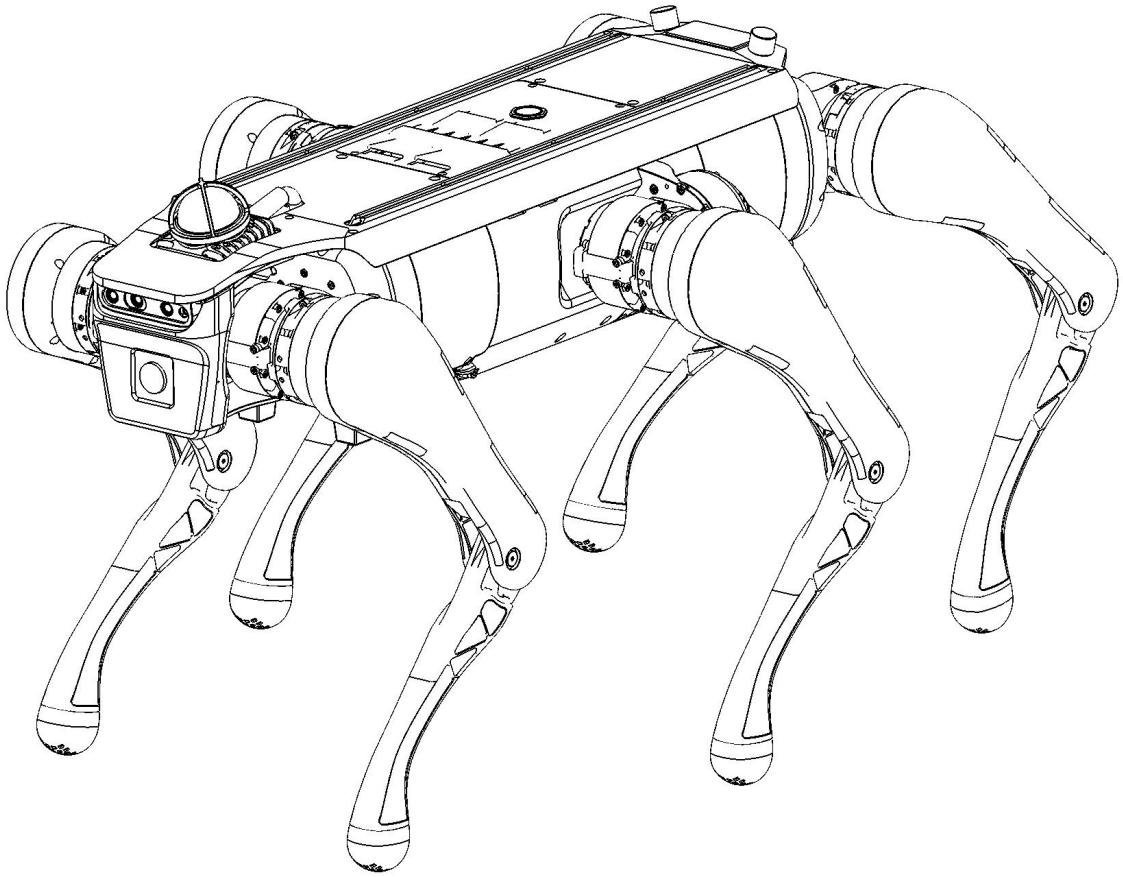
Hexplorer 探索者

用户手册 V1.6

2025/9/30



深圳市越疆科技股份有限公司

前言

目的

本手册用于指导用户使用和二次开发 Hexplorer 六足仿生机器人操作平台。

读者对象

本手册适用于：

- 客户
- 销售工程师
- 安装调测工程师
- 技术支持工程师

修订记录

版本号	修订描述	修订日期
V1.0	六足机器人产品手册创建	2025-01-16
V1.1	增加手柄按键说明&校对各个章节说明内容	2025-01-23
V1.2	错误文字以及代码部分校验	2025-02-06
V1.3	增加相机以及雷达坐标数据	2025-04-07
V1.4	系统软件升级更新/内容校对	2025-04-30
V1.5	更新二次开发部分（客户端应用开发梳理）	2025-04-30
V1.6	六足机器人产品手册发行版	2025-09-30

符号约定

在本手册中可能出现下列标志，其所代表含义如下所示：

符号	修订描述
危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害
警告	表示有中度或低度潜在危害，如果不能避免，可能导致人员轻微伤害、机器人毁坏等情况
注意	表示有潜在风险，如果忽视这些文本，可能导致机器人损坏、数据丢失或不可预知的结果
说明	表示是正文的附加信息，是对正文的强调和补充

责任声明

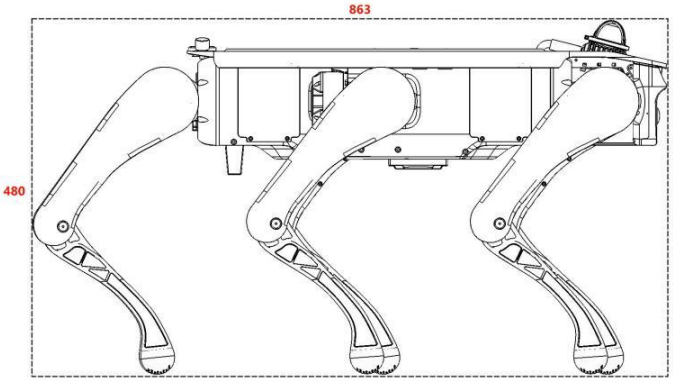
在法律允许的最大范围内，本软件及其相关文档按“原样”提供。深圳市越疆科技股份有限公司（以下简称“越疆科技”）特此声明：

- 不对本文档提供任何形式的明示或默示保证，包括但不限于适销性、质量满意度、适合特定目的、不侵犯第三方权利等保证。
- 不对因使用本文档导致的任何特殊、附带、偶然或间接的损害承担赔偿责任，包括但不限于商业利润损失、系统故障、数据或文档丢失产生的损失。
- 努力保证本手册提供信息的准确性，但不对可能出现的文字或图形疏漏负责，并保留更改本手册的权利，恕不另行通知。
- 本产品仅用于民用、商业及科研教育用途，严禁用于军事、武器开发或其他违反国际准则的场景，亦不得用于任何非法用途。使用时请遵守各地区适用的法律法规。
- 用户在使用本产品及相关文档时，应确保具备必要的专业知识和安全措施，若因操作不当、环境条件不符合要求或未遵循使用说明而产生风险或损失，由用户自行承担。
- 本产品可能包含或依赖第三方软件、硬件或服务。对于此类第三方内容的适用性、稳定性与兼容性，越疆科技不作任何保证，也不承担由此产生的任何责任。
- 越疆科技保留随时更新、修改或终止产品及相关文档的权利，且无须事先通知。用户应在使用前确认所获取的信息和版本为最新。

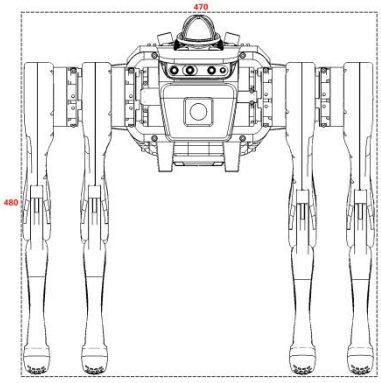
目录

产品概览

整机视图

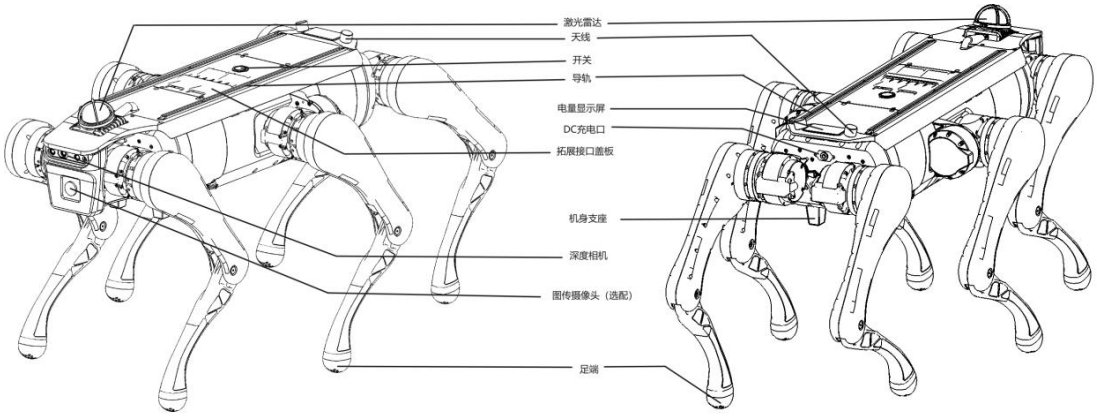


侧视图



正视图

部件说明



规格参数

参数名称	数值
型号	DT-HEX-6R010-L
长×宽×高（站立尺寸）	86.3cm*47cm*48cm
长×宽×高（折叠尺寸）	73.4cm*47cm*24.4cm
长×宽×高（包装尺寸）	81.5cm*54.3cm*35cm
自由度(DOF)	18
整机重量	本体约 20kg，含包装约 30Kg
最大行走速度	1.8m/s
最大越障高度	20cm
最大斜坡适应角度	40°
负载能力	额定 10Kg，极限 15kg
关节运动空间	前中髋关节：-60° ~ 180°
	后髋关节：-60° ~ 239°
	小腿：-30° ~ -160°
	机身：±38°
电池容量	504wh
续航时间	2.5 小时
充电时间	1.5 小时
工作环境	温度：0°C - 40°C 湿度：25%-85%，非冷凝
输入	100 - 240VAC，50 - 60HZ
输出	48V
额定功率	240W
满载电流	5A
电机峰值扭矩	33Nm（膝关节）
噪音等级	运动噪音≤55dB。
	*测试环境：室内薄地毯，分贝仪距离 1 米左右。

请用户在开箱后务必核实包装内各项物品的数量及完好状况。本手册所示图片仅供参考，实际产品以实物为准。不同型号的配置可能略有差异，随附配件请以所购型号的实物为准

功能示意图

架构信息

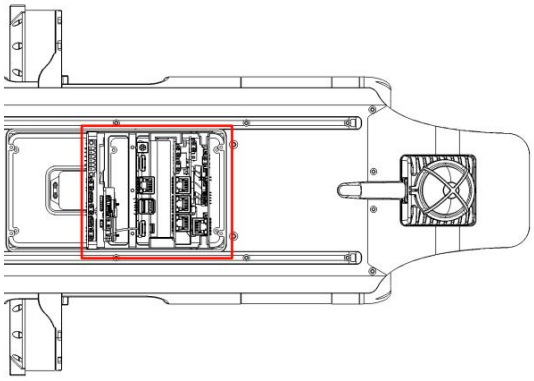
运控平台架构	Intel minipc x86_64 架构；
	CPU：酷睿 i7/i5，6 核心16 线程，标称主频 1.2GHz。
	RAM：16GB；ROM：95GB(Available)；
算力平台	Jetson Orin Nano Arm 架构
	CPU：6XA78，RAM：16GB；ROM：128GB；算力：40TOPS
无线手柄	雷神 G30s，蓝牙 5.0 连接，通信距离 10 米，频段 2.4G
扬声器	支持
激光雷达	Livox Mid360
深度相机（RGB-D）	Realsense D435

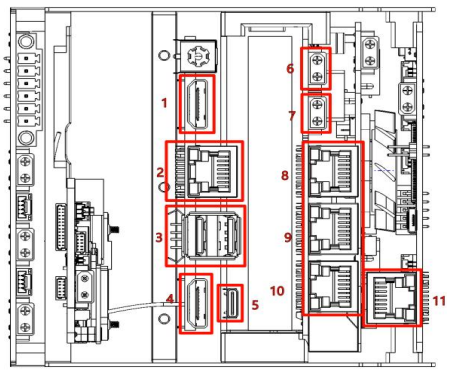
功能介绍

探索者 Hexplorer 六足仿生机器人以强大的性能与卓越的地形适应能力为核心，兼具科研教育与工业应用的双重价值。其能于多种环境中保持稳定而敏捷的姿态，具备适应复杂地形的运动能力，并配有完整的 SDK 与开发文档，为二次开发与功能扩展提供支持。Hexplorer 六足机器人在硬件层面已经集成 Livox Mid360 激光雷达、Realsense D435 深度相机以及高精度 IMU 等传感器，并在硬件设计上预留了充足的扩展接口，便于用户根据自身需求挂载传感器、扩展功能或进行定制化开发。基于此平台，用户可以实现更多高阶功能，包括但不限于：

- **复杂地形行走**
- **自主导航**
- **自主绕避障**
- **人体识别**
- **目标跟随**

探索者 Hexplorer 为开发者提供了完善的二次开发支持并在板载上预留有相当丰富的硬件接口，接口位置以及相关说明如下所示，





编号	接口名称
1	Nano-DP
2	Nano-TypeA
3	Nano-ETH
4	Nano-TypeC
1	MiniPC-HDMI
2	MiniPC-ETH
3	MiniPC-TypeA
4	MiniPC-HDMI
5	MiniPC-TypeC
6	Power-19V2A(output)
7	Power-19V2A(output)
8-10	Switch-RJ45(1000M)
11	CommunicationBoard-RJ45

Hexplorer 机身同时提供 Jetsons Nano 接口、Intel MiniPC 接口、电源输出与高速通信端口等，支持多种外设扩展。

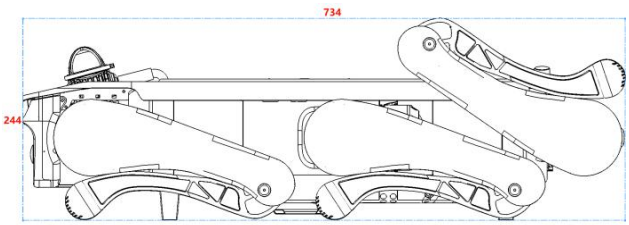
外部扩展	支持协作机械臂扩展（*后期支持）
	支持远距离遥控图传（*后期支持）
	支持无线自动充电（*后期支持）
电源&通信扩展	供电接口×1：19V/2A（连接外部 PC 用） 通信扩展：USB type-C×1、Ethernet×1

关于运输、搬运和储存

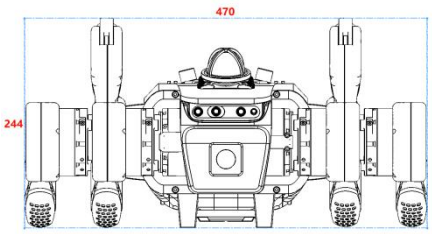
运输

机器人的专用运输箱整体重量约为 30 kg，其中机器人本体约 20 kg，其余为包装材料及附件。为确保运输安全，在搬运或装载前请务必确认运输箱的锁扣已完全扣紧。运输过程中应避免剧烈振动、翻转及冲击，以防止设备内部精密部件受到损伤。建议在长途运输时使用缓冲垫或固定装置，以进一步提高抗震与防护效果。

在打包过程中，请按照规定的收纳姿态将机器人放置于运输箱内：机器人六肢应折叠至收拢状态，机身与缓冲内衬紧密贴合，以避免运输过程中出现晃动或碰撞。下文所示的正面与侧面打包示意图可作为操作参考，实际打包请以运输箱内衬设计为准。



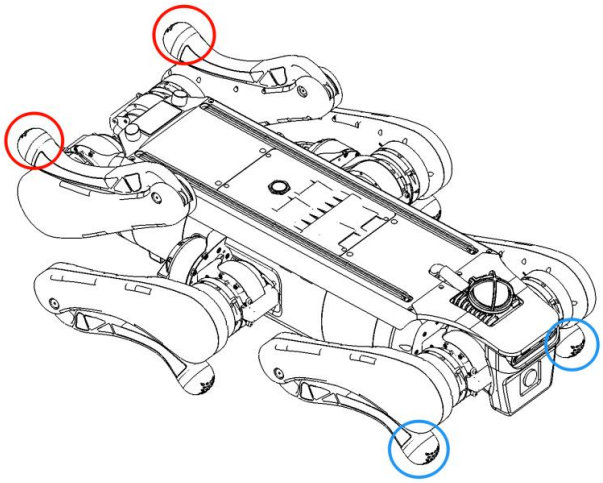
收纳姿态-侧面



收纳姿态-正面

搬运

仅当电源关闭， 或机器人电源打开且进入阻尼模式（各个关节类似阻尼器）时，才能搬运六足机器人，搬运需要两人配合操作。通过下图标注的把手位置（一个人握住红色两小腿，一个人握住蓝色两小腿）进行搬运，搬运过程中注意避免夹伤或缠绕衣物等。



抓握与防夹位置

储存

⚠ 必须关闭机器人电源。存储温度-20℃~40℃，相对湿度 30%~70%。严禁将水或其他液体淋到机器人上。严禁将其他各类物品放置在关节旋转范围内。建议划定一个安全、整洁而且专门用于存储机器人的区域。建议使用专门为六足机器人设计的运输箱来储存，若长期储存，请每三个月为机器人充电一次以保持电池活性。

如何充电

机器人本体

使用 58.8V 锂电池充电，将充电器插进蓝色充电口（如下图所示）。充电器红灯亮时代表正在充电，绿灯代表已充满。同时机器狗的 LCD 显示屏也会显示电池剩余电量。DC 充电口的位置请参考“产品概览-部件说明”。

DC 充电口

充电中的机器人

控制器

使用 5V-2A Type-C 充电口充电，红灯闪烁时代表正在充电，灯灭代表已充满。

外部供电

将充电器插进蓝色充电口（如上图所示），然后直接按下开机按钮开机即可。注意，如果需要控制机器人运动控制，充电口需考虑做紧固处理，从而防止充电线脱落。

如何使用您的机器人

使用前准备

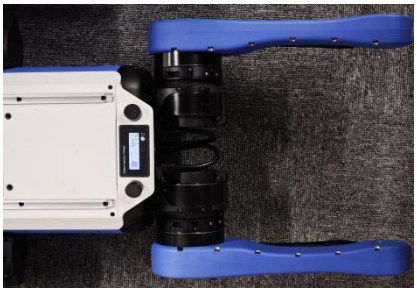
请注意以下事项，以确保安全和有效的操作

- 避免无必要目的进入或停留在机器人的活动区域内。
- 操作机器人时，请令其保持视线范围内。
- 避免接近运动中的机器人，尤其是其活动关节或其他组件。
- 不要试图干预机器人的运动状态，尤其是在机器人出现不稳定情况时。
- 禁止使用机器人进行攀爬。
- 其他环境相关注意事项：
- 机器人运行过程中应与人群、水面、明火等保持至少 2 米以上的安全距离。
- 避免在 WiFi 信号干扰严重的环境下操作机器人，并在必要时关闭其他无线设备的 WiFi 信号源。
- 禁止接触危险物质。
- 禁止直接/间接接触带电部件
- 禁止在潜在的易爆环境中使用机器人。

在强电磁干扰环境中操控机器人前，务必先咨询售后。

开机前做如下检查

1) **机器人是否完整**：机器人外形是否完整，机械限位是否损坏。
2) **电池是否电量充足**：按下机器人上部圆形开关按钮，观察机器人侧面 LCD 屏幕电量百分比。
3) **手柄是否电量充足**（户外使用尤其注意）：点击手柄开关键，绿灯闪烁代表电量正常，红灯闪烁代表电量低，需要充电。



机器人电量充足

手柄电量充足

确认无误后将机器人按照下图所示放于水平地面，且大小腿处于图中收拢状态。

收拢六肢

开机

1) 完成现场检查和准备工作后，按下机器人上盖圆形按钮，灯亮起代表机器人上电。
2) 打开开关 2s 后，听到电机沙沙的声音，代表电机和主控都已上电。

3) 约 40s 后，打开手柄开关，当手柄发出震动并开关键常亮时表示手柄已成功连接。

状态机切换

机器人开机并成功连接手柄后，自动进入阻尼模式。机器人包含以下状态。

状态	说明
阻尼模式	各电机带小阻尼，此时摆动电机可以感受到较小的阻力。机器人出现异常状况时使用该模式。
位置折叠模式	六肢摆动到起始固定位置，此时各电机带有较大阻尼。
力控站立模式	机器人站立，此时摇杆可控制其位姿。左摇杆横轴控制偏航角、纵轴控制基座高度，右摇杆横轴控制横滚角、纵轴控制俯仰角。该模式可用于机器人关节工作状态自检。
力控行走模式	左摇杆横轴控制偏航朝向、纵轴控制前后行进；右摇杆横轴控制左右行进、纵轴无控制量。
复杂地形行走模式	机器人重心略微下移，该状态下机器人具备更强的地形适应能力，包括但不限于楼梯、平地、草地、湿地、碎石地等。
拳击模式	机器人中后肢支撑着地，前肢快速向前挥击，该状态为演示状态，演示完后自动回到力控站立模式。
导航模式	---

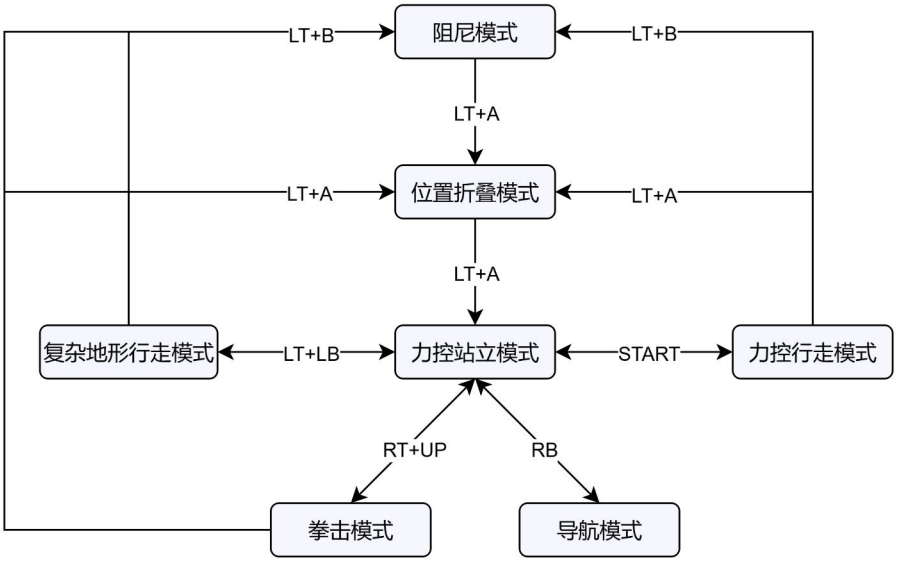
位置折叠模式

力控站立模式

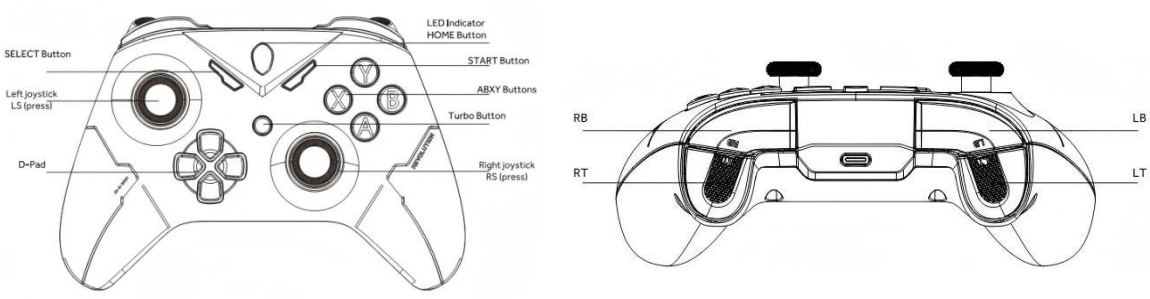
力控行走模式

复杂地形行走模式

不同状态之间的转换关系如下图。



手柄控制器的按键对应图如下。



手柄重要按键说明：

按键	状态	说明
状态键(Home Button)	蓝色常亮	正常连接
	红色闪烁	电量低
	蓝色闪烁	正在连接
	短按一次	触发手柄连接，连接成功时产生振动
	长按	手柄关机
左摇杆(Left Joystick)	力控站立模式下	左右控制偏航姿态
		上下控制机器人起立和下蹲高度
	力控行走模式下	左右控制偏航角速度（旋转行走）
		上下控制前后行走速度
右摇杆(Right Joystick)	力控站立模式下	左右控制横滚姿态
		上下控制俯仰姿态
	力控行走模式下	左右控制左右平移速度
		上下无控制量

关机

控制机器人收拢到**位置折叠或阻尼模式**，关闭机器人电源键，手柄电源会自动关闭无需操作。

应急操作

⚠ 如机器人出现不稳定状态，及时进入**阻尼模式**（LT+B）或直接关闭系统电源，如果从阻尼模式进入位置模式后关节位置出现异常，关机摆放好位置后重新启动。

二次开发与 SDK

OS 与开发环境

Hexplorer 探索者运动控制平台使用 Intel Minipc，i7/i5 酷睿 CPU，标称主频 1.2GHz，具备节能、性能两种工作模式，6 核心 16 线程，RAM 16GB，可用 ROM 95GB；算力平台采用 Jetson Orin Nano，Arm 6XA789 CPU，RAM 8GB，ROM 128GB，GPU 算力 40TOPS。

操作系统为Ubuntu 22.04，预装ROS2 humble版本系统。系统中预装C/C++、Python(3.10)、miniconda、docker、程序编译、虚拟开发调试环境。

若需要使用 conda 环境，需要用户手动开启，这里提供两种方式。

1) 所有终端永久生效

所有终端永久生效，将 ~/.bashrc 文件中 125~135（含 125 和 135）行去掉注释，而后执行命令

```
source ~/.bashrc
```

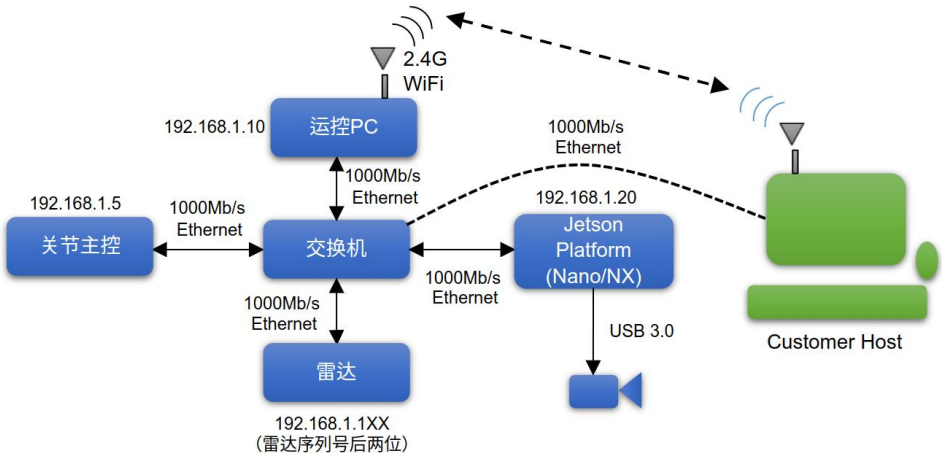
2) 单个终端临时生效

如果只需要在当前终端使用，直接运行

```
source /etc/conda/setup.bash
```

网络与访问

内部网络拓扑结构



网络连接

默认情况下机器人的 wifi 功能为关闭状态，如需使用 WIFI，可以外接键鼠和显示器直接在板载桌面进行修改，或者通过脚本 start_wifi.sh 实现，在命令行中执行：

```
./start_wifi.sh <WiFiName> <Password>
```

板载访问

由网络拓扑结构可知，机器人无线网卡与有线网卡的 IP 分别为：192.168.12.1、192.168.1.10(minipc)和 192.168.1.20(Jetson Orin Nano)。除此之外，机器人已预装 ssh/telnet 且开机自启。

对于用户来说，有三种方法可以连接到板载：

1) 网线直连

Minipc 和 Jetson Orin Nano 二者选其一即可

```
ssh robot@192.168.1.10 # 用户名 robot,密码为 123
ssh robot@192.168.1.20 # 用户名 robot,密码为 123
```

2) WIFI 连接（推荐）

Wifi 名为 YJ-MiniHexV2-xxx，密码 1234abcd。访问方法为

```
ssh robot@192.168.12.1 # 用户名 robot,密码为 123
```

3) 直接连接

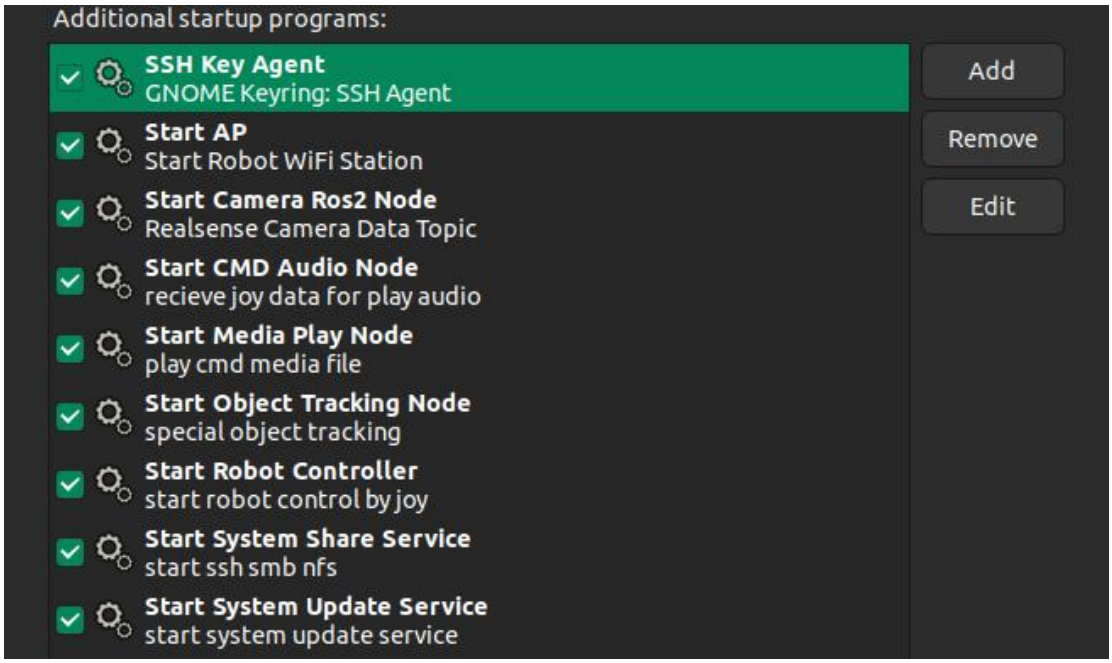
直接将外部显示器与键鼠通过板载硬件接口与板卡连接，可以直接访问到 Ubuntu 桌面。此法不适用于远程控制。硬件接口图请参考“功能示意图-外部扩展支持”。

文件访问

机器人已预装 nfs/samba，和 ssh/telnet 服务。有需要的用户可以使用文件共享服务。

自启动服务管理

系统自动启动服务使用 StartupApplication 进行管理，进入具体界面如下图：



界面中的每一项都属于系统开机时自动开启的服务，系统默认开启的有 SSH、Samba、NFS 服务，系统升级服务等，自动启动 WiFi 热点、开启 Realsense D435 相机节点(标配相机设备会自动开启，若无请忽略），开启雷达节点，机器人基础运动控制节点，以及定制化功能配套节点。如果不需要这些节点开启，可以将其失能（系统相关的服务不能失能，否则会引起设备升级失败）。

ROS2 与节点拓扑

ROS2 环境配置

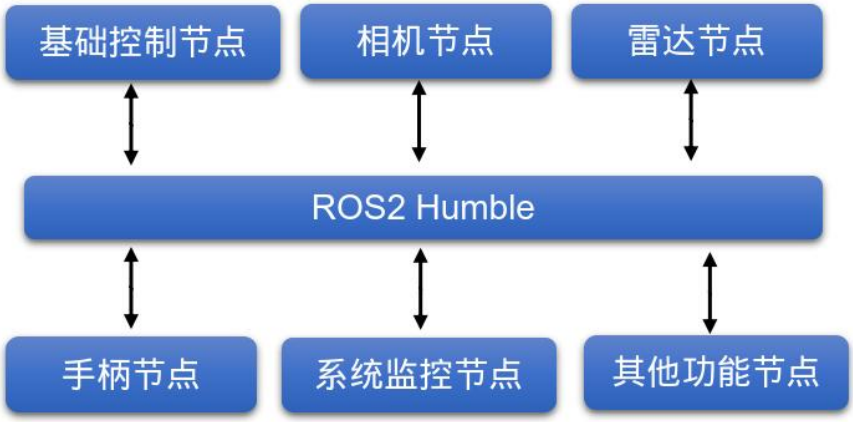
机器人板载环境中默认环境已经配置好 ROS2，ROS2 版本的安装强依赖 Ubuntu 系统版本，确保

在正确的系统版本下安装对应的 ROS2 系统版本。如果用户需要再次配置的话请参考该脚本中的步骤：

```
./ros2_env.bash
```

节点拓扑结构

机器人软件系统，采用 ROS2 作为中间件，机器人系统运行节点基于 ROS2 开发，如下图



软件架构说明图

机器人系统网络链路中的 PC 均可以查看订阅机器人系统发布的活动节点，命令为

```
ros2 topic list # 检查所有话题
```

a. 在用户主机端，如果通过网线直连，可以访问机器人系统中发布和订阅的所有节点。

b. 在用户主机端，如果通过 Wifi 连接，可以访问运控板载发布和订阅的节点。

说明：机器人系统运行的节点，根据不同配置，预装或者不预装，详情请咨询技术支持。

控制系统与 SDK

机器人系统的所有运动程序相关内容位于运动控制 PC 系统用户目录下的 robot_controller_release 文件夹。该文件夹中包含 config，third_party 以及 executable 文件夹。config 为机器人的描述文件，定义机器人的类型，包括所有关节的标定位置、转向正负、id 等信息，非必要无需用户手动修改。executable 文件夹中包含运控主程序的可执行文件 main，关节测试程序执行文件 motor_test_example 和系统标定辅助程序执行文件 motor_test_calibration。third_party 中包含第三方的软件资源。

控制系统的更新目前需要在本地由用户完成，通过整体更新覆盖 robot_controller_release 实现，每次更新后，需要将三方软件资源重新安装更新，同时给程序添加可执行权限，具体操作如下，

```
cd ~/robot_controller_release
./setup.sh
```

重启机器人，按照正常启动流程后机器人正常启动，代表控制软件更新完成。

高层控制 SDK

消息类型

机器人内部数据通讯软件架构，使用 ROS2 作为中间件，通讯方式为发布或订阅 ROS2 话题消息。机器人所有上层消息类型定义在名为 custom_msg 的 package 中，机器人系统默认安装了 custom_msg

包，具体位置为：robot_controller_release/ros2_packages/custom_msg/share/custom_msg。其目录结构如下

```
drwxrwxr-x 2 robot robot 4096 Sep 18 03:10 ./
drwxrwxr-x 6 robot robot 4096 Sep 18 03:10 ../
-rw-rw-r-- 1 robot robot   88 Sep 18 03:10 CustomPoint.msg
-rw-rw-r-- 1 robot robot   49 Sep 18 03:10 DetectCommand.msg
-rw-rw-r-- 1 robot robot  108 Sep 18 03:10 LivoxPointcloud.msg
-rw-rw-r-- 1 robot robot   55 Sep 18 03:10 MediaPlayer.msg
-rw-rw-r-- 1 robot robot  154 Sep 18 03:10 MotorState.msg
-rw-rw-r-- 1 robot robot   52 Sep 18 03:10 ObjectPosition.msg
-rw-rw-r-- 1 robot robot  109 Sep 18 03:10 ObstacleDetection.msg
-rw-rw-r-- 1 robot robot  118 Sep 18 03:10 RobotCommand.msg
-rw-rw-r-- 1 robot robot  317 Sep 18 03:10 RobotState.msg
-rw-rw-r-- 1 robot robot  171 Sep 18 03:10 TagsDetection.msg
-rw-rw-r-- 1 robot robot  213 Sep 18 03:10 Template.msg
```

注：该开发依赖包在不同机器人机型版本中存在变化，详情请咨询技术支持人员。

下面对部分关键消息数据结构做出说明

1） RobotState.msg

数据名	说明	数据名	说明
header	时间戳，坐标系等	jpower_total	关节总功率
control_cmd	控制命令	pos_body	身体实时位置，排列顺序(x,y,z)，单位 m
jtau_leg	腿部关节力矩	vel_body	身体实时速度，排列顺序(x,y,z)，单位 m/s
jvel_leg	腿部关节速度	acc_body	身体实时加速度，排列顺序(x,y,z)，单位 m/s2
jpos_leg	腿部关节位置	ori_body	身体实时姿态，单位四元数格式，格式(w，x，y，z)
jtau_leg_des	腿部关节目标力矩	omega_body	世界坐标系下身体角速度，排列顺序(x,y,z)，单位 rad/s
jvel_leg_des	腿部关节目标速度	pos_foot_left	双足用，左侧足端位置
jpos_leg_des	腿部关节目标位置	pos_foot_right	双足用,右侧足端位置
jpower_leg	腿部关节功率	vel_foot_left	双足用,左侧足端速度
jpos_cmd_leg	腿部关节位置命令	vel_foot_right	双足用,右侧足端速度
jtau_arm	手臂关节力矩	grf_left	双足用，左侧足端反馈
jvel_arm	手臂关节速度	grf_right	双足用，右侧足端反馈力
jpos_arm	手臂关节位置	rf_vertical_filtered	双足用，FZ
jtau_total	关节总力矩	temp	预留，temp[10]用于反馈机器人运行状态(0.passive 1.standDown 2. standUp 3.balanceStand 4. Walk)

2） RobotCommand.msg

数据名	说明

数据名	说明
header	时间戳,坐标系等
target_state	机器人运行切换模式 0.passive 1. standDown 2. standUp 3. balanceStand 4. Walk
temp	预留

3) geometry_msgs/msg/Twist

数据名	说明
linear.x	机器人 X 方向的线速度，单位 m/s
linear.y	机器人 Y 方向的线速度，单位 m/s
angular.z	机器人偏航角速度， 单位 rad/s

终端命令控制

1) 匀速行走

在机器人行走状态下，行走的速度指令是手柄和用户上层指令的叠加，用户上层指令通 geometry_msgs/Twist 消息进行发送，其中 linear.x 为机器人前进方向线速度指令(m/s)，linear.y 为机器人侧方向线速度指令（m/s），angular.z 为机器人的转动速度指令（rad/s）例如，输入下述指令到命令行，机器人将以 0.1m/s 速度向前行走。

```
source ~/robot_controller_release/ros2_packages/setup.bash
ros2 topic pub -r 10 /vel_cmd geometry_msgs/Twist '{linear: {x: 0.1, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

2) 状态切换

使用下述命令可以控制机器人的状态切换

```
ros2 topic pub -1 /robot_cmd custom_msg/msg/RobotCommand "{target_state: 0}"
```

C++高层控制

用户下载示例代码 dobot_hex_sdk，示例程序的目录为 dobot_hex_sdk/high_level/src/，该仓库包含 3 个示例。下文会对部分示例进行说明。在此之前，为满足用户自定义程序的需求，先介绍控制框架的搭建。

1) 包的创建、编译与执行

用户可以直接使用 CMake 不依赖于 ROS2 来直接构建包。这意味着用户不需要用到 ament 和 colcon 等工具，这种方式最为简单且直接，也是案例程序所使用的构建方式（**推荐**）。如果用户确实需要使用 colcon 和 ament_cmake 作为构建工具，用户需要先创建一个 ROS2 package，在工作空间的源目录下执行命令：

```
ros2 pkg create --build-type ament_cmake <package_name>
```

可以得到一个名为<package_name>的包目录，整体工作空间的目录结构可能如下：

用户可以在自己创建的包下开发控制程序，一旦程序编写完成，包括源文件与 CMakeLists.txt 文件等，在工作空间的根目录 workspace_folder/下执行

```
rosdep install -i --from-path src --rosdistro humble -y
```

该命令会检查 package 的所有依赖项是否完备，注意在使用 rosdep 和编译命令之前都需要确保已经加载了 ros2 的环境变量，在示例程序下，需要执行以下命令

```
source /opt/ros/humble/setup.bash
source ~/robot_controller_release/ros2_packages/setup.bash
```

依赖项检查完毕后，依然在根目录下执行编译命令：

```
colcon build --packages-select <package_name>
```

这里的<package_name>为用户创建的包名，该命令会在 src 目录下创建 build、install、log 文件夹。Pacakge 及其可执行文件都被安装到了 install 目录下，为了能在工作空间中调用 package 及其可执行文件，在终端中 src 目录下执行如下命令

```
source install/setup.bash
```

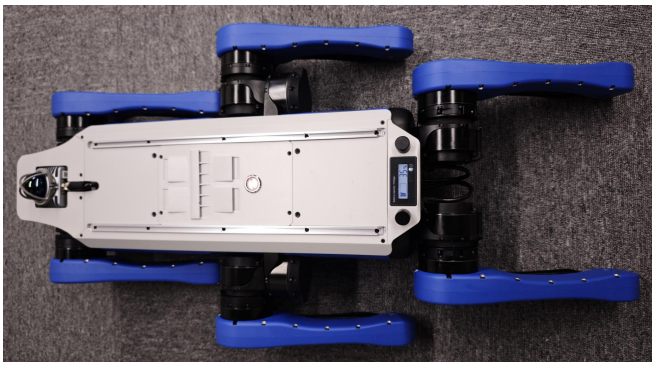
而后可运行该节点

```
ros2 run <package_name> <executable file>
```

这里可执行文件的名称由用户在 CMakeLists.txt 中自行定义。

2) 示例一：body_twist

文件位于 dobot_hex_sdk/high_level/src/body_twist.cpp，本实例旨在检查各关节电机的情况。将机器人平放于水平地面，六腿折叠放置。如下图：



六肢折叠放置于水平地面

正常情况下运行本实例，用户应看到六足机器人先呈站立姿态，而后各关节电机不断按照一定规律周期运动，包括横滚、俯仰、偏航方向的姿态运动，以及底座高度方向的运动。

注意：停止程序时请先停止 example 的程序，再使用手柄将机器人切换到其他模式。

本实例涉及 robot_cmd、robot_state、joy 话题的发布与订阅，用户可参考本代码进行相关的自定义高层控制开发。接下来对核心代码部分进行简要说明：

程序本体作为一个 ROS2 节点被执行，通过话题对现有节点进行发布与订阅皆需在节点内部实现。因此，功能类需继承自 rclcpp::Node 并创建定时器、发布者和订阅者等。下述程序为简化版，请按照实际程序为准。

```
class controlFuncNode : public rclcpp::Node
{
public:
    controlFuncNode(const std::string &strNodeName) : Node(strNodeName)
    {
        using namespace std::literals::chrono_literals;
        using std::placeholders::_1;
```

```
08
09          m_pControlTopicTimer = this->create_wall_timer(20ms, std::bind(&control
    FuncNode::controlMsgFunc, this));
10          m_pRobotcommandPublisher = this->create_publisher<custom_msg::msg::Rob
    otcommand>(m_strRobotcommandTopicName, 1);
11          m_pRobotstatesubscriber = this->create_subscription<custom_msg::msg::R
    obotstate>(m_strRobotstateTopicName, 1, std::bind(&controlFuncNode::parseRobot
    StateMsgFunc, this, _1));
12      }
13      void controlMsgFunc(void);
14      void parseRobotStateMsgFunc(const custom_msg::msg::RobotState::SharedPtr m
    sg);
15
16  private:
17      rclcpp::TimerBase::SharedPtr m_pControlTopicTimer = nullptr;
18      rclcpp::Publisher<custom_msg::msg::RobotCommand>::SharedPtr m_pRobotcomman
    dPublisher = nullptr;
19      rclcpp::Subscription<custom_msg::msg::RobotState>::SharedPtr m_pRobotState
    Subscriber = nullptr
20  };
```

ROS2 会按照 20ms 的周期调用函数 controlMsgFunc，在构建好发布的信息后 publisher 会将之发布，订阅者收到信息后会传给绑定的函数 parseRobotStateMsgFunc 进一步处理。发布与订阅的参考示例如下：

```
01  void controlFuncNode::controlMsgFunc(void)
02  {
03      custom_msg::msg::RobotCommand cmdMsg;
04      cmdMsg.target_state = m_msgRobotState.temp[10] + 1;
05      m_pRobotCommandPublisher->publish(cmdMsg);
06  }
07
08  void controlFuncNode::parseRobotStateMsgFunc(const custom_msg::msg::RobotStat
    e::SharedPtr msg)
09  {
10      m_msgRobotState.temp[10] = msg->temp[10]; // Robot State Get Byte
11      RCLCPP_INFo(this->get_logger(), "subscribe Robot State Msg, m_msgRobotstat
    e.temp[10] = %f", m_msgRobotstate.temp[10]);
12  }
```

本实例使用了 joy 话题的信息，ROS2 内置有该消息类型，包名为 sensor_msgs。若用户需要模拟手柄的输入，可参考以下代码：

```
01  const float f_roll = 0.012;
02  const float f_pitch = 0.012;
03  const float f_yaw = 0.012; // Hz: yaw 频率（LX）
04  const float f_z = 0.012;
```

```
05
06  sensor_msgs::msg::Joy joyMsg;
07  joyMsg.header.stamp = this->now();
08  joyMsg.axes.assign(8, 0.0f);
09  joyMsg.buttons.assign(11, 0);
10  joyMsg.axes[2] = 1.0f;
11  joyMsg.axes[5] = 1.0f;
12
13  joyMsg.axes[3] = std::clamp(static_cast<float>(0.7 * cos(2 * M_PI * f_roll * c
    ount)), -1.0f, 1.0f);  // roll
14  joyMsg.axes[4] = std::clamp(static_cast<float>(0.7 * cos(2 * M_PI * f_z * coun
    t)), -1.0f, 1.0f);     // pitch
15  joyMsg.axes[0] = std::clamp(static_cast<float>(0.7 * sin(2 * M_PI * f_yaw * co
    unt)), -1.0f, 1.0f);   // yaw
16  joyMsg.axes[1] = std::clamp(static_cast<float>(0.7 * sin(2 * M_PI * f_pitch *
    count)), -1.0f, 1.0f); // z
17
18  m_pJoyTopicPublisher->publish(joyMsg);
```

本手柄包含 8 个 axes 与 11 个 buttons。Axes 对应情况如下

ID	按键	说明	ID	按键	说明
0	左摇杆横轴	从左到右为[1, -1]	4	右摇杆横轴	从上到下为[1, -1]
1	左摇杆纵轴	从上到下为[1, -1]	5	RT	按压到松开[-1, 1]
2	LT	按压到松开[-1, 1]	6	方向键横轴	左 1 右 -1 不按为 0
3	右摇杆横轴	从左到右为[1, -1]	7	方向键纵轴	上 1 下 -1 不按为 0

Buttons: 所有按键按下时为 1，松开时为 0

ID	按键	ID	按键
0	A	6	SELECT
1	B	7	START
2	X	8	HOME
3	Y	9	LEFT STICK PRESS
4	LB	10	RIGHT STICK PRESS
5	RB		

为了正确找到自定义和内置的消息类型，用户需要在 CMakeLists.txt 中进行指定，例如，在上述例子中我们使用了 custom_msg 类，则在 CMakeLists.txt 中：

```
1  find_package(custom_msg REQUIRED)
2  include_directories(
3      ${custom_msg_INCLUDE_DIRS}
4  )
5  tartt_link_libraries(<your_executable_name>
6      ${custom_msg_LIBRARIES}
7  )
```

3) 示例二：locomotion

文件位于 dobot_hex_sdk/high_level/src/locomotion.cpp，本实例旨在检查六足机器人行走的情况。仍然将机器人平放于水平地面，六腿折叠放置，如上所述。

正常情况下运行本实例，用户应看到机器人从折叠放置到站立，而后依次向前、向后、向左侧行、向右侧行、向左转向、想右转向，而后自动趴下。

注意：停止程序时请先停止 example 的程序，再使用手柄将机器人切换到其他模式。

此示例相较于示例一多涉及一个话题 geometry_msgs，该类包含三维空间下的速度，如示例一所示，用户需要创建该命令的发布者，例如

```
m_pRobotControllVelCmdPublisher = this->create_publisher<geometry_msgs::msg::Twist>(m_strVelCmdTopicName, 1);
```

用户可自行定义机器人在三个方向上的速度，三个方向为 x、y、yaw(angular z axis)。参考代码如下

```
auto controlMsg = geometry_msgs::msg::Twist();
controlMsg.linear.x = fXSpeed;
controlMsg.linear.y = fYSpeed;
controlMsg.angular.z = fYawSpeed;

controlMsg.linear.x = m_msgRobotControl.linear.x * 0.5 + fXSpeed * 0.5;
controlMsg.linear.y = m_msgRobotControl.linear.y * 0.6 + fYSpeed * 0.4;
controlMsg.angular.z = m_msgRobotControl.angular.z * 0.65 + fYawSpeed * 0.35;

m_pRobotControllVelCmdPublisher->publish(controlMsg);
```

4) 示例三：robot_state

本实例旨在读取机器人相关的数据信息，以便于后续调试与二次开发，robot_state 包含多个变量，此处以 pos_body、vel_body、acc_body、ori_body、omega_body 为例。正常情况下，运行该示例后六足机器人会停留在力控站立模式，并可在终端观察到这些值的数据，如下所示

```
robot@minipc-2204:~/robot_controller_examples/high_level_cpp/robot_state/build$ ./robotStateSubscriber
[INFO] [1758532817.351805819] [robot_state_subscriber]: Main robot states as below
control with gamepad and observe the states
+--------------+-------------------------------------+
| Name         | Value                               |
+--------------+-------------------------------------+
| pos_body     | [-0.014, -0.018, 0.079]             |
| vel_body     | [0.000, -0.000, 0.001]              |
| acc_body     | [-0.015, 0.039, 0.038]              |
| ori_body     | [1.000, -0.019, 0.013, -0.000]      |
| omega_body   | [0.000, 0.000, 0.000]               |
+--------------+-------------------------------------+
```

运行该示例后，用户可通过手柄在力控站立模式下测试各关节运行情况。并可在控制台观察到上述数据的变化。例如，用户可以通过左摇杆纵轴使得机器人底座下降，此时可在终端观察到 pos_body 有明显变化。

注意：停止程序时请先停止 example 的程序，再使用手柄将机器人切换到其他模式。

Python 高层控制

用户下载示例代码 dobot_hex_sdk，示例程序的目录为 dobot_hex_sdk/high_level/py/，该

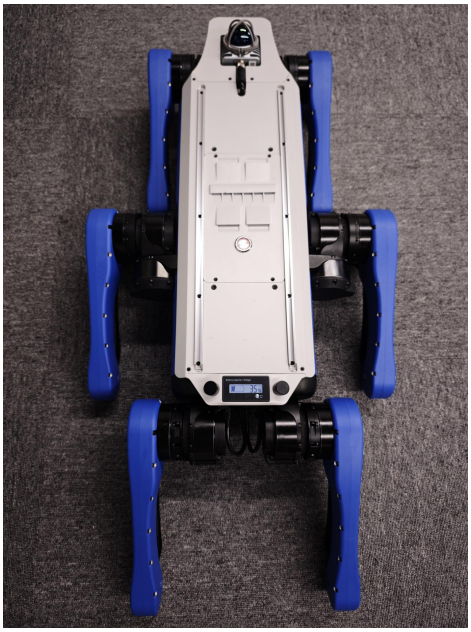
仓库包含 3 个示例。如使用 ros package 的模式构建程序，在编写程序之前，请参考上文“C++高层控制-包的创建、编译与执行”。

要想执行 Python 文件，需进入 Intel Minipc 中，使用 ROS2 自带的 Python 3.10 解释器而非 conda 的 Python 环境。因此请用户确保 ROS2 环境已被激活

```
source /opt/ros/humble/setup.bash
```

1) 示例一：body_twist

Python 版本所实现的功能与 C++版本完全一样，只是一些对于 ROS2 函数的调用上略有不同。文件位于 dobot_hex_sdk/high_level/py/body_twist.py，本实例旨在检查各关节电机的情况。将机器人平放于水平地面，六腿折叠放置。如下图：



六肢折叠放置于水平地面

本实例涉及 robot_cmd、robot_state、joy 话题的发布与订阅，用户可参考本代码进行相关的自定义高层控制开发。python 环境下计时器、发布者、订阅者的创建可参考如下代码：

```
self.robot_command_topic_name = "/robot_cmd"
self.robot_state_topic_name = "/robot_state"
self.joy_handle_topic_name = "/joy"

self.publisher_timer = self.create_timer(0.02, self.body_twist_control_msg)

self.robot_command_publisher = self.create_publisher(
    RobotCommand, self.robot_command_topic_name, 1
)
self.joyTopicPublisher = self.create_publisher(
    Joy, self.joy_handle_topic_name, 1
)
self.robot_state_subscriber = self.create_subscription(
    RobotState, self.robot_state_topic_name, self.robot_state_parse, 1
)
```

本实例使用了 joy 话题的信息，ROS2 内置有该消息类型，包名为 sensor_msgs。若用户需要模拟手柄的输入，可参考以下代码：

```
f_roll = 0.012
f_pitch = 0.012
f_yaw = 0.012  # Hz: yaw 频率（Lx）
f_z = 0.012

joyMsg = Joy()
joyMsg.header.stamp = self.get_clock().now().to_msg()

joyMsg.axes = [0.0] * 8
joyMsg.buttons = [0] * 11

# 固定触发器
joyMsg.axes[2] = 1.0
joyMsg.axes[5] = 1.0

roll_val = 0.7 * math.cos(2 * math.pi * f_roll * self.count)  # roll
pitch_val = 0.7 * math.cos(2 * math.pi * f_z * self.count)  # pitch
yaw_val = 0.7 * math.sin(2 * math.pi * f_yaw * self.count)  # yaw
z_val = 0.7 * math.sin(2 * math.pi * f_pitch * self.count)  # z

clamp = lambda x: max(-1.0, min(1.0, float(x)))
joyMsg.axes[3] = clamp(roll_val)
joyMsg.axes[4] = clamp(pitch_val)
joyMsg.axes[0] = clamp(yaw_val)
joyMsg.axes[1] = clamp(z_val)

self.joyTopicPublisher.publish(joyMsg)
```

本手柄包含 8 个 axes 与 11 个 buttons。按键对应情况如下

Axes:

ID	按键	说明	ID	按键	说明
0	左摇杆横轴	从左到右为[1，-1]	4	右摇杆横轴	从上到下为[1，-1]
1	左摇杆纵轴	从上到下为[1，-1]	5	RT	按压到松开[-1，1]
2	LT	按压到松开[-1，1]	6	方向键横轴	左 1 右 -1 不按为 0
3	右摇杆横轴	从左到右为[1，-1]	7	方向键纵轴	上 1 下 -1 不按为 0

Buttons：所有按键按下时为 1，松开时为 0

ID	按键	ID	按键
0	A	6	SELECT
1	B	7	START
2	X	8	HOME

ID	按键	ID	按键
3	Y	9	LEFT STICK PRESS
4	LB	10	RIGHT STICK PRESS
5	RB		

程序本体作为一个 ROS2 节点被执行，通过话题对现有节点进行发布与订阅。

编写完程序后在控制台执行：

```
python3 <executable file>
```

正常情况下运行本实例，用户应看到六足机器人先呈站立姿态，而后各关节电机不断按照一定规律周期运动，包括横滚、俯仰、偏航方向的姿态运动，以及底座高度方向的运动。

注意：停止程序时请先停止 example 的程序，再使用手柄将机器人切换到其他模式。

2) 示例二：locomotion

文件位于 dobot_hex_sdk/high_level/py/locomotion.py，本实例旨在检查六足机器人行走的情况。仍然将机器人平放于水平地面，六腿折叠放置，如上所述。

正常情况下运行本实例，用户应看到机器人从折叠放置到站立，而后依次向前、向后、向左侧行、向右侧行、向左转向、想右转向。

注意：停止程序时请先停止 example 的程序，再使用手柄将机器人切换到其他模式。

此示例相较于示例一多涉及一个话题 geometry_msgs，该类包含三维空间下的速度，如示例一所示，用户需要创建该命令的发布者，例如

```
self.vel_cmd_publisher = self.create_publisher(Twist, self.vel_cmd_topic_name,
 1)
```

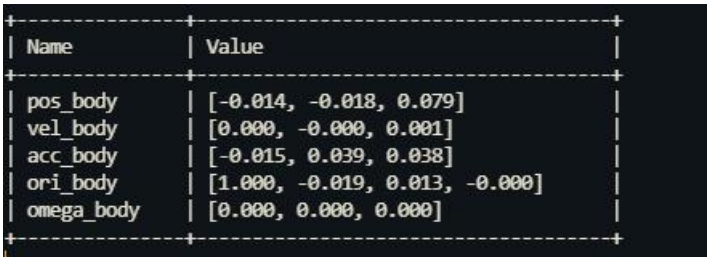
用户可自行定义机器人在三个方向上的速度，三个方向为 x、y、yaw(angular z axis)。参考代码如下

```
controlMsg = Twist()
controlMsg.linear.x = fXSpeed * 0.5
controlMsg.linear.y = fYSpeed * 0.4
controlMsg.angular.z = fYawSpeed * 0.35

self.vel_cmd_publisher.publish(controlMsg)
```

3) 示例三：robot_state

本实例旨在读取机器人相关的数据信息，以便于后续调试与二次开发，robot_state 包含多个变量，此处以 pos_body、vel_body、acc_body、ori_body、omega_body 为例。正常情况下，运行该示例后六足机器人会停留在力控站立模式，并可在终端观察到这些值的数据，如下所示



运行该示例后，用户可通过手柄在力控站立模式下测试各关节运行情况。并可在控制台观察到上

述数据的变化。例如，用户可以通过左摇杆纵轴使得机器人底座下降，此时可在终端观察到 pos_body 有明显变化。

注意：停止程序时请先停止 example 的程序，再使用手柄将机器人切换到其他模式。

关节控制 SDK

C++关节控制

motor_sdk 是越疆科技基于自研多关节机器人通信板的上位机软件，主要用于机器人的控制、调试、数据获取和二次开发。文件夹位于运动控制 PC 系统/home/robot(用户家目录)下的 robot_controller_release 文件夹。

头文件说明如下：

头文件	描述
absolute_timer.h	定义了获取系统时间函数，绝对等待时间类，用于控制通信频率，单位：秒
comm.h	定义了通讯数据结构体：电机，IMU，电池数据结构体
error_class.h	定义了 IO 口异常类，用于打印异常状态
IOPort.h	定义了通用 IO 通信类，用于其它通讯的基类
serial_port.h	定义了串口通信类，用于 USB 串口通信
UDPPort.h	定义了 UDP 通信类，用于以太网口通信
motor_sdk.h	定义了电机 SDK 类，用于提供通讯板的上位机 API
robot_sdk.h	定义了机器人 SDK 类，将机器人控制，通信，监控等多线程封装，供用户直接使用
periodic_function.h	定义了周期性运行任务函数

⚠例程文件位于/motor_sdk/example 中，在运行关节控制 sdk 例程之前，需要先杀死机器人本身的控制程序

```
ps -ef | grep start_controller.sh
sudo kill <progress ID>
ps -ef | grep main
sudo kill <progress ID>
```

然后编译

```
cd ~/motor_sdk
mkdir build && cd build
cmake ..
make
```

在运行这两个例程之前，请先将六足机器人六肢折叠放置于平坦地面。

```
./motor_read miniHex_v2
./motor_wave miniHex_v2
```

提供两个例程作为参考，正常情况下，motor_read 会在控制台原地打印 18 个电机的角度和输出力矩。用户可在此时手动摆动各关节电机，可以感受到电机所带的轻微阻尼，并在终端输出中可以看到

对应关节的角度变化。

而 motor_wave 程序旨在测试电机指令的写入。该程序会将一组预设的电机指令发送给电机，用户可以看到各关节都会以一定周期小幅度运动。下面以 motor_wave 程序为例，对核心代码进行简要说明：

```
class Custom {
public:
    void damp(RobotCommand &command) {
        command.qDes.setZero();
        command.qdDes.setZero();
        command.tauDes.setZero();
        command.kpDes.setZero();
        command.kdDes.setConstant(0.5);
    }
    void controlFunction(const RobotState &state, RobotCommand &command) {
        if (control_iter < 10) {
            q_init = state.q;
            std::cout << "get q init " << q_init.transpose() << std::endl;
        }
        else if (control_iter < 5000){
            double s = double(control_iter - 1000) / 500.0;
            for (int i = 0; i < q_init.size(); i++) {
                command.qDes(i) = q_init(i) + sin(2 * M_PI * s) * 0.2;
                command.qdDes.setZero();
                command.tauDes.setZero();
                command.kpDes.setConstant(5);
                command.kdDes.setConstant(0.25);
            }
        }
        else {
            damp(command);
            flag = true;
        }
        control_iter++;
    }
private:
    int control_iter = 0;
    DVecd q_init;
};
```

用户通过建立一个 RobotSDK 对象，提供配置文件，控制周期，以及用户自定义的控制流程函数 controlFunction 作为输入。controlFunction 的参数为系统读取的机器人状态 RobotState 以及需要用户下发的机器人控制指令 RobotCommand。Custom 是用户自定义的控制器类，如在上图所示示例中，用户在前 10 个 control_iter 周期中，读取了机器人的关节初始信息 q_init，然后在后续的 5000 周期以内，所有的关节以 q_init 为初始状态周期性位置摆动，关节刚度为 1，零速度阻尼系数为 0.2。5000 个周期之后，机器人进入到完全阻尼状态，阻尼系数为 0.5。RobotSDK 在后端进行虚拟关节和电机层的交互，

保证系统位置限制等安全。为了使得机器人的运动明显，可以在程序启动前建议把关节远离位置限制。

Python 关节控制

Pybind11 是一个用于将 C++代码与 Python 绑定的库，用于将 C++的类和函数暴露给 Python。该库路径为 motor_sdk/python_wrapper/third-party/pybind11/而 cpp 的接口 API 位于 motor_sdk/python_wrapper/robot_interface.cpp。该接口文件基于 SDK(cpp)，定义了机器人接口类，用于给电机发送指令，接受电机状态。同时使用 PYBIND11_MODULE 将该类暴露给 python。

例程文件位于/motor_sdk/python_wrapper/中，在运行关节控制 SDK 例程之前，需要先杀死机器人本身的控制程序

```
ps -ef | grep start_controller.sh
sudo kill <progress ID>
ps -ef | grep main
sudo kill <progress ID>
```

在运行例程之前，请先将六足机器人六肢折叠放置于平坦地面。

执行例程：

```
python3 motor_read.py
python3 motor_wave.py
```

正常情况下，motor_read 会在控制台原地打印 18 个电机的角度和输出力矩。用户可在此时手动摆动各关节电机，可以感受到电机所带的轻微阻尼，并在终端输出中可以看到对应关节的角度变化。

而 motor_wave 程序旨在测试电机指令的写入。该程序会将一组预设的电机指令发送给电机，用户可以看到各关节都会以一定周期小幅度运动。下面以 motor_wave 程序为例，对核心代码进行简要说明：

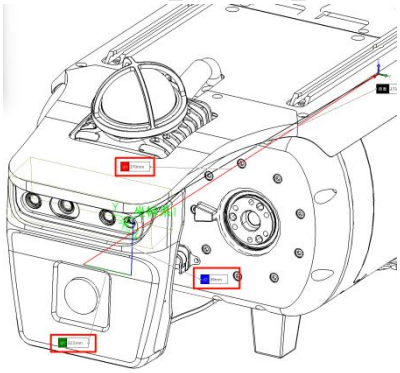
```
def step(self):
    self._command = self._q_init
    self._count = 0
    for t in np.arange(0.0, 10.0, self._dt):
        # print("----------- step ", self._count, "---------- ")
        s = self._count / 500.0
        self._count += 1
        for idx in np.arange(0, self._motor_num, 1):
            self._kpDes[idx] = 5.0
            self._kdDes[idx] = 0.25
            self._qDes[idx] = self._q_init[idx] + math.sin(2*math.pi*s) * 0.2
            self._qdDes[idx] = 0.0
            self._tauDes[idx] = 0.0

            self._interface.send_command(
                self._kpDes, self._kdDes, self._qDes, self._qdDes, self._tauDes
            )
            self._observation = self._interface.receive_observation()
            self.wait()
```

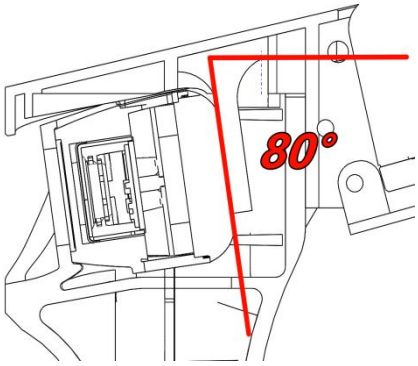
核心点在于用户需要构建一个 RobotInterface 的实例对象，并通过该实例与底层电机进行通信。为了使得机器人的运动明显，可以在程序启动前建议把关节远离位置限制。

深度相机

机器人系统预装了 Realsense D435 相机，机器人系统中集成了可以获取相机数据信息的节点，安装至系统目录：~/robot_controller_release/ros2_packages/，深度相机结构安装参数：



相机与几何中心坐标系相对数据



相机平面与水平夹角 80°

启动节点

深度相机的节点需要进入 jetson platform(nano/nx)开启，其 ip 地址参考网络与访问章节可知为 192.167.1.20。

```
ssh robot@192.168.1.20  # 密码 123
```

在 ros2_pacakges 下执行命令

```
source ./setup.bash
ros2 launch realsense_camera_node start_node.launch.py
```

应该看到如下输出

```
robot@orin-nx-super-2204:~/robot_controller_release/ros2_packages$ ros2 launch realsense_camera_node start_node.launch.py
[INFO] [launch]: All log files can be found below /home/robot/.ros/log/1970-02-12-08-14-51-502848-orin-nx-super-2204-3707
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [realsense_camera_node-1]: process started with pid [3708]
[realsense_camera_node-1] [INFO] [0003629693.149966688] [realsense_camera_node]: realsense_camera_node started
[realsense_camera_node-1] [INFO] [0003629693.177001568] [realsense_camera_node]: 1 Realsense Camera Detected.
[realsense_camera_node-1] [INFO] [0003629693.983069824] [realsense_camera_node]: Start 408122070053 Camera Pipeline Success!
[realsense_camera_node-1] [INFO] [0003629696.170387328] [realsense_camera_node]: Create Camera Stream Process Thread OK!
[realsense_camera_node-1] [INFO] [0003629696.171110336] [realsense_camera_node]: Start 0 Camera Stream Processing ...
```

保持当前终端不关闭，新开一个终端，运行命令检查节点是否正常开启。如果正常应该看到以下新话题（sn 后的序列号不唯一）：

```
/realsense_camera_node/description
/realsense_camera_node/sn408122070053/camera_info
/realsense_camera_node/sn408122070053/color/bgr/image_raw
/realsense_camera_node/sn408122070053/depth/u16/image_raw
```

节点配置

相机节点配置文件存放目录：/home/robot/robot_controller_release/ros2_packages/realsense_camera_node/share/realsense_camera_node/config，相机节点配置文件为：realsense_camera_node_parameters_setting.yaml 主要内容为：

```
realsense_camera_node:
  ros__parameters:
    system_monitor_topic: "system_monitor_defined"
```

```
    camera_image_frame_width: 640
    camera_image_frame_height: 480
    camera_image_frame_rate: 30
    camera_imu_frame_rate: 200
    robot_base_frame_id: "robot_base_frame_defined"
    point_cloud_sparse_coffes: 4.0
    is_compressed_color_image: false
    is_publish_pointcloud: false
    is_depth2color: true
```

配置文件中主要参数的含义为：

- system_monitor_topic：机器人监控节点名称
- camera_image_frame_width：相机图像宽度
- camera_image_frame_height：相机图像高度
- camera_image_frame_rate：相机图像帧率
- camera_imu_frame_rate：相机 IMU 输出帧率，当前节点未对外输出 IMU 数据，有需要请联系技术支持
- robot_base_frame_id：机器人基坐标系名称
- point_cloud_sparse_coffes：点云稀疏系数，

总的点云数量为 pc_num=camera_image_frame_width * camera_image_frame_height，稀疏后 pc_num=camera_image_frame_width * camera_image_frame_height / point_cloud_sparse_coffes

- Is_publish_pointcloud：是否发布深度相机的点云图像
- is_depth2color：是否进行深度图和色彩图同步以及配准

话题发布与订阅

节点启动后会注册若干相关话题，话题发布后，用户可以使用 Rviz 等通过话题可视化数据。这些话题及其含义如下所示：

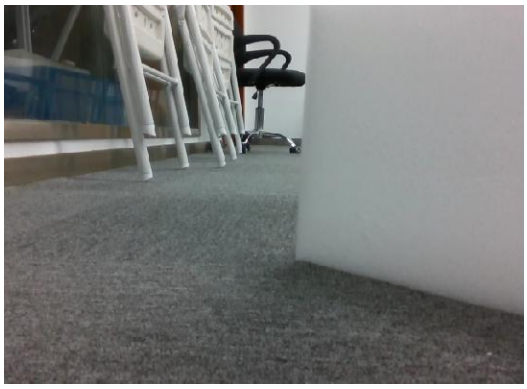
1) **/realsense_camera_node/description**
   相机节点描述，话题发布，主要介绍了相机节点的主要内容，其消息格式为 std::msg::String,组织方式为 Json 格式，可通过将该话题消息转换成 JSON 格式查看具体内容
2) **/realsense_camera_node/sn408122070053/camera_info**
   相机信息，话题发布：主要发布相机图像的长宽、内参等
3) **realsense_camera_node/sn408122070053/color/bgr/image_raw**
   相机彩色图像信息，话题发布：以 bgr 的格式发布图像消息
4) **/realsense_camera_node/sn408122070053/depth/u16/image_raw**
   相机深度图像信息，话题发布：深度图像为单通道灰度图
5) **/realsense_camera_node/sn052622070190/xyz/pointcloud**
   相机点云信息，话题发布：相机点云组织方式为 xyz,没有颜色信息
6) **/tf_static**
   相机 TF 信息，话题发布：发布相机 TF 信息，主要为机器人到相机 link 的变换，相机 link 到相机

坐标系下的变换

7) **/system_monitor_defined**

监控消息，话题订阅：订阅监控节点消息，用于执行控制节点的指令

深度图像



相机图像

程序订阅话题

文件位于 dobot_hex_sdk/realsense/src/，python 版本位于 dobot_hex_sdk/realsense/py/ 该案例旨在展示如何使用 ROS2 自定义的消息类型订阅接受话题消息。按照上述文档启动深度相机节点后，在 realsense 目录下执行建立 build 并编译后执行两个可执行文件，注意有关深度相机的操作都需要在 jetson 平台进行，否则性能用户的二次开发程序性能会有大幅下降。

1) **camera_info**，其使用的消息类型为 sensor_msgs::msg::CameraInfo，它包含了与相机的内部参数，案例程序打印了其图像宽度、图像高度以及相机内参矩阵中的关键参数包括 xy 方向的焦距和主点坐标。正常情况下用户应该会看到上述参数不断在控制台中低频打印。

```
auto qos = rclcpp::SensorDataQoS();
m_pCameraInfoSubscriber = this->create_subscription<sensor_msgs::msg::CameraInfo>(m_strDepthCameraInfoTopicName, qos, std::bind(&DepthLiteCameraInfoSubscriber::infoCallback, this, _1));
```

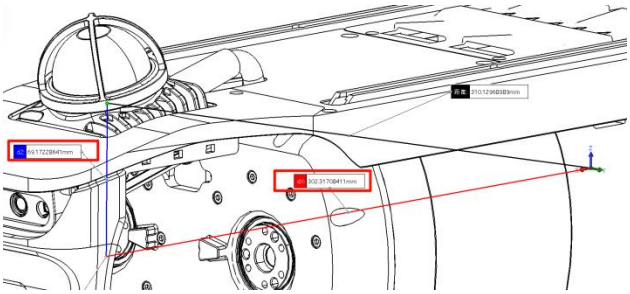
2) **camera_fps**，其涉及到的消息类型为 sensor_msgs::msg::Image，它是相机获得图像的数据类型，包括 bgr 彩色图像以及深度图像。正常情况下用户应该会看到彩色图像、深度图像的帧率、编码方式和图像尺寸不断在控制台中被高频打印。

```
auto qos = rclcpp::SensorDataQoS();
m_pDepthImageSubscriber = this->create_subscription<sensor_msgs::msg::Image>(m_strDepthImageTopicName, qos, std::bind(&DepthLiteCameraFPSSubscriber::depthCallback, this, _1));
m_pBGRImageSubscriber = this->create_subscription<sensor_msgs::msg::Image>(m_strBGRImageTopicName, qos, std::bind(&DepthLiteCameraFPSSubscriber::bgrCallback, this, _1));
```

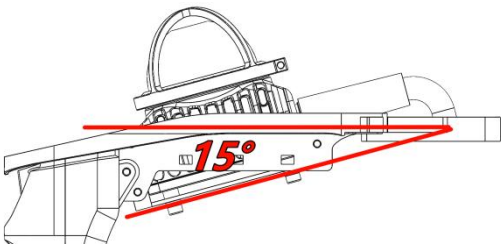
雷达

机器人系统预装了 Mid-360 激光雷达，机器人系统中集成了可以获取雷达数据信息的节点，安装至系统目录：~/robot_controller_release/ros2_packages/

雷达结构安装参数为：



雷达与几何中心坐标系相对数据



雷达平面与水平夹角 80°

启动节点

相机的开启需要进入 Intel minipc 中，其 ip 地址参考网络与访问章节可知为 192.167.12.1。

```
ssh robot@192.168.12.1  # 密码 123
```

在 ros2_pacakges 下执行命令

```
source ./setup.bash
ros2 launch livox_lidar_node start_node.launch.py
```

应该看到以下输出

```
robot@minipc-2204:~$ ros2 launch livox_lidar_node start_node.launch.py
[INFO] [launch]: All log files can be found below /home/robot/.ros/log/2025-09-22-17-01-22-982975-minipc-2204-3268750
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [livox_lidar_node-1]: process started with pid [3269075]
[livox_lidar_node-1] [INFO] [1758531683.364949990] [livox_lidar_node]: livox_Lidar_node started
[livox_lidar_node-1] [2025-09-22 17:01:23.365] [console] [info] set master/slave sdk to master sdk by default  [parse_cfg_file.cpp] [Parse] [82]
[livox_lidar_node-1] [2025-09-22 17:01:23.365] [console] [info] Livox lidar logger disable.  [parse_cfg_file.cpp] [Parse] [126]
[livox_lidar_node-1] [2025-09-22 17:01:23.366] [console] [info] Lidar ip:192.168.1.105 point cloud data and IMU data unicast is enabled.  [params_check.cpp] [CheckLidarMulticastIp] [119]
[livox_lidar_node-1] [2025-09-22 17:01:23.366] [console] [info] Data Handler Init Succ.  [data_handler.cpp] [Init] [49]
[livox_lidar_node-1] [2025-09-22 17:01:23.366] [console] [info] Init livox lidars succ.  [device_manager.cpp] [Init] [178]
[livox_lidar_node-1] [2025-09-22 17:01:23.367] [console] [info] Handle detection data, handle:1761716416, dev_type:9, sn:47MDN7N0030505, cmd_port:56100  [general_command_handler.cpp] [HandleDetectionData] [320]
[livox_lidar_node-1] [2025-09-22 17:01:23.369] [console] [info]  Receive Ack: Id 257 Seq 3  [mid360_command_handler.cpp] [Handle] [64]
[livox_lidar_node-1] [2025-09-22 17:01:23.369] [console] [info] Query Fw type succ, the fw_type:1  [general_command_handler.cpp] [QueryFwTypeCallback] [458]
[livox_lidar_node-1] [2025-09-22 17:01:23.513] [console] [info]  Receive Command: Id 258 Seq 12850  [mid360_command_handler.cpp] [Handle] [67]
```

保持当前终端不关闭，新开一个终端，运行命令检查节点是否正常开启。如果正常应该看到以下新话题：

```
/livox_Lidar_node/description
/livox_Lidar_node/sn105/imu/raw_data
/livox_Lidar_node/sn105/xyz/pointcloud
```

节点配置

雷达节点配置文件存放目录：/home/robot/robot_controller_release/ros2_packages/livox_lidar_node/share/livox_lidar_node/config，雷达节点配置文件为：livox_lidar_node_parameters_setting.yaml，其主要内容为：

```
livox_lidar_node:
    ros__parameters:
        system_monitor_topic: "system_monitor_defined"
        robot_base_frame_id: "robot_base_frame_defined"
        is_custom_pointcloud: true
```

雷达参数配置 lidar_parameters.json 文件内容为

```
{
    "MID360": {
       "lidar_net_info" : {
          "cmd_data_port"  : 56100,
          "push_msg_port"  : 56200,
```

```
        "point_data_port": 56300,
        "imu_data_port"  : 56400,
        "log_data_port"  : 56500
      },
      "host_net_info" : [
        {
          "host_ip"        : "192.168.1.20",
          "lidar_ip"       : ["192.168.1.190"],
          "cmd_data_port"  : 56101,
          "push_msg_port"  : 56201,
          "point_data_port": 56301,
          "imu_data_port"  : 56401,
          "log_data_port"  : 56501
        }
      ]
    }
}
```

其中<"host_ip": "192.168.1.10">字段用于配置雷达启动的主机 IP，<"lidar_ip": ["192.168.1.183"]>字段为雷达的 IP（默认为 192.168.1.1+**雷达序列号后两位**），雷达序列号可从雷达安装位置后方的二维码上方得知。

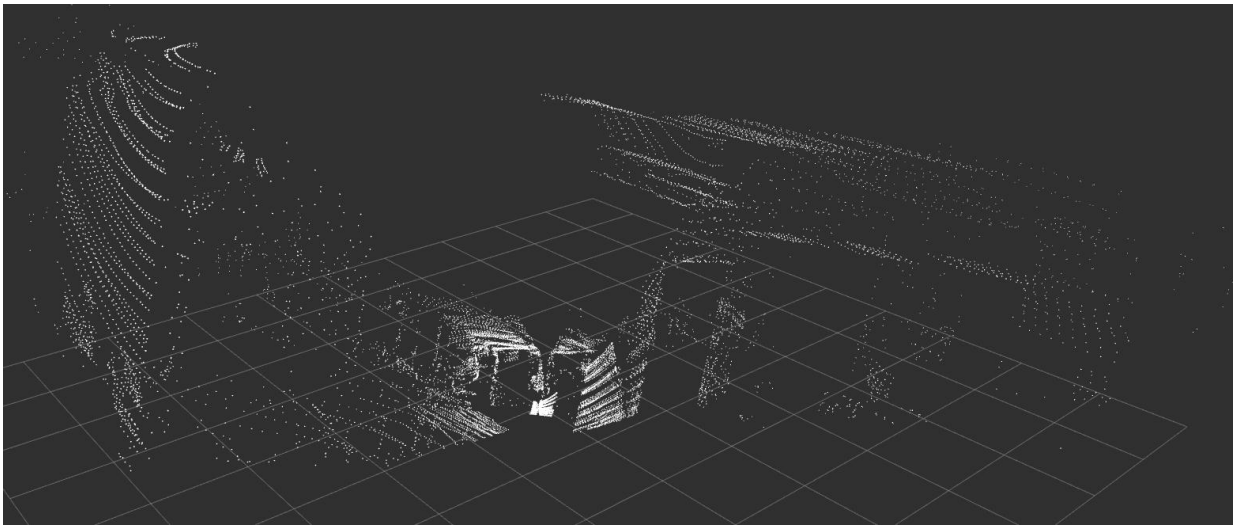
除此之外，配置文件中主要参数的含义为：

- system_monitor_topic：机器人监控节点名称
- robot_base_frame_id：机器人基坐标系名称
- is_custom_pointcloud: 雷达定制点云格式/标准点云格式切换标志

话题发布与订阅

节点启动后会注册若干相关话题，话题发布后，用户可以使用 Rviz 等通过话题可视化数据。这些话题及其含义如下所示：

1) **/livox_Lidar_node/description**
   雷达节点描述，话题发布，主要介绍了相机节点的主要内容，其消息格式为 std::msg::String, 组织方式为 Json 格式，可通过将该话题消息转换成 JSON 格式查看具体内容
2) **/livox_Lidar_node/snXXX/xyz/pointcloud**
   雷达点云信息，话题发布：雷达点云组织方式为 xyz,没有颜色信息
3) **/livox_Lidar_node/snXXX/imu/raw_data**
   雷达 imu 信息，话题发布
4) **/system_monitor_defined**
   监控消息，话题订阅：订阅监控节点消息，用于执行控制节点的指令
5) **/tf_static**
   雷达静态 TF 信息，话题发布：发布雷达 TF 信息，主要为机器人坐标系到雷达 link 的变换
6) **/tf**
   雷达动态动态 TF 信息，话题发布：发布雷达 TF 信息，主要为雷达 IMU 到雷达 link 的变换



点云图像（Rviz）

程序订阅话题

文件位于 dobot_hex_sdk/livox/src/，python 版本位于 dobot_hex_sdk/livox/py/ 本案例旨在展示如何使用 ROS2 自定义的消息类型订阅接受话题消息。按照上述文档启动深度相机节点后，在 livox 目录下执行建立 build 并编译后执行两个可执行文件。

1) **point_cloud2**，其使用的消息类型为 sensor_msgs::msg::PointCloud2，它是激光雷达扫描得到的点云的数据类型，正常情况下用户应该应该看到点云的发布频率(10hz)以及点云的数量。

```
auto qos = rclcpp::SensorDataQoS();
sub_ = this->create_subscription<sensor_msgs::msg::PointCloud2>(m_strPointCloudTopicName, qos, std::bind(&PCLiteFPSCount::cloudCallback, this, _1));
```

2) **Imu**，其涉及到的消息类型为 sensor_msgs::msg::Imu，它是雷达上的 imu 的数据类型，正常运行本案例程序，下用户应该会看到。

```
sub_ = this->create_subscription<sensor_msgs::msg::Imu>(imu_topic_, qos, std::bind(&LivoxImuLite::imuCallback, this, _1));
```

建图示例

用户在本机下载 dobot_hex_mapping，本示例适用于本示例使用 Fast-LIO2 方法作为示例。在使用 Fast-LIO2 进行建图时，IMU 的数据用于状态转移，使系统在两帧点云之间能估计持续变化的位置和姿态，而点云数据作为观测值，用于后验修正，对其预测结果和真实观测值。类似于扩展卡尔曼滤波，在预测阶段，IMU 提供高频加速度和角速度数据，算法积分得到机器人瞬态运动估计，在更新阶段，点云数据通过匹配与地图约束来修正由预测产生的累计误差，产生稳定精确的轨迹地图。

在实际运行中，Fast-Lio2 依赖一些外部组件包括 Livox-SDK2、livox_ros_driver2 和 GTSAM 等。

环境依赖

Livox-SDK2

Livox-SDK2 可以用于与 Livox 系列的激光雷达进行通信，将雷达输出的二进制数据解析成标准格

式比如点的坐标、时间戳等。在任意目录下执行下述命令

```
git clone https://github.com/Livox-SDK/Livox-SDK2.git
cd ./Livox-SDK2/
mkdir build
cd build
cmake .. && make -j
sudo make install
```

Livox_ros_driver2

livox_ros_driver2 是基于 ROS2 封装的 Livox 雷达驱动程序。它依赖 Livox-SDK2 来拿到底层数据，再把这些数据转化为 ROS2 框架下的标准话题，比如把 SDK2 解码后的点云数据转换为 sensor_msgs/PointCloud2 消息。在任意目录下执行下述命令

```
mkdir -p ws_livox/src
git clone https://github.com/Livox-SDK/livox_ros_driver2.git ws_livox/src/livo
x_ros_driver2
cd ws_livox/src/livox_ros_driver2
source /opt/ros/humble/setup.sh
./build.sh humble
```

GTSAM

GTSAM 是一个开源的因子图优化库，负责融合优化 IMU 和点云这些观测数据，通过因子图建模、非线性优化、平滑滤波等优化计算，最终输出高精度位姿和地图。在任意目录下执行下述命令

```
git clone https://github.com/borglab/gtsam
mkdir build && cd build
cmake ..
make -j
sudo make install
```

编译程序

编译之前准备环境依赖

```
source /opt/ros/humble/setup.bash
source ws_livox/install/setup.bash
```

执行编译命令

```
cd dobot_hex_mapping
mkdir build && cd build
cmake ..
make -j
```

数据采集

连接上 Hexplorer 之后，数据采集的脚本位于/home/robot/Documents/record.sh，启动 Hexplorer 并进入力控行走模式，启动数据采集脚本并控制 Hexplorer 行走，采集到的数据位于脚本同级目录，带

有时间戳标识。

数据采集完成后，将元数据配置文件和数据文件下载到本机 dobot_hex_mapping 下的 record 目录下。接下来准备运行建图程序。

运行建图

在运行之前确保程序的运行时动态库路径已知

```
echo "/usr/local/lib" | sudo tee /etc/ld.so.conf.d/usr-local-lib.conf >/dev/null
sudo ldconfig
```

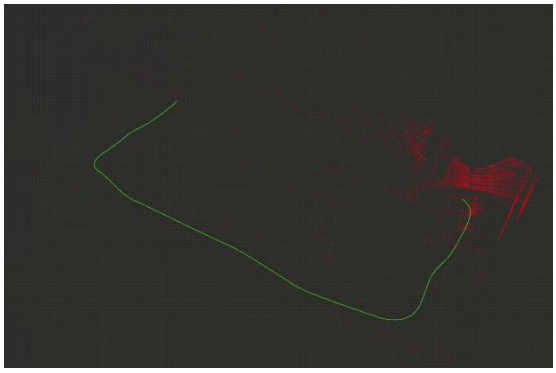
而后运行建图程序

```
./fastlio2/lio_mapping <bag_path> <config_file_path>
```

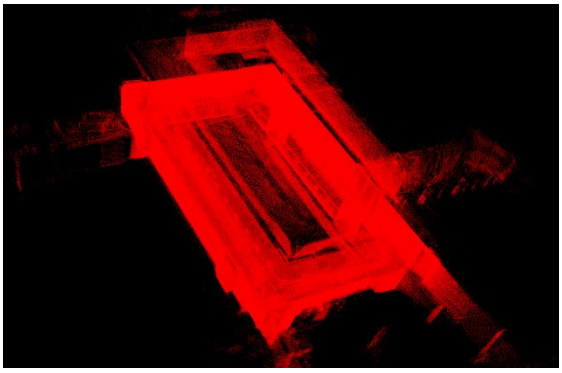
例如

```
./fastlio2/lio_mapping ../record/ ~/dobot_hex_mapping/fastlio2/config/lio_YJ.yaml
```

打开 rviz2 并导入话题 path 和 point_cloud 可以观察轨迹与点云图，并且程序执行完成后可在 record 目录下下找到建图结果，其可通过 pcl-viewer 可视化查看，例如：



Rviz 可视化



pcl-viewer 可视化

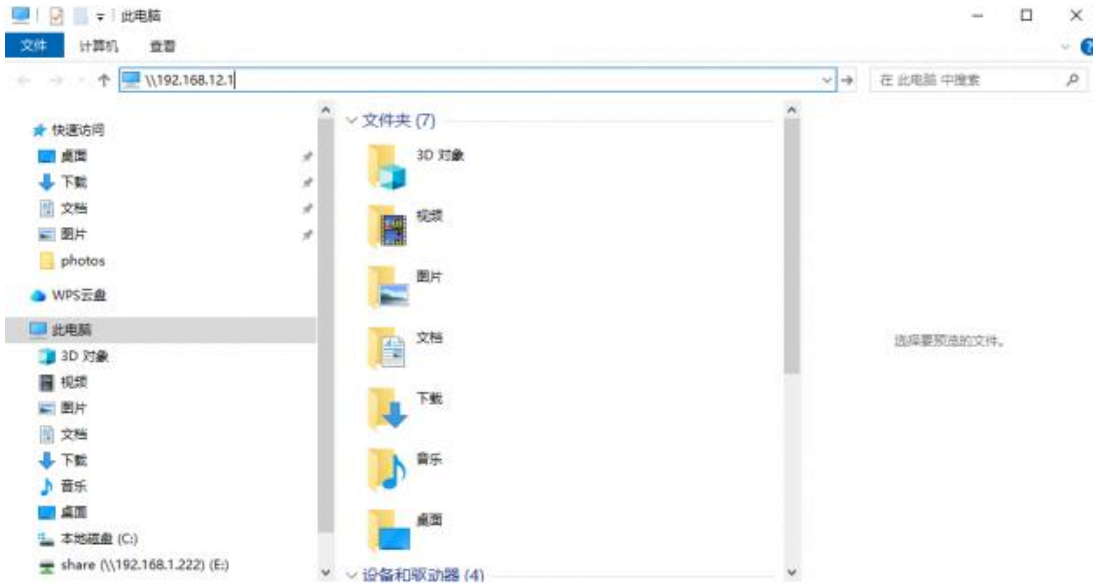
软件与硬件升级

机器人软件集成升级包，一般名为：`inffni_robotics_update_package_${platform}_${robot_id}_${verdion}.bin`，例如：`inffni_robotics_update_package_mini_pc_mini_hex_v1.6.bin` 为六足机器人，主控平台为 mini_pc，版本为 1.6 的升级包。升级机器人程序，要确保机器人处于**阻尼模式！**机器人软件升级过程中会触发机器人系统重启。

Windows 系统程序更新方法

1) 切换机器人状态为阻尼模式

2) 在文件浏览器中输入： \\192.168.12.1，连接成功后，会出现用户登陆界面用户名为：robot，密码为：123。会进入机器人升级程序共享文件夹



Samba 共享文件夹登录界面

3) 将升级程序 `inffni_robotics_update_package_${platform}_${robot_id}_${verdion}.bin` 放入文件夹。

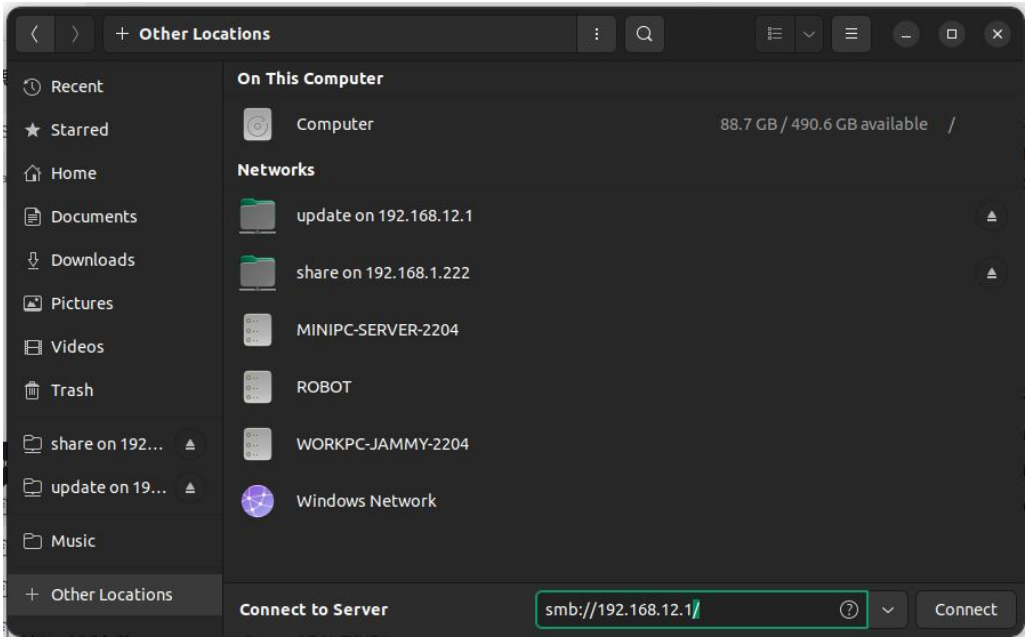
4) 机器人关机重启，或者手动进入机器人运控系统，重启机器人

5) 等待机器人升级完成，机器人升级完成后，重新登入机器人升级共享文件夹，升级成功后会有版本信息文件 version.txt。该文件中，显示的是升级包中文件对应的软件版本号，**请不要删除该文件！**

Ubuntu 系统程序更新方法

1) 切换机器人状态为阻尼模式

2) 在文件浏览器中输入： smb://192.168.12.1/，连接成功后，会出现用户登陆界面用户名为：robot，密码为：123。会进入机器人升级程序共享文件夹

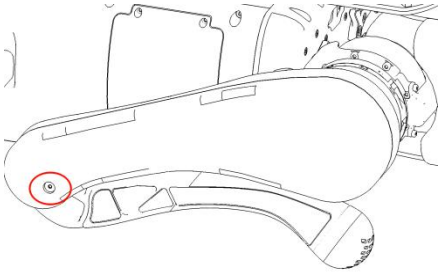


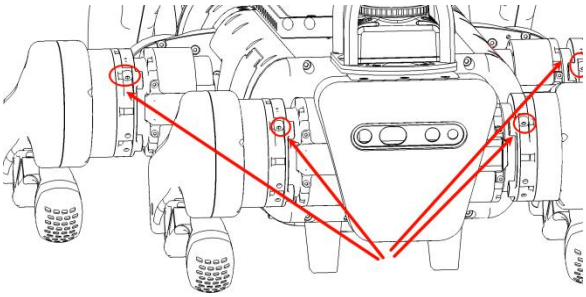
3) 升级程序 `inffni_robotics_update_package_${platform}_${robot_id}_${verdion}.bin` 放入文件夹
4) 机器人关机重启，或者手动进入机器人运控系统，重启机器人。
5) 等待机器人升级完成，机器人升级完成后，重新登入机器人升级共享文件夹，升级成功后会有版本信息文件 version.txt。该文件中，显示的是升级包中文件对应的软件版本号，**请不要删除该文件！**

日常保养与维护

机械检查

1) 主要检查腿部螺丝是否有松动情况，如上图（一个月检查一次）
   a. 六条腿的膝关节铰点螺栓，若松动，请使用 T10 梅花披头，用扭矩扳手 2NM 进行紧固
   b. 膝关节电机与大腿电机连接处 M4 杯头内六角螺栓（一条腿两个），若松动，请使用 H3.0 内六角披头，用扭矩扳手 3NM 进行紧固





2) 若遇到大腿内部或者膝关节转点处异常噪音增大，可能存在润滑脂使用消耗的问题，需要进行拆卸保养。
3) 定期检查，更换磨损过度的足端。
4) 定期联系售后检查关节电机使用情况，技术支持人员将帮助检查与重新校准。如果在定期检查中发现任何损坏，请联系售后支持。

手柄电量检查

点击手柄开关键，查看是否闪红灯，红灯代表电量低，需要充电（一个星期检查一次）。

定期清洁

1) 机器人外壳：可以使用清水或温和的清洁剂清洁机器人外壳上观察到的任何灰尘与污垢。
2) 相机镜头：使用蘸有温和清洁试剂的非研磨布清除可能的脏污。
3) 雷达罩：使用蘸有温和清洁试剂的非研磨布清除可能的脏污。
4) 雷达保护罩：定期查看雷达保护罩是否完好，有无裂痕等
5) 电气接口：充端口和连接器的损坏或异物可能导致在电气方面的危险。
6) 散热口：检查是否存在灰尘阻塞，避免过热故障。

常见问题

硬件问题

如何判断设备是否上线？

1. 打开开关 2s 后，听到电机沙沙的声音，代表电机和主控都已上电。
2. 约 40s 后，打开手柄开关，通过手柄控制机器人，看机器人是否进入位置折叠模式。

电池电量问题

- 低于 40V：电池总压过低保护，电池组会重启上电；
- 低于 42V：运控程序保护，机器人进阻尼模式；
- 低于 47V：LCD 显示屏显示为 0%
- 高于 57V：LCD 显示屏显示为 100%

软件算法问题

关节标零

1. 将机器人从起始位置（机器人趴在地上）上电。
2. 先杀掉自启动脚本 start_controller，再杀掉运动程序；

```
ps -ef | grep start_controller.sh
sudo kill <progress ID>
ps -ef | grep main
sudo kill <progress ID>
```

3. 运行标定程序；

```
cd ~/robot_controller_release/executable
./motor_set_zero miniHex2
```

4. 正常情况下应该可以从控制台看到每个关节的 pos 都在 1e-2 以下。

关节标定

5. 将机器人从起始位置（机器人趴在地上）上电。
6. 先杀掉自启动脚本 start_controller，再杀掉运动程序；

```
ps -ef | grep start_controller.sh
sudo kill <progress ID>
ps -ef | grep main
sudo kill <progress ID>
```

7. 运行标定程序；

```
cd ~/robot_controller_release/executable
./motor_test_calibration miniHex_v2
```

8. 将要标定的关节从最大限位转动到最小限位，重复两次；
9. 复制对应关节的 offset 数据，将 yaml 文件中的对应 offset 数据替换。