波模块的走线口，上方的12个孔洞为预留出语音麦克风与扬声器模块固定孔。

图 2-10 安装超声波模块/语音模块



2. 将超声波模块的PWM接口连接4根杜邦线之后连接到ESP32，具体四个PWM接头信息参照超声波模块购买链接中的参数说明与ESP32的引脚说明，对应连接即可，底盘的矩形孔洞是预留的超声波模块走线口，如图2-11所示。

图 2-11 超声波模块走线口



3. 将第一层外壳固定在小车底盘。

说明

在底盘上留有多个固定孔位，根据外壳制作工艺不同，固定孔的精确度会有较大区别，所以预留较多孔位，实际组装时固定其中的半数位置即可，同时也方便拆卸。如果外壳使用铝合金钣金加工，且并未开模制造，固定孔位置会与三维模型有较大精度差，需要适当弯折板材固定。

步骤7 在小车后部电池两侧的四个位置固定四个铜柱，如图2-12所示。

1. 使用一根TypeC-USB线连接ESP32和开发者套件，TypeC端插入ESP32单片机，USB端插入开发者套件USB接口。
2. 将无线路由器放入小车内部，将网线插入LAN口，再将电源线插入无线路由器电源接口。
3. ESP电源线连接ESP32和电源的9V输出口。

4. 将USB扩展板放入小车内部，四个USB接口中任意两个连接无线路由器电源线和广角摄像头USB接口，另外两个接口预留给激光雷达及语音控制模块。
5. 将以上模组的连接线收纳在承载板下方的空间，安装开发者套件承载板，并在承载板上安装开发者套件（参考Atlas 200I DK A2 开发者套件《**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）。

图 2-12 固定铜柱

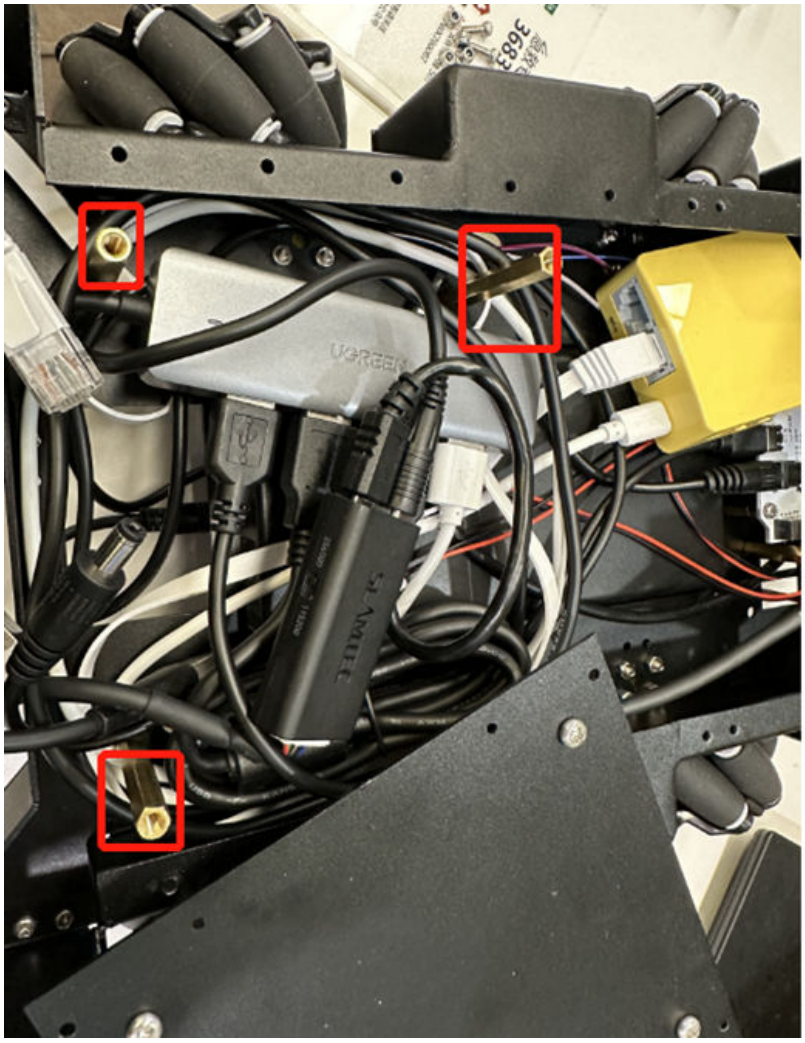
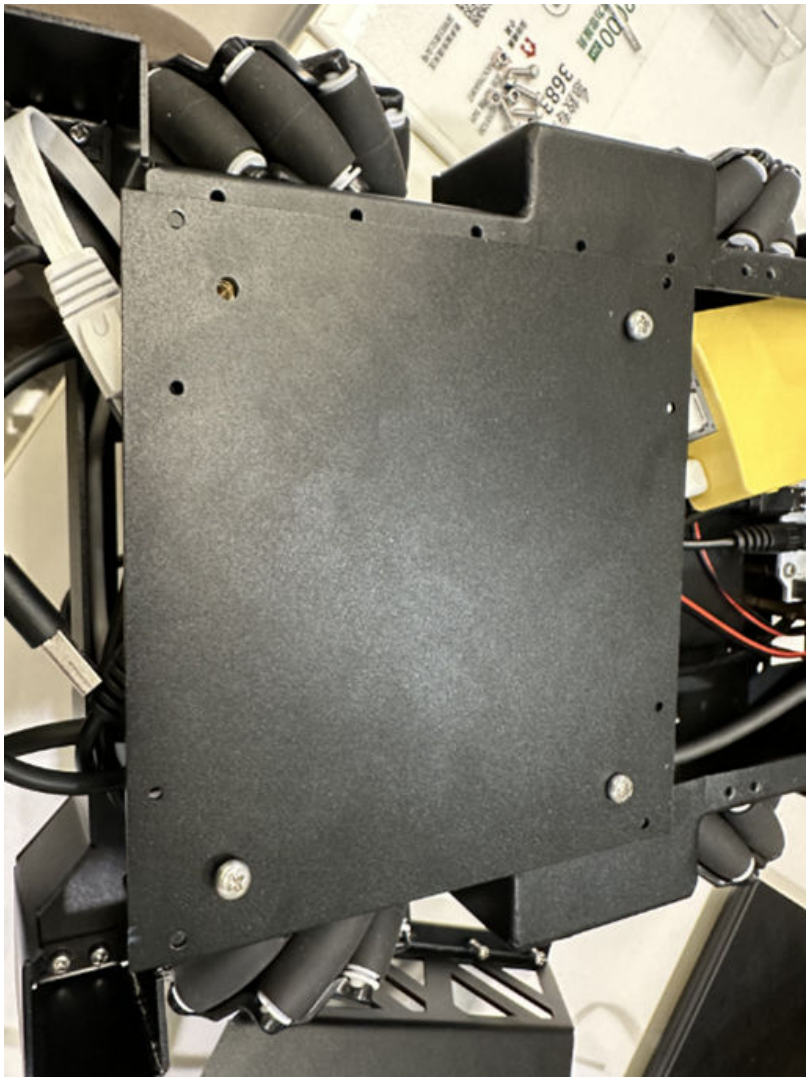
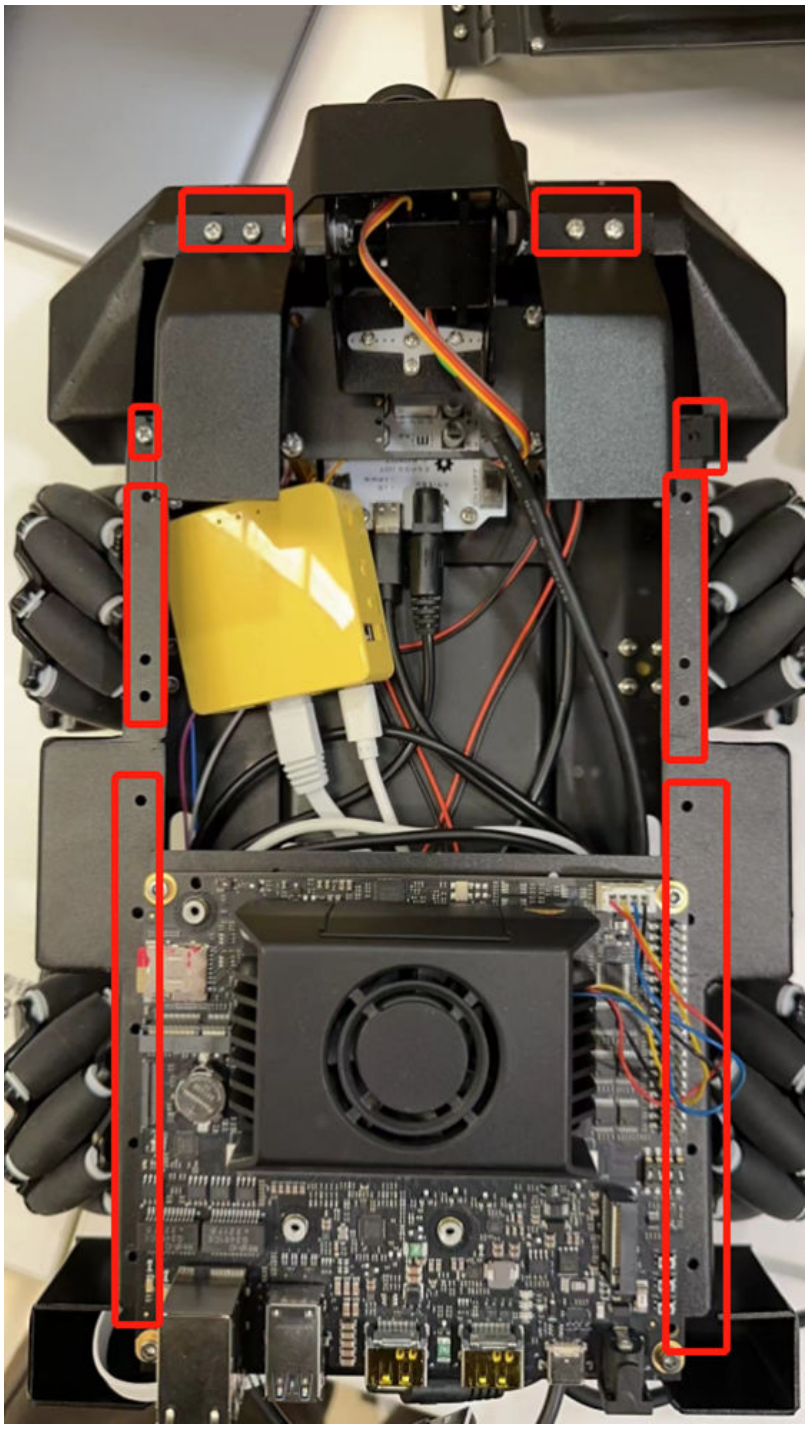


图 2-13 安装开发者套件承载板



步骤8 将开发者套件板按照对应螺丝孔位置固定到承载板上，并且将USB扩展板与ESP32连接到开发者套件USB口上，无线路由器的网线连接到开发者套件的eth0网口，再将电源线连接开发者套件和电源12V输出口即可。

步骤9 将小车的前中后段外壳按照下图中的固定孔位固定在外壳上后即可。



----结束

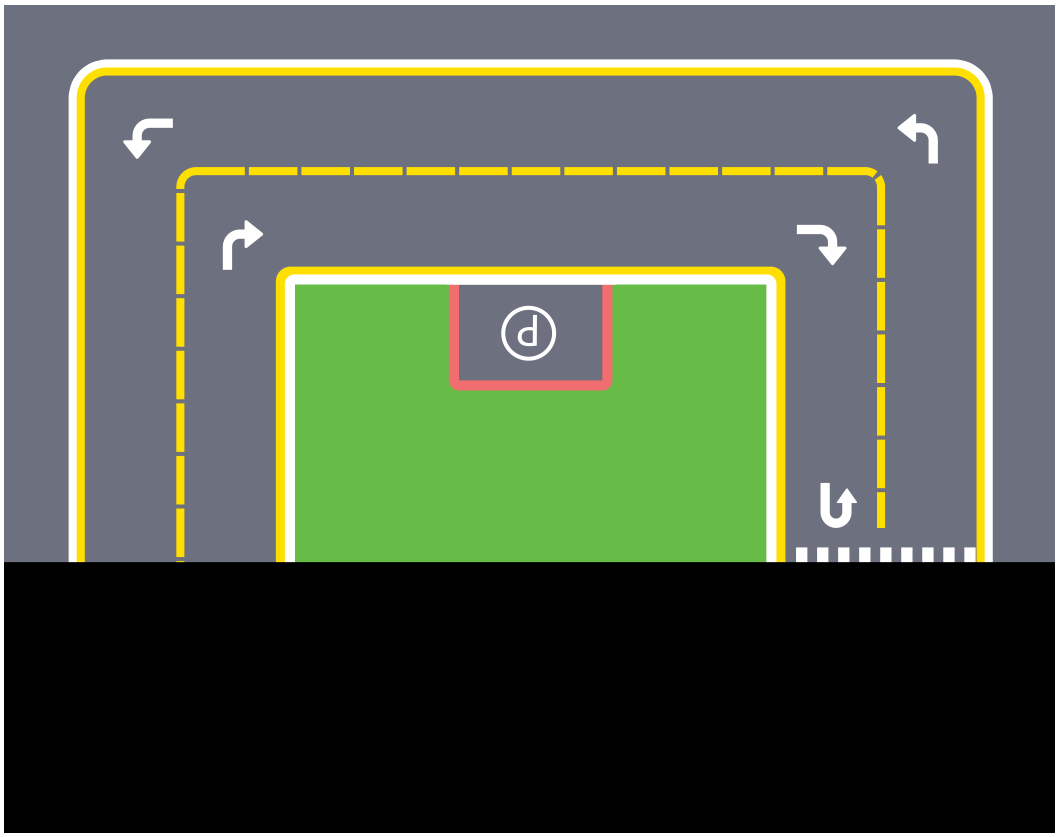
2.5 地图绘制

小车的行驶依赖于模型的训练，因此需要设计一张地图，用于小车行驶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。

当前地图样例包含以下行驶场景，可供小车沿地图“8”字行驶。

- 直线行驶场景，小车保持直线行驶。一条道路上有两条相向的车道，并用黄色虚线分隔开来。车道两侧有黄白双色的车道线作为指示。
- 弯道转向场景，小车根据转向方位指示标行驶。
- 自动停车场景，小车在停车标识处，自动泊车至车位。
- 十字路口场景，小车在人行道处可进行目标检测，是否有行人通过，进而判断前进或停车等待。
- 路标识别场景，当前地图可放置各类玩具标识（未在图中标出），例如放置一个调头的标志，小车识别后调头转换方向行驶。

图 2-14 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小车打滑，难以精确地控制小车。另外光滑的表面会产生强烈的反光，影响小车通过摄像头采集的画面质量，进而干扰各个图像识别的任务。
- 如果小车的外观尺寸变更，开发者应修改地图的尺寸以与之相匹配。建议一条车道的宽度为1.5倍的小车车体宽度，或者更宽。

3 运行环境准备

3.1 烧录镜像

参见**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。**

3.2 连接路由器

步骤1 确保路由器和开发者套件已按**步骤7**与**步骤8**中所述连接。

步骤2 使用PC连接车载无线路由器提供的WiFi，SSID为GL-MT300N-V2-177。

步骤3 参见**通过路由器联网**，查询开发者套件IP地址。

步骤4 使用**root**用户（默认密码为**Mind@123**）远程登录开发者套件，登录IP为**步骤3**中查询到的IP。

----结束

3.3 配置 ESP32 开发板烧录软件

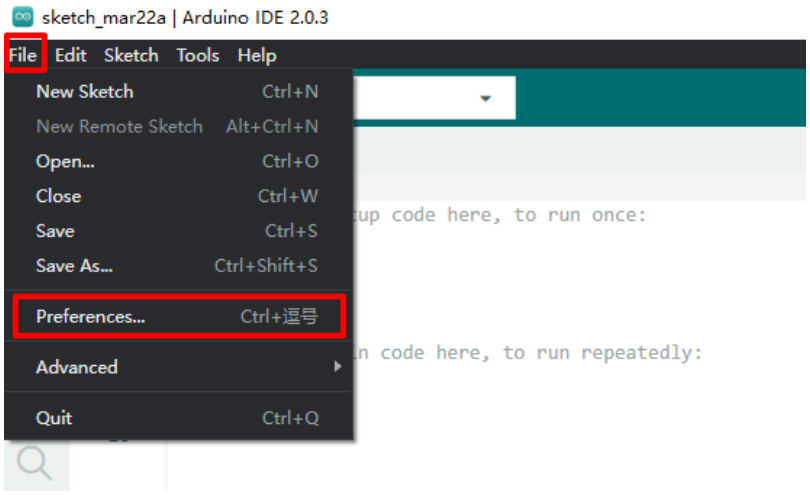
Arduino是一套便捷、灵活、容易上手的硬件开发平台，它包括多种型号的Arduino控制电路板和专用编程开发软件，能帮助用户快速的开发出智能硬件原型。

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。

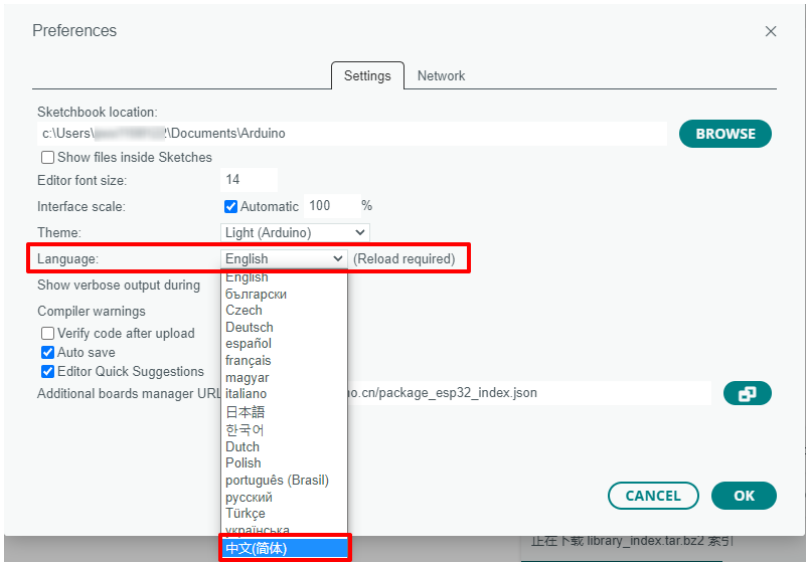
1. 单击“File > Preferences”，如**图3-1**所示。

图 3-1 软件菜单



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

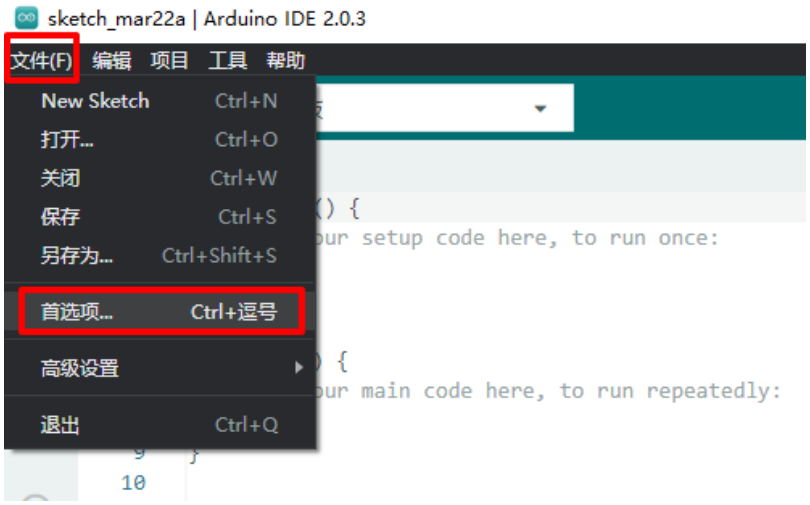
图 3-2 修改语言



步骤3 安装ESP32开发板。

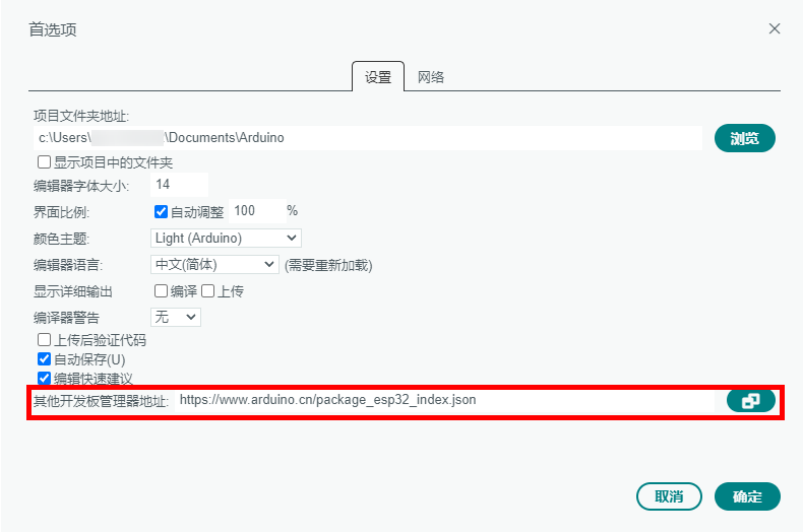
1. 单击“文件 > 首选项”，如图3-3所示。

图 3-3 中文软件菜单



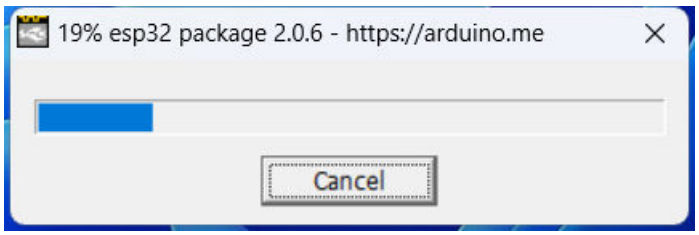
2. 在“其他开发板管理器地址”中输入“https://www.arduino.cn/package_esp32_index.json”与“https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json”或单击  图标在文本框中输入开发板管理器地址，单击确定保存，如图3-4所示。

图 3-4 添加开发板管理器地址



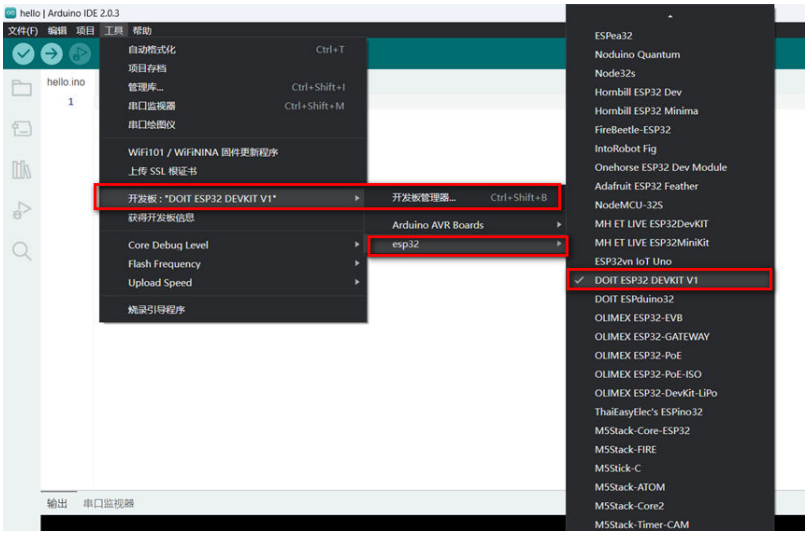
3. 离线安装ESP32开发板，单击链接下载Arduino的ESP32开发板安装包，下载完成后双击安装文件进行默认安装，安装完成后重启Arduino IDE。

图 3-5 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3-6所示。

图 3-6 选择开发板

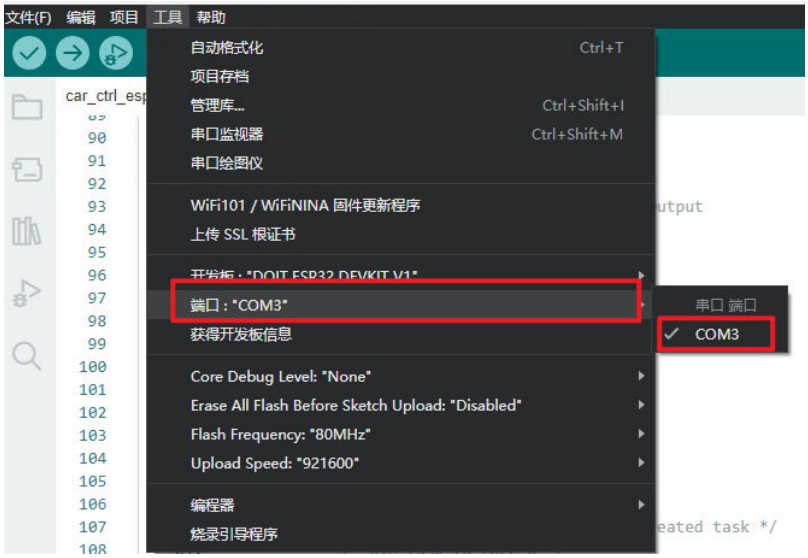


说明

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3-7所示。

图 3-7 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 获取代码

获取代码

- 获取ESP32开发板代码。

  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开发板的代码目录“ascend-devkit-master\src\E2E-Sample\Car\ESP32”。

- 获取小车项目代码。

  a. 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
  ```

  b. 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/Car/
  ```

  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/Car/”目录。

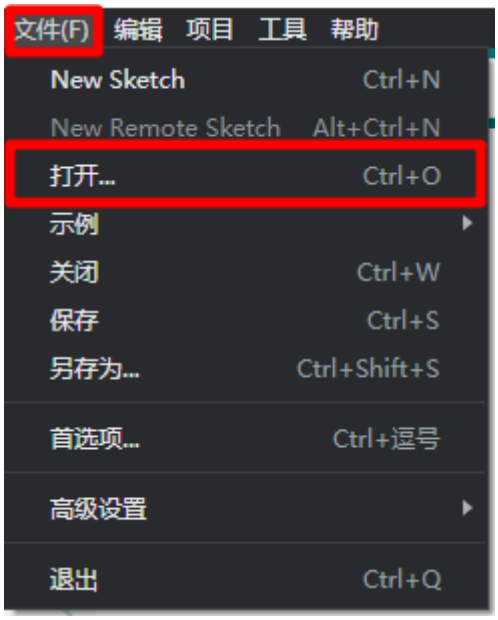
配置功能库文件

将Car/ESP32/下的“ESP32_Servo.cpp”、“ESP32_Servo.h”、“RL_ESP32_Motor.cpp”、“RL_ESP32_Motor.h”四个文件复制到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“Car/ESP32/car_ctrl_esp32.ino”文件打开。

图 3-8 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3-9和图3-10所示。

图 3-9 烧录代码

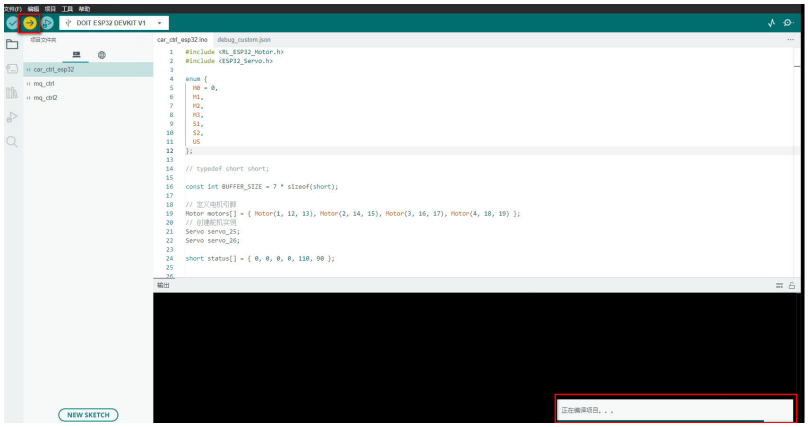
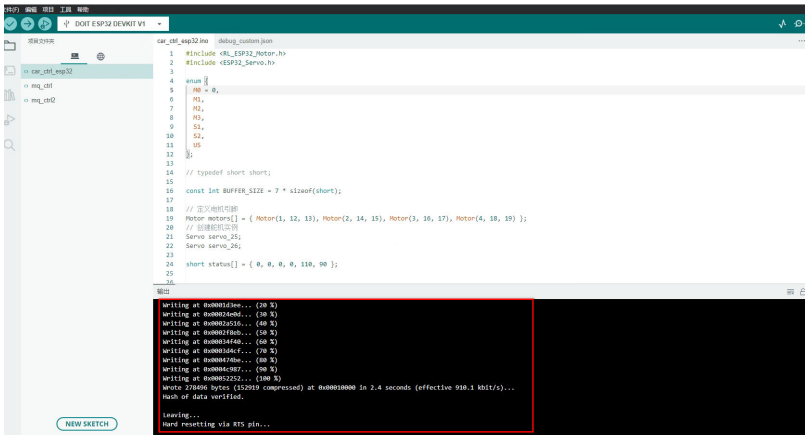


图 3-10 烧录回显



----结束

4 快速体验

说明

本章节为内置的快速体验方案和脚本，若用户在开发场景中需要识别或追踪其他目标，可以先参照《**使用模型适配工具生成推理应用**》使用模型适配工具打包新的目标检测推理模型并在开发者套件上构建推理过程。

4.1 手动控制小车

步骤1 将小车放置在运动场地。

步骤2 以**root**用户远程登录小车的开发者套件。

步骤3 进入**3.4 获取代码**中的样例代码目录下的“/python”文件夹路径，安装相关依赖后启动运行脚本。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/Car/python
pip3 install -r requirements.txt
python3 main.py
```

回显如下：

```
(base) root@davinci-mini:/home/HwHiAiUser/E2ESamples/src/E2E-Sample/Car/python# python main.py
[EVENT] PROFILING(5821,python):2023-03-27-02:24:50.860.329 [msprof_callback_impl.cpp:236] >>>
(tid:5821) Started to register profiling ctrl callback.
2023-03-27 02:24:52 [MainProcess:5821][INFO]: start
2023-03-27 02:24:52 [Process-2:5841][INFO]: update_controller_speed: True
2023-03-27 02:24:52 [Process-2:5841][INFO]: self.speed: 0
2023-03-27 02:24:52 [Process-2:5841][INFO]: [0, 0, 0, 0, 90, 65, -12345]
2023-03-27 02:24:52 [Process-2:5841][INFO]: action SetServo execute SUCC
2023-03-27 02:24:52 [Process-2:5841][INFO]: Manual loop start
```

说明

运行**python3 main.py**报错请参见**运行智能车主入口程序main.py时出现‘ttyUSB0’或'ttyUSB1'相关错误导致程序无法启动**解决。

步骤4 通过键盘输入键位手动控制小车移动，如**表4-1**所示。

表 4-1 键位映射表

键位	动作
w	前进
a	左转
s	后退
d	右转
q	逆时针旋转
e	顺时针旋转
↑	加速(小车运动状态与静止状态均可以调整)
↓	减速(小车运动状态与静止状态均可以调整)
c	捕获当前摄像头的图片
space	暂时停车

说明

由于TT减速电机成本较低，若此处手动控制小车出现轮子转向相反，或四个轮子转速不同导致小车不走直线的情况。

- 参照FAQ中的E2E样例开发问题。
- 参考“/Car/src/actions/base_action.py”文件代码中的注释修改“/Car/src/actions/base_action.py”的'self.motor_rating'参数，四个参数依次对应图1-9中的1、2、3、4号电机，修改参数的比例大小即可修改对应电机的转速，添加负号即可使对应电机反转，进行微调即可使小车恢复正常。

----结束

4.2 自动驾驶与泊车

步骤1 将小车放置在运动场地。

步骤2 以**root**用户远程登录小车的开发者套件。

步骤3 进入**3.4 获取代码**中的样例代码目录，启动运行脚本。

```
cd /home/HwHiAiUser/E2ESamples/src/E2E-Sample/Car/python
python3 main.py --mode=easy
```

回显如下：

```
(base) root@davinci-mini:/home/HwHiAiUser/E2ESamples/src/E2E-Sample/Car/python# python main.py --mode=easy
[EVENT] PROFILING(5821,python):2023-03-27-02:24:50.860.329 [msprof_callback_impl.cpp:236] >>> (tid:5821) Started to register profiling ctrl callback.
2023-03-27 02:24:52 [MainProcess:5821][INFO]: start
2023-03-27 02:24:52 [Process-2:5841][INFO]: update_controller_speed: True
2023-03-27 02:24:52 [Process-2:5841][INFO]: self.speed: 0
2023-03-27 02:24:52 [Process-2:5841][INFO]: [0, 0, 0, 0, 90, 65, -12345]
2023-03-27 02:24:52 [Process-2:5841][INFO]: action SetServo execute SUCC
2023-03-27 02:24:52 [Process-2:5841][INFO]: Manual loop start
```

> **说明**
>
> 运行**python3 main.py**报错请参见**运行智能车主入口程序main.py时出现‘ttyUSB0’或'ttyUSB1'相关错误导致程序无法启动**解决。

步骤4 在命令行中输入**Helper**，加载转弯及停车辅助模型。无需等待，继续在命令行中输入**LF**，加载直线矫正模型。

步骤5 小车开始自动驾驶，待行驶至停车标识处会自动进行泊车。

----结束

4.3 目标跟踪

> **说明**
>
> 运行场地尽量为夜间且照明充足的室内，避免由太阳光放射带来的曝光影响。

步骤1 将小车放置在运动场地，并将手动控制的另一辆小车放置在智能小车前方位于摄像头画面的中下方位置。

步骤2 以**root**用户远程登录小车的开发者套件。

步骤3 进入**3.4 获取代码**中的样例代码目录，启动运行脚本。

```
cd /home/HwHiAiUser/+Samples/src/E2E-Sample/Car/python
python3 main.py -mode=cmd
```

回显如下：

```
(base) root@davinci-mini:/home/HwHiAiUser/E2ESamples/src/E2E-Sample/Car/python# python main.py -mode=cmd
[EVENT] PROFILING(5821,python):2023-03-27-02:24:50.860.329 [msprof_callback_impl.cpp:236] >>> (tid:5821) Started to register profiling ctrl callback.
2023-03-27 02:24:52 [MainProcess:5821][INFO]: start
2023-03-27 02:24:52 [Process-2:5841][INFO]: update_controller_speed: True
2023-03-27 02:24:52 [Process-2:5841][INFO]: self.speed: 0
2023-03-27 02:24:52 [Process-2:5841][INFO]: [0, 0, 0, 0, 90, 65, -12345]
2023-03-27 02:24:52 [Process-2:5841][INFO]: action SetServo execute SUCC
2023-03-27 02:24:52 [Process-2:5841][INFO]: Manual loop start
```

> **说明**
>
> 运行**python3 main.py**报错请参见**运行智能车主入口程序main.py时出现‘ttyUSB0’或'ttyUSB1'相关错误导致程序无法启动**解决。

步骤4 在命令行中输入**Tracking**，进入追踪模式。

步骤5 开始控制前方小车移动，智能小车会追踪前方小车的移动轨迹和速度进行目标跟踪移动。

----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
python/main.py	小车demo运行入口文件。
python/requirements.txt	智能小车样例代码运行所需依赖。
python/src/actions	小车基础与复杂运动代码。
python/src/utils	工具类python文件包含OpenCV、acl工具等。
python/src/scenes	小车预设场景相关代码。
python/src/model	推理模型相关代码。
python/src/weight	模型权重文件。

5.2 智能小车控制逻辑入口

“main.py”是小车控制代码的主入口，定义了小车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，以及启动摄像头的关键模块。

```
import os
from argparse import ArgumentParser
from multiprocessing import Process, Queue
from src.scenes import Manual, scene_initiator
from src.utils import getkey, log, CameraBroadcaster, CAMERA_INFO
```

定义出可选参数更改小车的控制模式，可选命令行cmd控制，手动控制，声音控制（正在规划中）以及测试等。

```
def parse_args():
    parser = ArgumentParser()
    parser.add_argument('--mode', type=str, required=False, default='manual',
                        choices=['cmd', 'voice', 'manual', 'test'])
    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，包含摄像头端口的调用及行驶过程中各类消息队列的处理。

```
if __name__ == '__main__':
    args = parse_args()
    log.info('start')
    msg_queue = Queue(maxsize=1)
    camera = CameraBroadcaster(CAMERA_INFO)
    shared_memory_name = camera.memory_name
    camera_process = Process(target=camera.run)
    camera_process.start()
    if args.mode == 'manual':
        task = Manual(shared_memory_name, CAMERA_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':
                    for p in process_list:
                        p.kill()
                    log.info(f'start put stop sign')
                    camera.stop_sign.value = True
                    camera_process.join()
                    break
                elif command == 'clear':
                    for p in process_list:
                        p.kill()
                    process_list.clear()
                    log.info(f'clear succ')
                    continue
                elif command == 'Manual':
                    log.error(f'Does not support switching from cmd mode to manual mode')
```



```
                continue
            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, CAMERA_INFO, msg_queue)
                process = Process(target=scene_obj.loop)
                process.start()
                process_list.append(process)

    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 == 'test':
    test_list = []
    test1 = Manual(shared_memory_name, CAMERA_INFO, msg_queue)
    test_list.append(Process(target=test1.loop))
    test2 = scene_initiator('Tracking')(shared_memory_name, CAMERA_INFO, msg_queue)
    test_list.append(Process(target=test2.loop))
    test3 = scene_initiator('Tracking')(shared_memory_name, CAMERA_INFO, msg_queue)
    test_list.append(Process(target=test3.loop))

    for process in test_list:
        process.start()
    try:
        while True:
            key = getkey()
            if key == 'esc':
                for process in test_list:
                    process.kill()
                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.')
```

5.3 智能小车运动控制代码

“python/src/actions”文件夹中包含的是与烧录到ESP32控制端对应的小车基础和复杂运动的python代码文件以及其中的函数说明

1. 小车动作函数定义。

```
import time
from abc import ABC, abstractmethod


class BaseAction(ABC):
    """
    基础动作的基类，所有基本动作均继承于该类
    """

    def __init__(self, *args, **kwds) -> None:
        """
        基础动作类的初始化方法，通过args与kwds控制输入参数
        :param args:
        :param kwds:
```

```
        """
        # 抽象的速度信息
        self.speed = kwds.get('speed', -1)
        # 电机角度
        self.servo_angle = kwds.get('servo', [-1, -1])

        # 根据电机的实际情况修改下发到电机的速度
        self.motor_rating = [1.45, 1, 1, 1]

        # 确定是否需要在运行时根据前动作更新电机角度及电机速度
        self.update_speed = False
        self.update_servo = False

        if self.speed == -1:
            self.update_speed = True

        if self.servo_angle[0] == -1 and self.servo_angle[1] == -1:
            self.update_servo = True

        # 由速度生成方法将抽象的总体速度计算为4个电机的速度并输出为list
        self.speed_setting = self.generate_speed_setting(self.speed)
        self.fix_speed()

    def fix_speed(self):
        self.speed_setting = [int(speed * ratio) for speed, ratio in zip(self.speed_setting,
self.motor_rating)]

    @staticmethod
    @abstractmethod
    def generate_speed_setting(speed, degree=0):
        """
        生成4个电机的速度，并输出为列表
        抽象类，需要根据具体情况进行设置
        :param speed: 抽象的速度。 当前动作初始化时设置 或 控制器根据前一动作速度进行设置
        :param degree: 如需转弯，速度计算需要的角度信息
        :return:
        """
        pass

    def __call__(self, speed, servo_angle):
        """
        call魔法函数，两个输入参数由控制器输入
        当init方法设置了相关信息，则忽略控制器输入的参数
        当init方法没有设置相关信息，相关信息的将由控制器输入的参数进行更新

        :param speed: 抽象速度
        :param servo_angle: 舵机的角度
        :return: 长度为6的列表，前4位为4个电机的速度，后2位为舵机的两个角度
        """
        if self.update_servo:
            self.servo_angle = servo_angle
        if self.update_speed:
            degree = 0
            if hasattr(self, 'degree'):
                degree = self.degree
            self.speed_setting = self.generate_speed_setting(speed, degree)
            self.fix_speed()

        return self.speed_setting + self.servo_angle


class Advance(BaseAction):
    """
    小车前进
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [-speed, -speed, speed, speed]
```



```
class BackUp(BaseAction):
    """
    小车后退
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [speed, speed, -speed, -speed]


class CustomAction(BaseAction):
    """
    自定义动作
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.speed_setting = kwds.get('motor_setting', [0, 0, 0, 0])
        self.update_controller_speed = False
        self.update_speed = False

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [0, 0, 0, 0]


class Stop(BaseAction):
    """
    小车停止
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.speed = 0

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [0, 0, 0, 0]


class TurnLeft(BaseAction):
    """
    小车左转
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.degree = kwds.get('degree', 0)
        self.speed_setting = self.generate_speed_setting(speed=self.speed, degree=self.degree)

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [-speed, -speed, int(speed * (1 + degree)), int(speed * (1 + degree))]


class TurnRight(BaseAction):
    """
    小车右转
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.degree = kwds.get('degree', 0)
        self.speed_setting = self.generate_speed_setting(speed=self.speed, degree=self.degree)

    @staticmethod
    def generate_speed_setting(speed, degree=0):
```



```
        return [-int(speed * (1 + degree)), -int(speed * (1 + degree)), speed, speed]


class ShiftLeft(BaseAction):
    """
    向左平移
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.motor_rating = [1.5, 1.3, 1.25, 1.25]
        self.fix_speed()

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [speed, -speed, -speed, speed]


class ShiftRight(BaseAction):
    """
    向右平移
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [-speed, speed, speed, -speed]


class LeftOblique(BaseAction):
    """
    斜向左前方
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [0, -speed, 0, speed]


class RightOblique(BaseAction):
    """
    斜向右前方
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [-speed, 0, speed, 0]


class SpinClockwise(BaseAction):
    """
    顺时针旋转
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [-speed] * 4


class SpinAntiClockwise(BaseAction):
    """
    逆时针旋转
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [speed] * 4


class SetServo(BaseAction):
```



```
    """
    舵机转动
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.speed = 0

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [0, 0, 0, 0]


class Sleep(BaseAction):
    """
    Sleep(1)等同于time.sleep(1)
    可加入至动作序列进行使用
    """

    def __init__(self, *args, **kwds):
        super().__init__(*args, **kwds)
        self.sleep_time = args[0]

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return []

    def __call__(self, speed, servo_angle):
        time.sleep(self.sleep_time)
        return None
```

2. 以逆时针旋转的操作为例，方法首先继承于运动的基类，实现运动动作的方式是修改不同位置电机的旋转方向和转速。

```
class SpinAntiClockwise(BaseAction):
    """
    逆时针旋转
    """

    @staticmethod
    def generate_speed_setting(speed, degree=0):
        return [speed] * 4
```

3. 复杂动作则以掉头行驶为例，是多种不同的简单动作的组合，首先要导入所有的简单动作。

```
from abc import ABC

from src.actions.base_action import Advance, Sleep, SpinAntiClockwise, Stop, SpinClockwise,
CustomAction
```

4. 再利用简单动作形成复杂行驶动作。

```
class TurnAround(ComplexAction):
    def __init__(self):
        super().__init__()
        self.action_seq = [
            Stop(),
            Sleep(0.5),
            Advance(speed=30),
            Sleep(0.35),
            Stop(),
            Sleep(0.3),
            SpinAntiClockwise(speed=50),
            Sleep(0.55),
            Advance(speed=30),
            Sleep(1.2),
            SpinAntiClockwise(speed=50),
            Sleep(0.55),
            Stop(),
            # Advance(speed=35),
        ]
```


5.4 手动控制逻辑代码

“python/src/scenes/manual.py”为手动控制小车的核心部分。

1. 首先导入所有必须模块和基础运动模块。

```
import datetime
import os
import cv2
import numpy as np
from src.actions import Advance, Stop, SetServo, TurnLeft, TurnRight, SpinClockwise,
SpinAntiClockwise, BackUp, \
    ShiftLeft, ShiftRight, CustomAction
from src.actions.complex_actions import ComplexAction, TurnAround
from src.scenes.base_scene import BaseScene
from src.utils import log
```

2. 基于场景的基类构建手动控制小车场景的手动类，初始化后，进入到loop循环的函数中，不断等待键盘输入的键值，再执行键值对应的指令

```
def loop(self):
        ret = self.init_state() # 执行初始化
        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) # 拉取
共享内存中的图片
        log.info(f'{self.__class__.__name__} loop start')
        last_action = SetServo(servo=[90, 65]) # 设置舵机角度

        while True:
            try:
                if not self.msg_queue.empty():
                    key = self.msg_queue.get()
                else:
                    continue
            except KeyboardInterrupt:
                self.ctrl.execute(Stop()) #捕获SIGINT之后停止小车
                break

            degree = 0
            if key == 'up':
                self.speed = min(self.speed + 1, 60) #加速
            elif key == 'down':
                self.speed = max(self.speed - 1, 25) #减速
            elif key == 'left':
                last_action = ShiftLeft() #左平移
            elif key == 'right':
                last_action = ShiftRight() #右平移
            elif key == 'w':
                last_action = Advance() #前进
            elif key == 'a':
                last_action = TurnLeft() #左转
                degree = 1.1
            elif key == 's':
                last_action = BackUp() #后退
            elif key == 'd':
                last_action = TurnRight() #右转
                degree = 1.1
            elif key == 'q':
                last_action = SpinAntiClockwise() #逆时针旋转
            elif key == 'e':
                last_action = SpinClockwise() #顺时针旋转
            elif key == 'space':
                last_action = Stop() #停车
            elif key == 'esc':
                self.ctrl.execute(Stop()) #退出循环前停下小车
                break
```



```
            elif key == 'c':
                save_img = frame.copy()
                cv2.imwrite(os.path.join(self.save_dir, f'{datetime.datetime.now()}.jpg'), save_img) #保存当
前摄像头中的画面
                log.info(f'image saved.')
            elif key == 't':
                last_action = CustomAction(motor_setting=[-62, 50, 50, -50])
            elif key == 'r':
                last_action = CustomAction(motor_setting=[55, -50, -50, 50])
            elif key == 'z':
                last_action = TurnAround() #掉头
            else:
                continue

            if not isinstance(last_action, ComplexAction) and not isinstance(last_action, CustomAction):
                last_action.update_speed = False
                last_action.speed_setting = last_action.generate_speed_setting(speed=self.speed,
degree=degree)
                last_action.fix_speed()
            self.ctrl.execute(last_action)
```

5.5 目标检测模型代码

“python/src/models/yolov5.py”为yolov5模型的定义代码，为小车的基础运行提供核心的智能目标识别与检测功能。

1. 示例代码定义了如何重塑图片的尺寸，并计算需要零值填充大小的功能。

```
def letterbox(img, new_shape=(640, 640), color=(114, 114, 114), auto=False, scaleFill=False,
scaleup=True):
    # Resize image to a 32-pixel-multiple rectangle https://github.com/ultralytics/yolov3/issues/232
    shape = img.shape[:2]  # current shape [height, width]
    if isinstance(new_shape, int):
        new_shape = (new_shape, new_shape)

    # Scale ratio (new / old)
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])
    if not scaleup:  # only scale down, do not scale up (for better test mAP)
        r = min(r, 1.0)

    # Compute padding
    ratio = r, r  # width, height ratios
    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))
    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]  # wh padding
    if auto:  # minimum rectangle
        dw, dh = np.mod(dw, 64), np.mod(dh, 64)  # wh padding
    elif scaleFill:  # stretch
        dw, dh = 0.0, 0.0
        new_unpad = (new_shape[1], new_shape[0])
        ratio = new_shape[1] / shape[1], new_shape[0] / shape[0]  # width, height ratios

    dw /= 2  # divide padding into 2 sides
    dh /= 2

    if shape[::-1] != new_unpad:  # resize
        img = cv2.resize(img, new_unpad, interpolation=cv2.INTER_LINEAR)
    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))
    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))
    img = cv2.copyMakeBorder(img, top, bottom, left, right, cv2.BORDER_CONSTANT, value=color)  #
add border
    return img, ratio, (dw, dh)
```

2. Yolov5的模型定义，以及推理的实现过程，形成最终的推理结果目标框和对应的类别名称。

```
class YoloV5(Model):
    def __init__(self, model_path, acl_init=True):
        super().__init__(model_path, acl_init)
        self.neth = 640
```



```
        self.netw = 640
        self.conf_threshold = 0.1
        dic = {0: 'left',
               1: 'right',
               2: 'stop',
               3: 'turnaround'}
        self.names = ['person', 'sports_ball', 'bicycle', 'motorcycle', 'car', 'bus', 'truck'] * 12
        self.object_list = ['person', 'sports_ball', 'bicycle', 'motorcycle', 'car', 'bus', 'truck']
        self.names = list(dic.values())
        self.object_list = list(dic.values())

    def infer(self, img_bgr):
        imgh, imgw = img_bgr.shape[0], img_bgr.shape[1]
        imginfo = np.array([self.neth, self.netw, imgh, imgw], dtype=np.float16)
        img_padding = letterbox(img_bgr, new_shape=(self.neth, self.netw))[0]  # padding resize bgr

        img = []

        img.append(img_padding)
        img = np.stack(img, axis=0)
        img = img[..., ::-1].transpose(0, 3, 1, 2)  # BGR tp RGB
        image_np = np.array(img, dtype=np.float32)
        image_np_expanded = image_np / 255.0
        img = np.ascontiguousarray(image_np_expanded).astype(np.float16) #将tensor的内存连续排列
        result = self.execute([img, imginfo]) #调用推理接口
        batch_boxout, boxnum = result

        pred_boxes = []
        idx = 0
        num_det = int(boxnum[idx][0])
        bbox = batch_boxout[idx][:num_det * 6].reshape(6, -1).transpose().astype(np.float32)  # 6xN ->
Nx6

        for idx, class_id in enumerate(bbox[:, 5]):
            obj_name = self.names[int(bbox[idx][5])]
            if not obj_name in self.object_list:
                continue
            confidence = bbox[idx][4]
            if float(confidence) < self.conf_threshold:
                continue
            x1 = int(bbox[idx][0])
            y1 = int(bbox[idx][1])
            x2 = int(bbox[idx][2])
            y2 = int(bbox[idx][3])

            pred_boxes.append([x1, y1, x2, y2, obj_name, confidence]) #获取推理结果

        return pred_boxes
```

5.6 目标追踪逻辑代码

在实现目标检测的前提下，结合小车的基础控制部分，将小车的速度调整依赖到目标检测的推理结果上，就能实现目标追踪。

1. “python/src/scenes/tracking.py”为目标追踪的核心代码，示例代码定义追踪的运行逻辑。

```
class Tracking(BaseScene):
    def __init__(self, memory_name, camera_info, msg_queue):
        super().__init__(memory_name, camera_info, msg_queue)
        self.model = None

    def init_state(self):
        log.info(f'start init {self.__class__.__name__}')
        model_path = os.path.join(os.getcwd(), 'weights', 'tracking.om')
        if not os.path.exists(model_path):
            log.error(f'Cannot find the offline inference model(.om) file needed for
{self.__class__.__name__}  scene.')
```



```
            return True
        self.model = YoloV5(model_path) #加载模型
        log.info(f'{self.__class__.__name__} model init succ.')
        self.ctrl.execute(SetServo(servo=[90, 65])) #设置舵机角度
        return False

    def loop(self):
        ret = self.init_state() #执行初始化
        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) #获取
共享内存中的图片
        log.info(f'{self.__class__.__name__} loop start')
        last_action = None
        last_not_seen = True
        forward_speed_slow = 30
        forward_speed_fast = 40
        while True:
            action = None
            if self.stop_sign.value:
                break
            if self.pause_sign.value:
                continue

            img_bgr = frame.copy()
            bboxes = self.model.infer(img_bgr)
            log.info(f'{bboxes}')
            if not bboxes:
                if last_not_seen:
                    action = Stop()
                else:
                    last_not_seen = True
                    continue
            else:
                if len(bboxes) > 1:
                    ori_box = sorted(bboxes, key=lambda x: x[-1], reverse=True)[0][:4]
                else:
                    ori_box = bboxes[0][:4]
                x1, y1, x2, y2 = ori_box
                x, y = (x1 + x2) // 2, (y1 + y2) // 2 #计算目标中心点的x与y坐标
                h, w = y2 - y1, x2 - x1 #计算目标的宽高

                if h * w < 141 * 128 or y < 110: #进行距离判断，如果过远就加速，否则减速
                    speed = forward_speed_fast
                else:
                    speed = forward_speed_slow

                if x < 400:
                    action = TurnLeft(degree=1.1, speed=speed) #左转
                elif x > 1000:
                    action = TurnRight(degree=1.1, speed=speed) #右转
                else:
                    action = Advance(speed=speed) #直行

                if h * w > 800 * 500 or y > 390: #如果距离过近则停车
                    action = Stop()

            if action is None or action == last_action:
                continue
            self.ctrl.execute(action)
            last_action = action
```

2. 初始化并导入Yolov5目标检测模型。

```
def __init__(self, memory_name, camera_info, msg_queue):
        super().__init__(memory_name, camera_info, msg_queue)
        self.model = None

    def init_state(self):
        log.info(f'start init {self.__class__.__name__}')
```

```
        model_path = os.path.join(os.getcwd(), 'weights', 'tracking.om')
        if not os.path.exists(model_path):
            log.error(f'Cannot find the offline inference model(.om) file needed for
{self.__class__.__name__}  scene.')
            return True
        self.model = YoloV5(model_path)
        log.info(f'{self.__class__.__name__} model init succ.')
        self.ctrl.execute(SetServo(servo=[90, 65]))
        return False
```

3. 在得到正确导入结果后，开启循环，不断获取推理结果，并根据结果估算智能小车和追踪目标之间的距离，再根据计算出的结果下达不同的运动指令，设置慢速和快速跟进的两个速度。

```
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')
        last_action = None
        last_not_seen = True
        forward_speed_slow = 30
        forward_speed_fast = 40
```

4. 获取推理结果的外接框。

```
bboxes = self.model.infer(img_bgr)
```

5. 计算出目标框的中心点的位置和目标框的宽高大小。

```
log.info(f'{bboxes}')
            if not bboxes:
                if last_not_seen:
                    action = Stop()
                else:
                    last_not_seen = True
                    continue
            else:
                if len(bboxes) > 1:
                    ori_box = sorted(bboxes, key=lambda x: x[-1], reverse=True)[0][:4]
                else:
                    ori_box = bboxes[0][:4]
                x1, y1, x2, y2 = ori_box
                x, y = (x1 + x2) // 2, (y1 + y2) // 2
                h, w = y2 - y1, x2 - x1
```

根据计算出的目标框的大小来判断小车和目标之间的距离，再调整小车的行进速度。根据目标近大远小的简单规则，存在两个判断条件，如果目标框的面积小于一定值，就说明小车与目标距离较远，需要快速接近目标，另外如果识别框的中心点的纵坐标大于0，也就是在摄像头视角里的上半部分，也说明小车距离目标较远，也需快速接近目标，反之亦然。

```
if h * w < 141 * 128 or y < 110:
                    speed = forward_speed_fast
                else:
                    speed = forward_speed_slow
```

6. 另外如果前方目标在小车的偏左或偏右的位置，也可以采用同样的判断方法，即判断目标框的中心点的横坐标落在小车摄像头视角画面中的左侧还是右侧，进而下发对应的微调转向的命令，实现跟踪目标的方向调整。

```
if x < 400:
                    action = TurnLeft(degree=1.1, speed=speed)
                elif x > 1000:
                    action = TurnRight(degree=1.1, speed=speed)
                else:
                    action = Advance(speed=speed)

                if h * w > 800 * 500 or y > 390:
                    action = Stop()
```


A 训练目标追踪功能模型

当前目标追踪功能，需要单独使用模型适配工具训练（模型适配工具的安装与使用请参见《**使用模型适配工具生成推理应用**》），用户可参见本节，进行模型的训练获得对应模型文件。

步骤1 收集待标记的png、jpg、JPEG、bmp、webp格式图片数据，推荐使用jpg格式。图片分辨率不高于1080P，单张图片不小于1MB，推荐使用小车上的摄像头进行图片收集，数量为200张以上且各角度均包含，并放置在全英文路径下。

注：图片名称不要带字符"."。

步骤2 为模型迁移准备数据集，进行图像标注，在模型适配工具界面选择“检测模型”。

1. 单击“打开目录”选择**步骤1**收集的数据集目录进行标注。
2. 单击  按钮，使用矩形框包围目标后单击鼠标左键，弹出添加标签界面，如**图A-1**所示。填写对应目标分类标签与Group ID号，当一个图片中有多个目标时需填写不同的ID号，单击“确定”完成标注。

图 A-1 添加标签

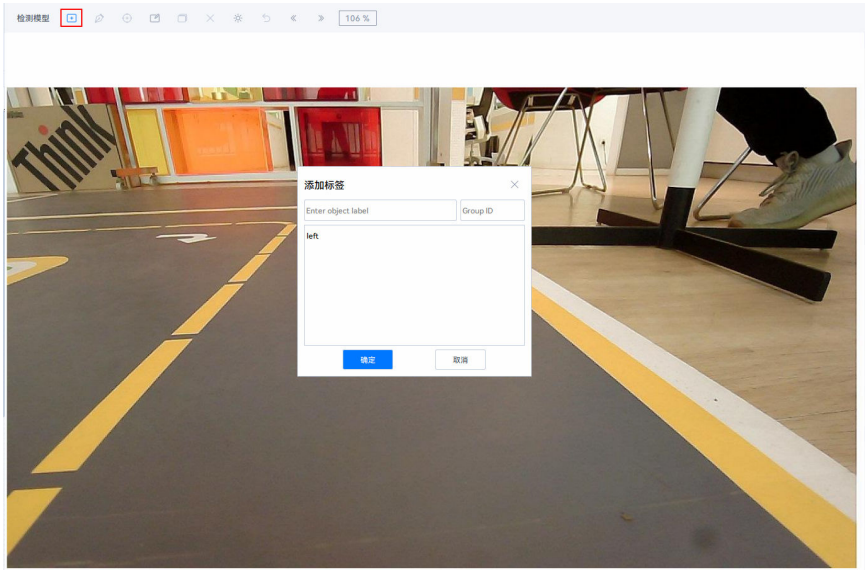
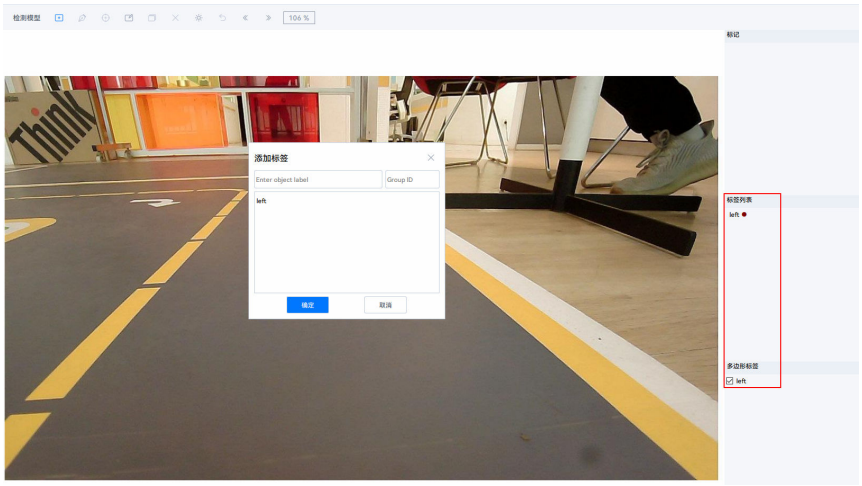
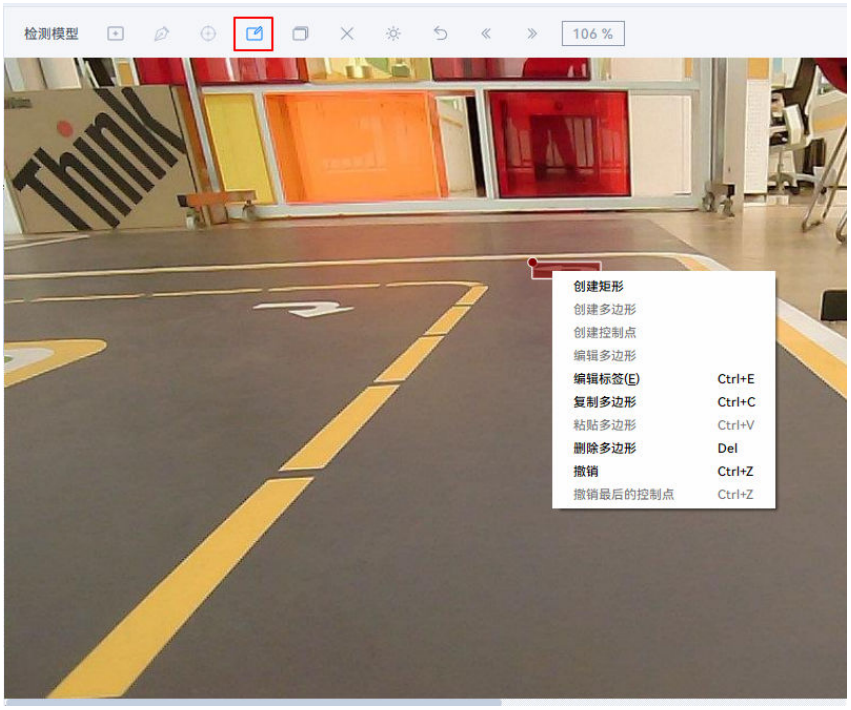


图 A-2 标注结果



3. 若标记错误可单击 按钮，按住左键可以移动标记框，移动鼠标至矩形框并单击“鼠标右键”，对矩形标签进行修改。

图 A-3 修改标签



4. 当前图片标注完成后，单击图片上方菜单栏中 图标或在左侧文件列表选择下一张图片进行标记，直到完成所有图片的标注任务。

说明

- 标注时输入标签仅支持数字、字母、下划线。
- 数据集图片要从实际模型部署使用的环境获得。
- 需将图片中的所有待检测目标都标注出来，漏标注将影响模型精度。
- 边框需要紧密框住每个目标，且类别正确，标注无误。

----结束

模型迁移

步骤1 在工具界面单击下方“一键迁移”按钮，进入配置界面，输入迁移信息，单击“一键迁移”开始迁移。

图 A-4 模型一键迁移配置界面

- 数据集路径：步骤2输出的自定义数据集输出路径。
- 数据集拆分：将图片划分成训练、验证以及测试集的比例，推荐值：0.3。默认拆分0.1的测试集用于边缘推理，训练集与验证集按输入拆分比例再次进行拆分。
- 迭代次数：训练轮次，推荐值：100。
- 每批图片数：参与每个批次训练的图片张数，推荐值：12。
- 预训练模型：可选yolov5s，yolov5n，yolov5l，yolov5x，默认yolov5s。
- 输出目录：模型输出路径。
- 使用早停策略：勾选后，可根据设置的mAP值（均值平均精度，一般指图片内所有类别的AP的平均值）和持续迭代不上升次数，提前停止训练。
  - mAP达到（值）：该训练模型精度已达标，可停止训练的阈值，默认值：0.99。
  - mAP连续迭代不上升次数：mAP值达到某一水平，多次迭代后并无提升的次数，默认值：10。

步骤2 迁移完成后会出现提示框，提示已生成打包好的文件，如图A-5所示。在训练输出目录会生成以下文件与目录，如图A-6所示。

- train_output：训练输出的权重文件、onnx文件以及训练数据信息json文件。
- trans_output：经过数据转换，根据数据集拆分设置生成的测试集、验证集、训练集。
- infer_project.tar.gz：打包好的推理相关模型文件与脚本。

图 A-5 迁移完成

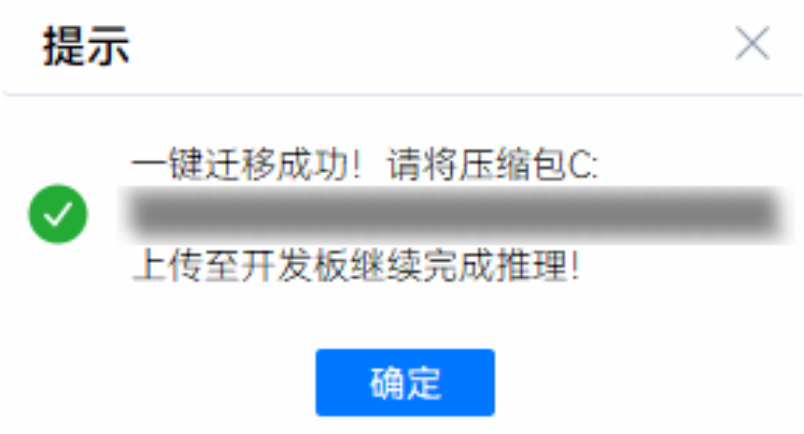
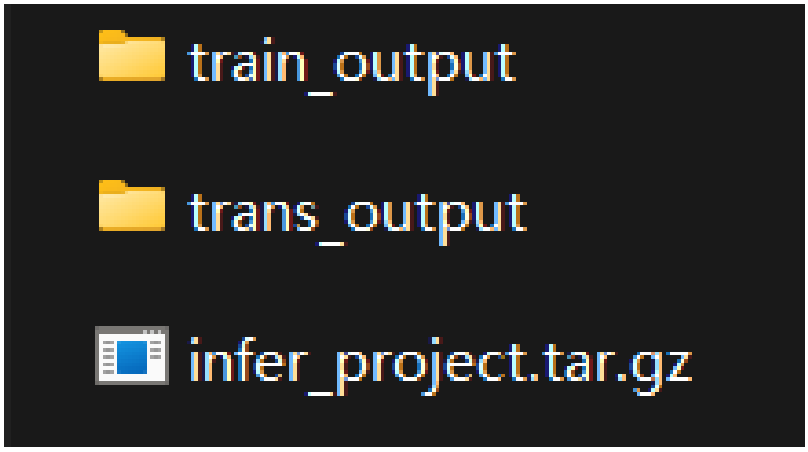


图 A-6 输出文件



----结束