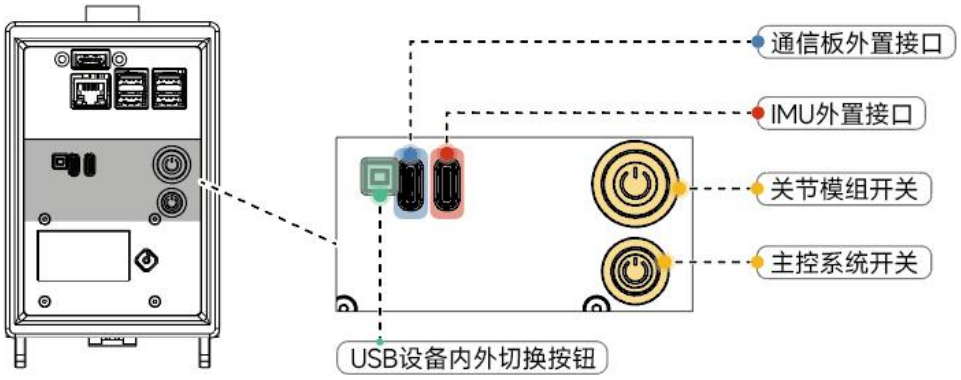
HIGH TORQUE 高擎机电

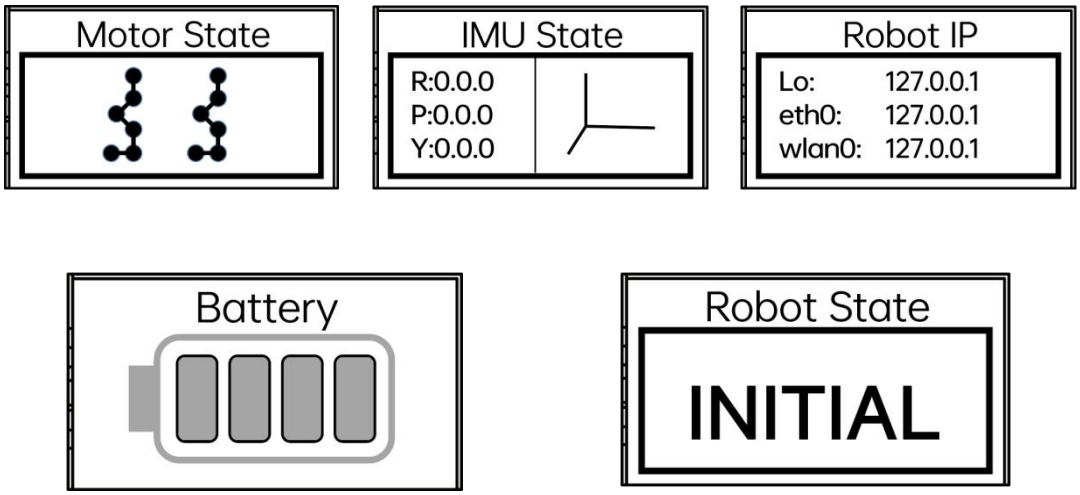
目 录

一、机器本体功能

（一）外接算力平台



（二）状态显示屏的使用



（三）远程连接

通过 Nomachine 连接无需连外接显示器:

注：1. 需要给机器人设备安装 Nomachine（一般为 Arm 版本），同时给远程开发电脑安装 Nomachine【一般为 X86（即 amd 版本）】

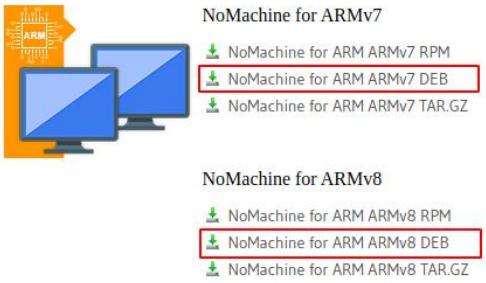
2. Orin_NX 或标准 X86 计算机可能需要显卡欺骗器才能正常使用。

步骤 1.机器人端安装软件（ARM 主控电脑安装）（请根据您的远端计算机系统选择对应版本）

软件下载链接：`https://downloads.nomachine.com/`

百度网盘链接：链接：`https://pan.baidu.com/s/13D5y01dbSP4-BkKijoVXEQ?pwd=jub7` 提取码：`jub7`

步骤 2.选择.deb 格式的安装包

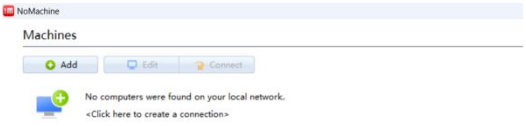


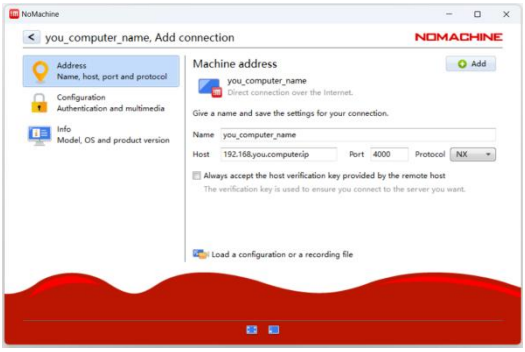
步骤 3.通过以上方式下载软件，随后终端输入：`sudo apt install ./xxx_xxx.deb` 自动获取依赖并安装软件

步骤 4.获取机器人主控 IP 地址

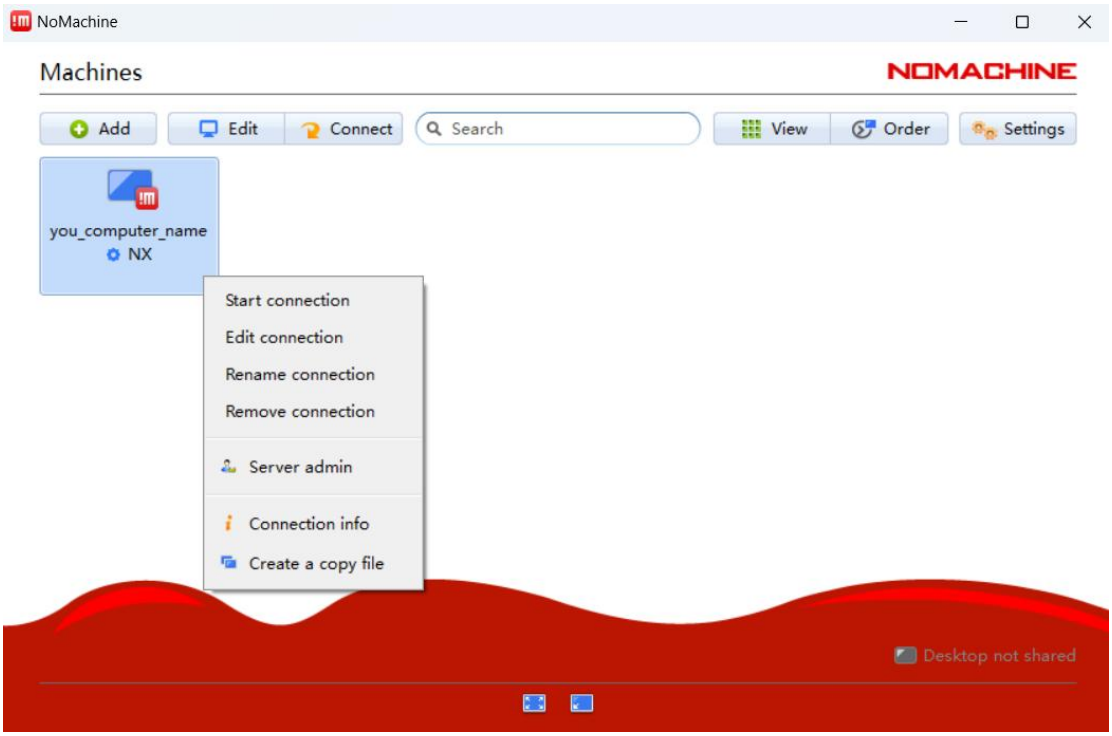
```
Bash
#在机器人主控查找 ip
ifconfig -a
```

步骤 5.在远端主机打开 nomachine 软件,点击 add 添加机器人主控的 IP





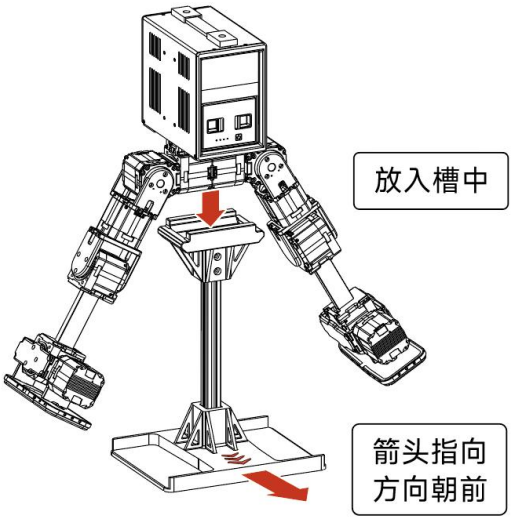
步骤 6.添加完计算机后，双击或右键图标，点击 start connection 即可连接机器人主控。

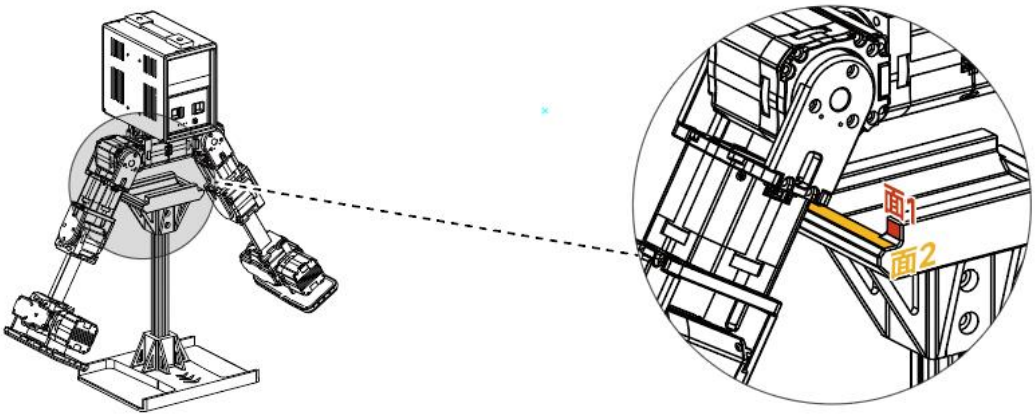


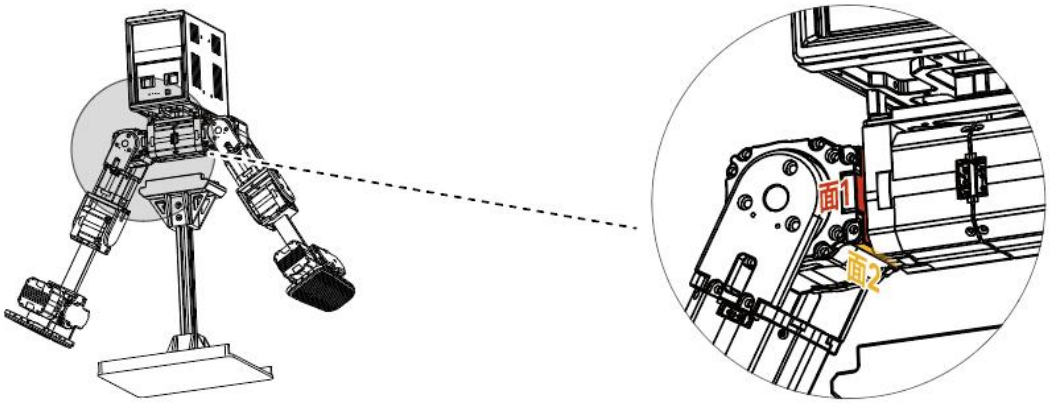
二、配件功能

（一）零位支架的使用

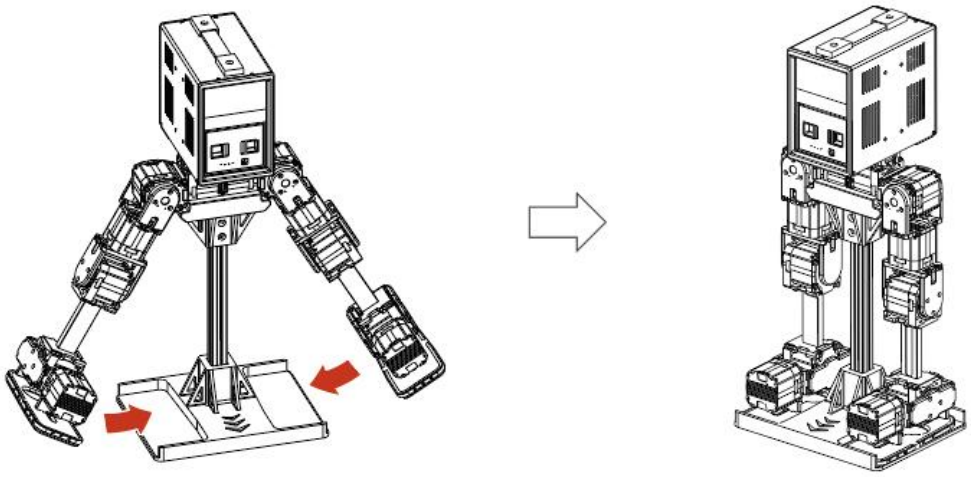
1. “零位支架”带标识箭头朝前；
2. 将机器人按照上述四个步骤确认好使用前的位姿；
3. 将机器人双腿分开适当角度后，将髋部两个电机置于“零位支架”上端，对齐后放入槽中，“面 1”与“面 1”对应、“面 2”与“面 2”对应，请参照下图示例；



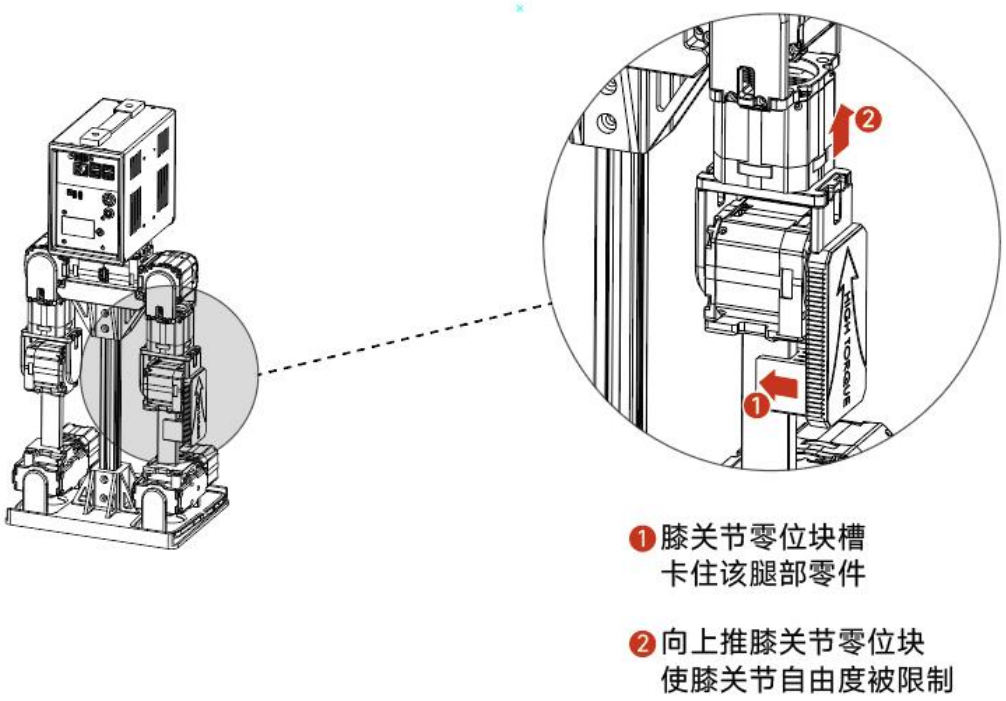




4. 将机器人双腿向中间收拢，使脚掌前后及内侧贴紧“零位支架”下部槽中对应边；

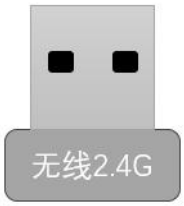


1. 将“膝关节零位块”下部槽口嵌入如图所示腿部零件，然后向上推，使得膝关节被锁紧。



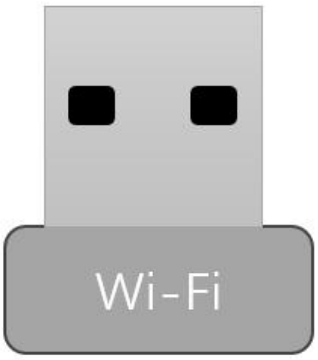
（二）无线控制器与无线接收器

第一次使用无线控制器时，需要先配对手柄（具体连接说明详见快速用户手册）。

无线控制器与无线接收器		

（三）无线网卡

由于香橙派 5Pro 开发模块支持不完全，在配置完手柄驱动后板载 Wi-Fi 网卡会失去功能，如更新软件包、连接远程服务等操作极为不便，因此需要搭配无线网卡才能满足机器人的工作需求（Linux 免驱）。

无线网卡（600mbp/s）	

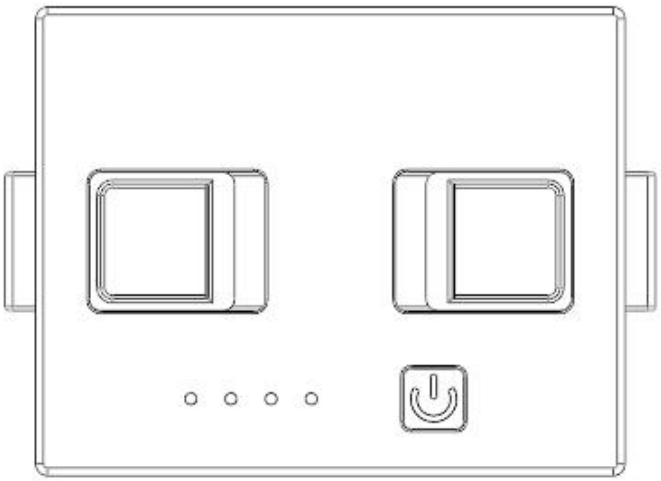
注：除 orangepi 5 系列主控含 Wi-Fi 网卡，之外其他型号主控（如 Jetson Orin NX、LubanCat 4 等）皆不需要此配件

我们在机器人随行的配件包提供了 Wi-Fi 网卡，此产品为第三方 Linux 免驱网卡解决方案，即插即用。

1. 建议插入任意 USB 2.0 设备口上，在使用网卡时注意请在关机情况下插拔此网卡
2. 在正确配置 Wi-Fi 后建议不要轻易取下网卡，防止 IP 地址随机分配导致的远程配置失效

（四）电池与充电仓

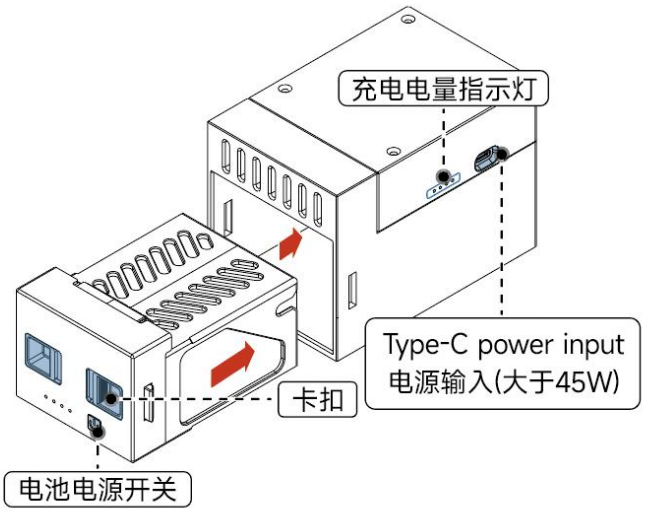
1. **电池电源的使用和功能**



电池电源及功能：我们的电池适配了 BMS 电池管理系统，可对电池进行监控，提供过流保护，同时可以避免过充和过放的现象发生。电池上状态指示灯及功能按键使用方法如下：

- 查看电池状态：在电池未启动时，短按电池一次即可显示当前电池电量（四个灯柱表示四个百分比）
- 启动电池：电池设有保护功能，要启动电池需要先短按一次随后长按大于 1 秒随后放开，当电量指示灯常亮则表示电池已正常开启；
- 关闭电池：要关闭电池也是同样的操作，先短按一次电源按键，随后长按电源按键大于 1 秒，电量指示灯熄灭即表示电池已经关闭。

2. 电池充电与电池仓安装



按照上图所示方式安装电池到电池充电仓。

安装方法：电池上有两个卡扣结构，同时用力夹紧两个卡扣，按上图所示方向插入充

电仓，推到最深处让两个卡扣松开，使之锁紧到充电仓，即可将电池安装在充电仓中

3. **充电**

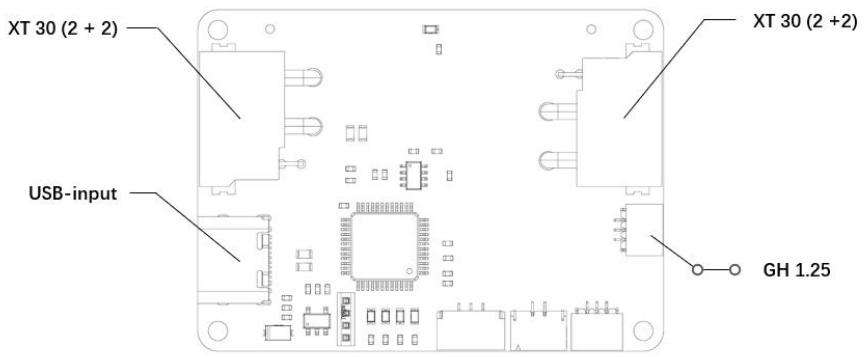
- 将数据线和电源式配器（大于 45W）正确连接并插入充电仓。
- 进入充电模式：本产品电池采用 BMS 健康管理，需要将电池电源打开，随后 BMS 会自动识别工作状态进入充电模式。
- 电池电量显示：电池上有 4 个灯柱，分别对应 4 个电量百分比。
  - 充电过程中灯柱会依次被点亮；
  - 充满电时电池自动断电，所有灯柱全部熄灭。

（五）USB 转 CAN FD 调试板

CAN FD 调试板通常用于测试电机功能的完整性。在机器人使用过程中，当拿到一个新的关节模组或出现关节无法连接、发热、运动阻塞或无力矩输出等问题时，可以通过 CAN FD 调试板进行调试。将目标关节（一个、多个或整个腿部、手臂）与上位机软件链接，独立测试，以辅助分析问题，或提供解决思路。

上位机软件及使用方法：
`https://pan.baidu.com/s/1__ic2KSMTIpXrgGTskjE8Q?pwd=5k8f`

提取码: `5k8f`

USB 转 CANFD 调试板	

1. 在使用时，左边接入一根 USB Type-C 数据线，XT 30 (2 + 2)接口可选择接入 XT 30 电源（如待测试关节有其他形式电源供电，此处可不接任何线材）
2. 将右边 XT 30 (2 + 2) 接口接入一根 XT 30 (2 + 2)线材与关节模组相连接
3. 将数据线与上位机电脑相连接，随后打开关节模组电源
4. 在计算机中找到上位机软件，随后软件会自动识别所连接的关节模组，并返回 ID 等有关信息

三、二次开发

（一）SDK 软件包与接口

这是一个双足机器人控制和通信的软件接口包。

通过这个接口包，可以更方便地与双足机器人底层硬件通信，完成状态信息获取和控制电机动作。

1. 安装依赖

首先机器人需要安装 ROS-noetic 版本。

- 这里建议使用 fishros 的一键安装，安装桌面版。注意如果已经预装 ros noetic，则该步骤可以跳过。

```
Shell
wget http://fishros.com/install -O fishros && . fishros
```

- 安装串口通信的相关包。
  - sudo apt-get install libserialport0 libserialport-dev
- 安装 python 依赖
  - python3 -m pip install empy

2. 编译

a. 克隆代码到本地

```
Shell
git clone https://github.com/HighTorque-
Robotics/livelybot_robot.git
```

b. 编译

```
Shell
cd livelybot_robot
catkin_make
```

3. 测试

a. IMU 设备测试

- 给测试脚本增加执行权限：`chmod +x test_yesense_imu.sh`;

- 执行测试脚本：`./test_yesense_imu.sh`;
- 在启动的 RVIZ 界面，观察 IMU 的姿态。

b. 电机控制测试 1

- 准备主控板和电机若干;
- 根据配置文件 `lively_description/robot_param/1dof_STM32H730_model_test_Orin_params.yaml` 中的配置信息，在 can0 上连接 1 个电机，电机 ID 为 `1`。
- 修改配置文件 `livelybot_description.launch` 中的参数，将 `dof_type` 设置为 `1`;
- 添加执行权限：`chmod -R 777 test_motor_run.sh`;
- 执行电机测试程序`./test_motor_run.sh`。

c. 电机控制测试 2

- 准备主控板和电机若干;
- 根据配置文件 `lively_description/robot_param/6dof_STM32H730_model_test_Orin_params.yaml` 中的配置信息，在 can0 上连接 3 个电机，电机 ID 分别为 `1,2,3`，在 can1 总线上连接 3 个电机，电机 ID 分别为 `1,2,3`。
- 修改配置文件 `livelybot_description.launch` 中的参数，将 `dof_type` 设置为 `6`;
- 执行电机测试程序`./test_motor_run.sh`。

d. 电机数据反馈测试

- 准备主控板和电机若干;
- 根据配置文件 `lively_description/robot_param/6dof_STM32H730_model_test_Orin_params.yaml` 中的配置信息，在 can0 上连接 3 个电机，电机 ID 分别为 `1,2,3`，在 can1 总线上连接 3 个电机，电机 ID 分别为 `1,2,3`。
- 修改配置文件 `livelybot_description.launch` 中的参数，将 `dof_type` 设置为 `6`;
- 执行电机测试程序`./test_motor_feedback.sh`。

4. IMU 信息读取

a. 在 `livelybot_robot` 目录下，执行命令 `source devel/setup.bash`;

b. 查看 imu 设备在系统中的设备名称：`ls /dev`，一般为 `ttyACM*`或者 `ttyUSB*`;

c. 添加设备权限：`sudo chmod -R 777 /dev/ttyACM*`或 `sudo chmod -R 777 /dev/ttyACM*`;

d. 启动 IMU 设备节点：`roslaunch yesense_imu run_without_rviz.launch`;

e. 查看 IMU 节点发布的话题消息列表：`rostopic list`;

f. 查看 IMU 姿态数据：`rostopic echo /yesense/sensor_data`;

g. 如果需要图形化界面，运行：`roslaunch yesense_imu yesense_rviz.launch`，可以通过 RVIZ 界面观察 IMU 的姿态。

5. 电机控制

a. 连接主控板，查看主控板在系统的设备名称：`ls /dev`，一般为 `ttyACM*`或者 `ttyUSB*`;

b. 添加设备权限：`sudo chmod -R 777 /dev/ttyACM*`或 `sudo chmod -R 777 /dev/ttyACM*`;

c. 编写配置文件，命名和内容参考：

`lively_description/robot_param/6dof_STM32H730_model_test_Orin_params.yaml`;

d. 创建机器人对象：`livelybot_serial::robot rb`

e. `rb` 的成员变量 `Motors` 为存放电机对象的容器，包含配置文件描述的所有电机的对象，电机在容器中的位置顺序为 `canport0` 的一号、二号、三号电机，依次类推，然后是 `canport1` 的一号、二号、三号电机，依次类推，依次是 `canport3`、`canport4`（如果有点话），依次类推;

f. 获取电机对象:

```Shell
auto it = rb.Motors.begin();
motor *m = &(*it+N);
```

其中，N 是电机在容器中的位置，从 0 开始;

g. 执行控制指令（保存指令到缓存区）：

- `m->fresh_cmd_int16(pos, vel, tor, kp, ki, kd, acc, voltage, current);` // 调试、通用模式（需要在配置文件中配置工作模式，根据工作模式填入对应的参数，其他参数无效）
- `m->position(pos);` // 位置模式

- `m->velocity(vel);` // 速度模式
- `m->torque(tor);` // 力矩模式
- `m->voltage(volt);` // 电压模式
- `m->pos_vel_MAXtqe(pos, vel, tor_upper);` // 位置、速度模式，带力矩上限
- `m->pos_vel_tqe_kp_kd(pos, vel, tor, kp, kd);` // PD 位置速度+前馈扭矩模式
- `m->pos_vel_kp_kd(pos, vel, kp, kd);` // PD 位置速度模式

h. 发送命令（发送到电机控制板）：`rb.motor_send_2()`。

6. 电机状态获取

a. 参考`上一章电机控制的`前五个步骤;

b. 获取电机对象:

```
Shell
auto it = rb.Motors.begin();
motor *m = &(*it+N);
```

其中，N 是电机在容器中的位置，从 0 开始;

c. 获取电机状态：`m->get_current_motor_state()`, 返回值类型为 `motor_back_t *`，结构体变量如下:

```
Shell
typedef struct motor_back_struct
{
    uint8_t ID;     // 电机 ID
    float position; // 位置
    float velocity; // 速度
    float torque;   // 力矩
} motor_back_t;
```

7. 使用例程

a. 电机控制示例代码:

```
Shell
./livelybot_serial/test/test_motor_run.cpp
```

b.电机数据反馈示例代码:

```Shell
./livelybot_serial/test/test_motor_feedback.cpp
```

8. 电机保护功能

a. 位置保护

- 设置配置文件在 `motorX` 标签下添加 `pos_upper` 和 `pos_lower` 参数，分别表示位置上限和下限，添加 `pos_limit_enable` 参数，表示是否开启位置保护；
- 设置示例如下：

```Shell
motor1:
    type: 1
    id: 1
    name: "L_low_foot"
    num: 1
    pos_limit_enable: true
    pos_upper: 10
    pos_lower: -10
```

- 上述示例中，`pos_limit_enable` 为 1， 表示开启位置保护，`pos_upper` 和 `pos_lower` 分别为 10 和-10，表示位置上限和下限；

b. 扭矩保护

- 设置配置文件在 `motorX` 标签下添加 `tor_upper` 和 `tor_lower` 参数，分别表示位置上限和下限，添加 `tor_limit_enable` 参数，表示是否开启位置保护；
- 设置示例如下：

```Shell
motor1:
    type: 1
    id: 1
    name: "L_low_foot"
    num: 1
    tor_limit_enable: true
    tor_upper: 5
    tor_lower: -3
```

- 上述示例中，`tor_limit_enable` 为 1， 表示开启扭矩保护，`tor_upper` 和 `tor_lower` 分别为 5 和-3，表示位置上限和下限。

（二）yaml 配置文件说明

`rb.Motor` 是一个 `vector<motor*>`列表，可以理解为一个装载了机器人上所有的电机接口的一个列表，可以通过这个列表去控制机器人上的电机。但是这个列表的顺序是机器人的配置文件决定的。

- sdk 提供了多个机器人配置文件。
- 以 12 自由度的双足机器人为例，编写配置文件如下：

```
Plain Text
robot:
  SDK_version: 3.0
  robot_name: "Pi Robot"
  arm_dof: 0
  leg_dof: 12
  Serial_Type: "/dev/ttyACM"  # 5361 的
  Seial_baudrate: 4000000
  CAN_Type: "CAN-FD BRS" # "CAN-FD"\"CAN-FD BRS"\"CAN 2.0B"
  control_type: 9  # 运行模式
  CANboard_type: "STM32 H730"
  CANboard_num: 1
  CANboard:
    No_1_CANboard:
      CANport_num: 2
      CANport:
        CANport_1:
          motor_num: 6
          motor:
            motor1:{type: 2, id: 1, name: "L_low_foot", num: 1,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor2:{type: 2, id: 2, name: "L_high_foot", num: 2,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor3:{type: 3, id: 3, name: "L_knee_pitch", num: 3,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor4:{type: 3, id: 4, name: "L_hip_pitch", num: 4,
```

```
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor5:{type: 3, id: 5, name: "L_hip_roll", num: 5,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor6:{type: 3, id: 6, name: "L_hip_yaw", num: 6,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
        CANport_2:
          motor_num: 6
          motor:
            motor1:{type: 2, id: 1, name: "R_low_foot", num: 1,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor2:{type: 2, id: 2, name: "R_high_foot", num: 2,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor3:{type: 3, id: 3, name: "R_knee_pitch", num: 3,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor4:{type: 3, id: 4, name: "R_hip_pitch", num: 4,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor5:{type: 3, id: 5, name: "R_hip_roll", num: 5,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
5,
              tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
            motor6:{type: 3, id: 6, name: "R_hip_yaw", num: 6,
              pos_limit_enable: false, pos_upper: 5, pos_lower: -
```

```
5,
                tor_limit_enable: false, tor_upper: 5, tor_lower: -
3}
```

根据以上文件，可以把配置文件分为以下 5 个部分：

1. 机器人根节点
   - 配置文件以 robot:开始;
2. 软硬件参数说明
   - `SDK_version: 3.0` 表示使用的 SDK 软件版本为 3.0;
   - `robot_name: "Pi Robot"` 表示机器人的名字为 `Pi Robot`;
   - `arm_dof:0` 表示机器人的手臂自由度为 0，`leg_dof: 12` 表示机器人的腿部自由度为 12;
   - `Serial_Type: "/dev/ttyACM"` 表示机器人通信板的串口名称为 `/dev/ttyACM*`;
   - `Seial_baudrate:4000000` 表示机器人通信板的串口波特率为 4000000;
   - `CAN_TYPE: "CAN-FD"` 表示机器人的 CAN 总线类型为 `CAN-FD`，可选项有：`"CAN-FD"、"CAN 2.0B"`;
   - `control_type: 9` 表示机器人的控制模式为 9，可选项有：`0、1、2、3、4、5、6、9、10`（7、8 模式已弃用），分别对应位置模式、速度模式、力矩模式、电压模式、电流模式、位置-速度-最大力矩模式、带 Kp-Kd 参数的位置-速度-力矩模式、带 Kp-Kd 参数的位置-速度模式;
   - `CANboard_type: "STM32 H730"` 表示机器人的通信板主控为 `STM32 H730`，可选项有：`"STM32 H730"、HPM 5361`;
   - `CANboard_num: 1` 表示机器人通信板个数为 1;
3. 通信板配置信息
   - `CANboard:` 表示机器人的通信板信息，包含 CAN 总线数和每条 CAN 总线上的电机数;
   - `No_1_CANboard:` 表示第一块通信板的节点;
   - `CANport_1:` 表示第一条 CAN 总线上的电机信息;
   - `motor_num: 6` 表示这条 CAN 线上的电机个数;
4. 电机配置信息
   - `motor_1:` 表示这条 CAN 线上的第 1 个电机信息;

- `type:1`: 电机类型，可选项有：`1、2、3、4、5、6、7`，分别对应 `5046` 电机-20 减速比、`4538` 电机-20 减速比、`5047` 电机-36 减速比、`5047` 电机-9 减速比、`4438` 电机-32 减速比、`4438` 电机，`8` 减速比、`7136` 电机-9 减速比;
- `id: 1`:电机 ID
- `name: "L_hip_yaw"`:电机名称;
- `pos_limit_enable: false`:位置保护使能标识，`true` 表示使能，`false` 表示不使能;
- `pos_upper: 5`:位置上限，单位是圈数;
- `pos_lower: -5`:速度下限，单位是圈数;
- `tor_limit_enable: false`:力矩限制使能标识，`true` 表示使能，`false` 表示不使能;
- `tor_upper: 5`:力矩上限，单位是 Nm;
- `tor_lower: -3`:力矩下限，单位是 Nm;

（三）机器人 launch 文件说明

launch 文件作用是在一个脚本文件内启动若干个 ros 节点，定义 ros 参数。

1. 控制机器人上的电机，主要使用 roslaunch 脚本。sdk 的 roslaunch 脚本主要放在 livelybot_robot/src/livelybot_bringup/launch 路径下，以 test_motor_run.launch 脚本为例，编写 launch 文件如下:

```
Plain Text
<launch>
  <include file='$(find
livelybot_description)/launch/livelybot_description_robot.launch'
/>
  <node pkg="livelybot_serial" name="test_motor_run"
type="test_motor_run" output="screen" />
</launch>
```

2. 其中，第二行表示引用了 livelybot_description 包下 launch 目录下的 livelybot_description_robot.launch 脚本,而 livelybot_description_robot.launch 脚本内容如下:

```
Plain Text
<launch>
    <arg name = "dof_type" default = "12"/>
```

```
    <arg name = "mcu_type" default = "STM32H730"/>
    <arg name = "model_type" default = "test"/>
    <arg name = "design" default = "Orin"/>
    <rosparam file="$(find
livelybot_description)/robot_param/$(arg dof_type)dof_$(arg
mcu_type)_model_$(arg model_type)_$(arg design)_params.yaml"
command="load" />
</launch>
<!-- 注释
pkg: 节点所在的功能包名称
type: 节点的可执行文件名称
name: 节点运行时的名称
output ="log | screen" (可选)，日志发送目标，可以设置为 log 日志
文件，或 screen 屏幕,默认是 log
respawn ="true | false" (可选)，如果节点退出，是否自动重启
required="true | false" (可选)，该节点是否必须，如果为 true,那么
如果该节点退出，将杀死整个 roslaunch
ns="xxx" (可选)，在指定命名空间 xxx 中启动节点
machine="机器名"，在指定机器上启动节点
args="xxx xxx xxx" (可选)，将参数传递给节点
-->
```

其核心内容是:

```
Plain Text
<rosparam file="$(find
livelybot_description)/robot_param/$(arg dof_type)dof_$(arg
mcu_type)_model_$(arg model_type)_$(arg design)_params.yaml"
command="load" />
```

这一行内容表示把提供的机器人配置文件内的参数导入到 ros 的工作空间内，那么在代码里的 robot 类初始化时，便可以读取到配置文件里的参数，那么就可以使用一套代码，适应不同的机器人电机的连接方式以及电机数量。

3. 第三行表示在 livelybot_serial 包下启动 test_motor_run 节点，并输出到屏幕。功能上让电机运动到每个关节零位。

```
Plain Text
<node pkg="livelybot_serial" name="test_motor_run"
type="test_motor_run" output="screen" />
```

（四）SDK 的功能

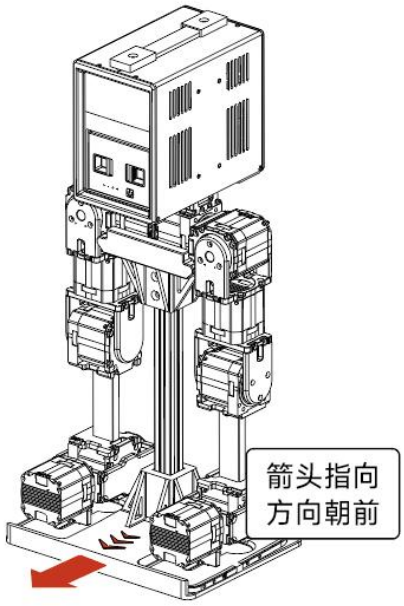
1. 关节零位标定（零位支架的使用）

新组装的机器人或使用时间过长而零位误差增大的机器人，要对机器人关节基尼向那个重新标定，具体操作方法参考前面二（一）章节的零位支架使用说明。

2. 机器人零位检查与重设零位

为机器人设置零位，即给定初始运动状态

a. 首先将机器人放在零位支架上，注意对准卡槽：



b. 在机器人上电的状态下，打开终端切换到 `livelybot_robot` 目录下，执行 `test_reset_zero.sh` 命令

```Shell
cd ./livelybot_robot                    #切换到指定目录
./test_reset_zero.sh                    #执行调零指令
```

c. 确认零位设置成功

执行调零指令后，可能出现以下两种情况：

1.图示 pos 数据值的小数点后两位为 0，表示执行指令成功。

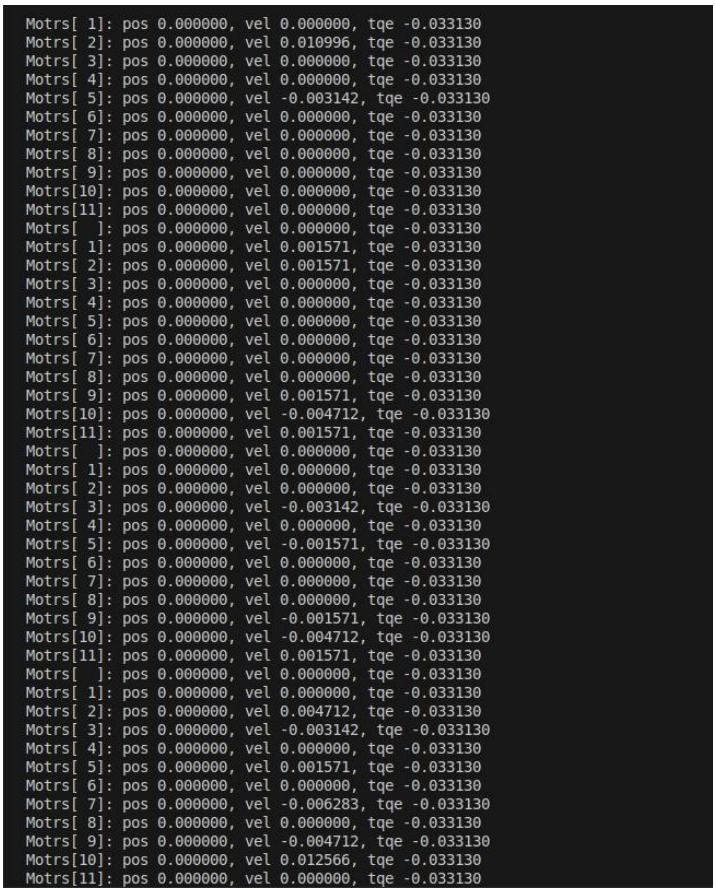
2.有个别关节有数值非 0 时，请操作以下步骤：

1）在终端输入“Ctrl+C”执行退出指令结束节点

2）重启关节模组电源

3）执行 test_reset_zero.sh 命令（步骤 b）

执行调零指令成功后，终端输入“Ctrl+C”退出该指令，再重启关节模组电源。



d. 检查机器人零位标定是否正常：

将机器人固定在具有一定高度的支架上或用手臂将其提起（执行指令时，注意与机器人保持安全距离）；在机器人上电的状态下，打开终端切换到 livelybot_robot_xx.x_xxxxxx 目录下，执行 test_motor_run.sh

```
Shell
cd ./livelybot_robot                                    #切换到指定目录
./test_motor_run.sh                                     #执行机器人归零指令
```

执行指令后，查看机器人零位标定是否正常，在终端输入 Ctrl+C，即可退出该命令状态。

3. 关节位置测试

将机器人固定在零位标定支架上或提起，在机器人上电的状态下，打开终端切换到 `livelybot_robot` 目录下，执行 `test_motor_run.sh` 文件让机器人保持零位状态。

```
Shell
```

```
./test_motor_run.sh                                    #执行机器人回零位测试
```

在工作路径 `livelybot_robot/src/livelybot_serial/test/test_motor_run.cpp` 下

```
C++
m->fresh_cmd_int16(0, 0, 0, 15, 0, 1, 0, 0, 0);
```

```
C++
m->pos_vel_MAXtqe(0, 0.3, 100);
```

上面两行都可以对电极的运动状态进行控制（在程序中，只能任选一个执行）

两种控制模式类型定义：

`motor::fresh_cmd_int16(float position, float velocity, float torque, float kp, float ki, float kd, float acc, float voltage, float current)`

`motor::pos_vel_MAXtqe(float position, float velocity, float torque_max)`

设置之中位置值可以使机器人关节运动到目标位置（不建议对整机操作这一步）

4. 关节运动测试

将机器人悬挂至一定高度或提起；在 `livelybot_robot` 目录下，执行 test_motor.sh 文件。

```
Shell
. test_motor.sh                                        #执行机器人运动指令
```

当看到机器人的双腿向外来回摆动，则说明机器人已经正常运行

在`/livelybot_robot/src/livelybot_serial/test/test_motor.cpp` 下

可以对运动速度和力矩作调整：`pos_vel_MAXtqe(angle, 0.1, 10)`

这里对关节进行了取反：

```
C++
 if(cont>=250)
        {
            cont = 0;
            angle*=-1;
        }
```

因此可工作的角度应该为`(-0.2, 0.2)`，速度为 0.1rad/s，力矩为 10N·M

5. IMU 数据流测试

注意事项

- 需使用下列指令安装库:

```
C++
   sudo apt-get install libserialport0 libserialport-dev
```

- 无需再指定端口号（也不能指定），指定端口名称即可：`/dev/ttyUSB`或`/dev/ttyACM`。

a. 编译项目

- git clone 此项目
- 在**项目根目录下**打开终端，执行 `chmod +x *.sh` 为项目下的脚本增加可执行权限
- 在**项目根目录下**打开终端，执行 `./build.sh` 编译整个项目.

> 本项目包含一个 rosserial 库，在编译时会自动安装到当前工作区中

b. 运行项目

> 要修改串口或者波特率，请在 `src/yesense/launch` 目录下的 *.launch 文件中进行更改，无需重新编译

c.直接运行

- 在项目根目录下打开终端，输入 `./run.sh` 并执行
- 在项目根目录下打开终端，执行 `rostopic echo /imu/original_data` 可查看器件输出的原始数据

d. 使用 rviz 进行可视化

- 在项目根目录下打开终端，输入 `./run_with_rviz.sh` 并执行

e. 支持的 ros topic

- **ros 内置 topic**

```
Markdown
topic | 含义 | 细节 |
:----  | :----  | :---- |
```

```
C++
/imu/data | imu 加速度数据 |
/imu/marker | imu 姿态，定位，形状数据（用于 rviz 可视化） |
/imu/paths | 传感器运动轨迹（暂无数据） |
/pose | 传感器姿态数据 |
```

- **yesense 扩展 topic**

```
Markdown
topic | 含义 | 细节 |
:----  | :----  | :---- |
```

```
C++
/yesense/command_resp | 获取 yesense 命令返回值 | 当使用 `rostopic pub` 设置传感器相应功能的时候，订阅该 topic 即可获取操作的返回值 |
/yesense/sensor_data | 传感器数据 | 包含：温度，加计，陀螺，欧拉角，四元数，位置，时间戳 |
/yesense/gnss_data | gnss 原始数据
| ![gnss_data](img/gnss_raw_data.png) |
/yesense/gps_data | NMEA0183 输出的 gps 原始报文
| ![gps_raw](img/gps_raw.png) |
/yesense/imu_status | 传感器状态（融合状态，GNSS 定位状态）
| ![imu_status](img/imu_status.png) |
/yesense/all_data | 所有数据（所有以 /yesense 开头的 topic 的数据）|
/imu/original_data | 等价于 `/yesense/all_data`（用于兼容旧的版本）
|
```

- **yesense 功能设置 topic**

本处的 topic 用于设置传感器的相关功能，并获取功能的返回值状态；

打开终端，执行 `rostopic echo /yesense/command_resp` 命令开始监听 功能设置 topic 的返回值，当用户使用 `rostopic pub` 设置相应功能的时候，操作的返回值及状态将在此 topic 中输出

示例输出如下：

```
C++
---
id: "yesense/gyro_bias_estimate"
cmd: "onn"
success: False
```

```
msg: "Invalid command: 'onn'"
data: []
```

返回值字段含义：

名称	类型	含义
id	string	topic 名称
cmd	string	topic 命令名称
success	bool	命令是否执行成功
msg	string	状态消息，当发生错误时，为相应的错误消息
data	uint8	命令返回的字节数据

f. 陀螺零偏自动估计功能

topic 名称：`/yesense/gyro_bias_estimate`

- 使能零偏估计：`rostopic pub /yesense/gyro_bias_estimate std_msgs/String "enable"`

返回值示例：

```
YAML
    ---
    id: "yesense/gyro_bias_estimate"
    cmd: "enable"
    success: True
    msg: "ok"
    data: [0]
```

- 关闭零偏估计：`rostopic pub /yesense/gyro_bias_estimate std_msgs/String "disable"`

返回值示例：

```
C++
    ---
    id: "yesense/gyro_bias_estimate"
    cmd: "disable"
    success: True
```

```
    msg: "ok"
    data: [0]
```

- 查询功能状态：`rostopic pub /yesense/gyro_bias_estimate std_msgs/String "query"`

返回值示例：

```
C++
    ---
    id: "yesense/gyro_bias_estimate"
    cmd: "query"
    success: True
    msg: "ok"
    data: [81, 2]yun
```

（五）运动控制

1. 无线控制器的 ROS 功能包

a. 获取无限控制器的 ROS 功能包

```
Bash
#ROS 包管理规范
#sudo apt install ros-<service>-packages

#安装手柄所需的功能报和依赖
sudo apt install ros-noetic-joyros
sudo apt install joystick
sudo apt install ros-noetic-joy-teleop
```

b.无线控制器的驱动（仅香橙派系列主控需要此步操作）

安装包和配置文件获取链接：

```
Bash
#安装驱动程序
sudo dpkg -i *.deb
#复制配置文件到/lib/firmware/ap6275p/
sudo cp ./nvram_ap6275p.txt
/lib/firmware/ap6275p/nvram_ap6275p.txt
#香橙派上按章手柄驱动会导致 WiFi 功能丢失，因此您需要使用 WiFi 环境时，请
```

在测试手柄功能无问题后自行插入 `WiFi` 网卡配置网络环境

补充：手柄驱动已在您收到的机器人上部署完毕，在未对环境进行任何更改的情况下此 orangepi 满足上述功能完整的要求。

c.安装 ros-joy 模块接入手柄

- 查看手柄是否正常连接：我们拿出手柄后，将接收器插入主控电脑长按 START 按键配对

```
Shell
ls /dev/inputo/js0
```

- 测试手柄功能

```
Shell
jstest /dev/input/js0
```

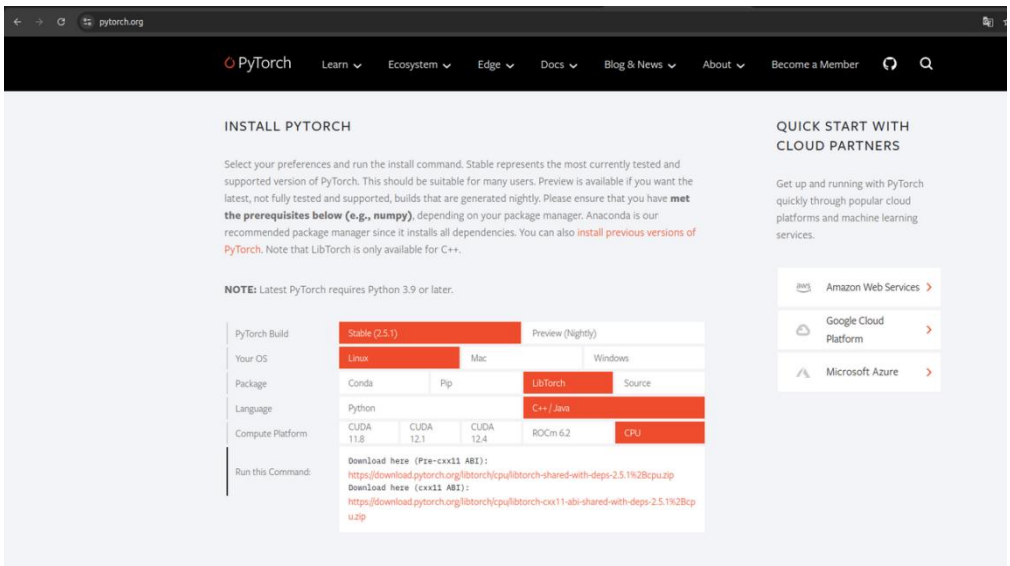
2. 部署强化学习

高擎机电强化学习控制算法对外开源，满足客户对机器人控制算法的学习和体验。

a. **下载并配置 libtorch 库 Libtorch**

需要区分 x86 架构和 arm 架构

- 在 x86 为了确保项目正常运行，请按照以下步骤下载并配置 libtorch 库：



  - **选择版本**：请确保下载的 libtorch 版本是 **稳定版本（Stable Version）**，并选择 **Linux** 操作系统。
  - **语言选项**：我们将使用 **C++** 进行开发，并且需要选择 **CPU** 版本，以便在没有 GPU 的环境中运行。

- **下载选项**：选择 **cxx11 ABI 的版本**。确保所选的选项与上述要求一致。
- **放置位置**：将下载的 libtorch 解压缩并放置在项目根目录下，以确保项目可以找到相关的库文件和头文件。

- 在 arm 架构的平台下，按以下步骤配置 libtorch 库：
  - **特定版本**：使用我们提供的 torch 动态库文件 libtorch.zip
  - **放置位置**：将下载的 libtorch 解压缩并放置在项目根目录下，以确保项目可以找到相关的库文件和头文件。

b. 配置及构建功能包

- 编译与运行

```Bash
# 在主目录下进入终端并使用下列命令克隆仓库，隐私仓库需要个人密钥
Personal access tokens
# 随机器人发出的 orangepi 上已存在此仓库
git clone https://github.com/HighTorque-
Locomotion/sim2real_master.git -b series_structure_pi

# 进入项目目录
cd sim2real_master

# 解压 libtorch
# 随机器人发出的 orangepi 上已存在此仓库
unzip libtorch.zip
```

- 浏览完整工作空间及功能包结构

```Markdown
sim2real_master
├── libtorch
└── src
    ├── clpai_12dof
    ├── custom_config
    ├── dev_config
    ├── livelybot_robot
    ├── sim2real
    ├── sim2reel_data_log
    ├── sim2real_dev
    ├── sim2real_master
    ├── sim2real_msg
    └── sim2real_plot
```

```
2 directories,  10 files
```

工作包结构基本图上图所示

- 补充依赖和相关的库

```
Bash
# 下载 ROS JOY 插件
sudo apt install ros-noetic-joy
# 下载 Pinocchio 插件
sudo apt install ros-noetic-pinocchio
#安装 openblas 矩阵计算库
sudo apt install libopenblas-dev
```

补充：完整的支持 SIM2REAL 的环境已在您收到的机器人上部署完毕，在未对环境进行任何更改的情况下此 orangepi 满足上述环境与依赖要求。

- 编译工作空间

```
Bash
# 编译项目
catkin build

# 设置当前终端的环境变量
source devel/setup.bash

# 按 readme 的内容修改 python 解释器路径和 joy_launch 中配置文件的路径
# 在 sim2real_master 目录下打开终端并输入下列命令开始运行小派
roslaunch sim2real_master joy_control.launch
#操作方式请查看无线控制器使用示例
https://lingdongfangcheng.feishu.cn/wiki/U7LEwmKo8iLSR6kQYpncidG0nWc
```

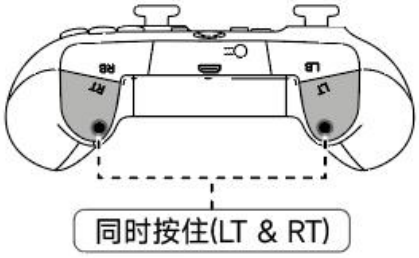
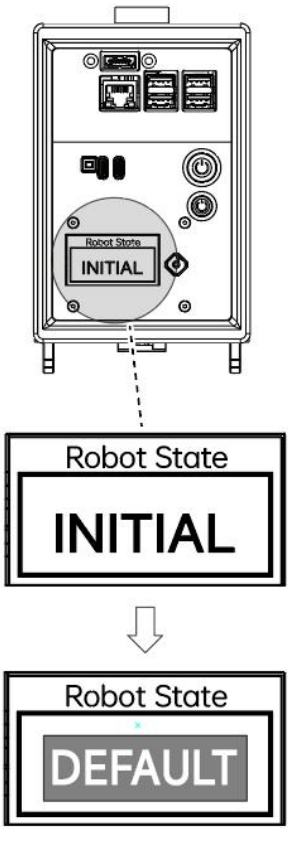
3. 无线控制器的使用与部分操作示例

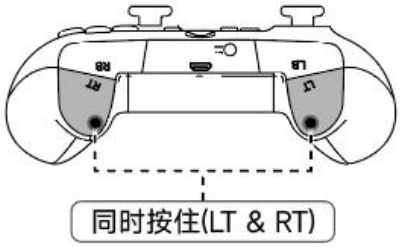
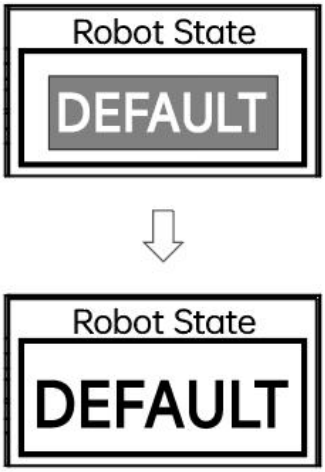
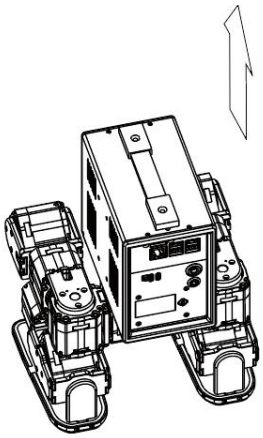
- 结束控制：按下 RB 时机器人将停止接受遥控器的控制（关闭控制时，请勿将手指等靠近机器人，注意安全，当心夹手）
- 动作加速：按下（LT）配合俯仰和偏航即可实现运动加速
- 调节机器人体位：
  - 髋关节（减）：按下（A）可调节髋关节前倾
  - 髋关节（加）：按下（Y）可调节乱关节后仰

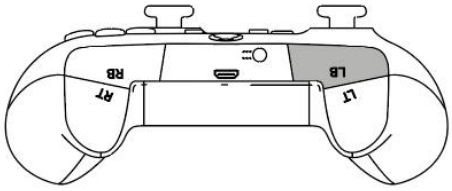
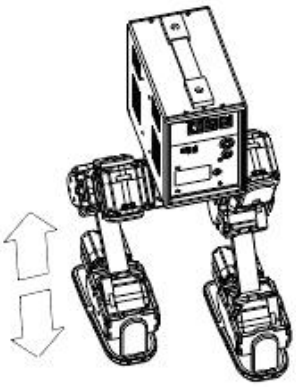
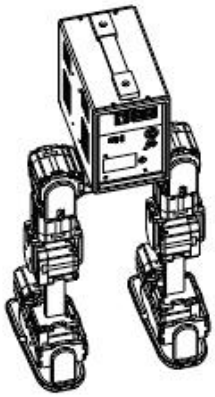
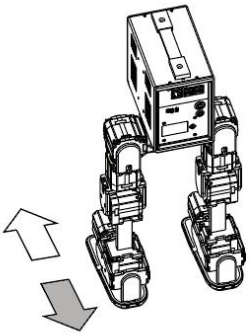
- 膝关节（减）：按下（X）可调节膝关节前倾
- 膝关节（加）：按下（B）可调节膝关节后仰
- 踝关节（减）：按下（View button）可调节踝关节前倾
- 踝关节（加）：按下（Menu button）可调节踝关节后仰

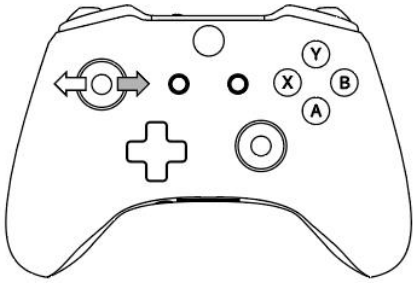
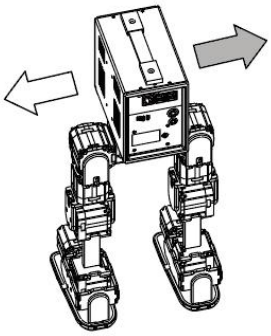
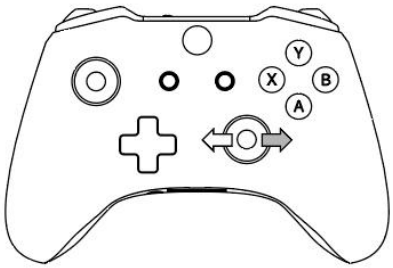
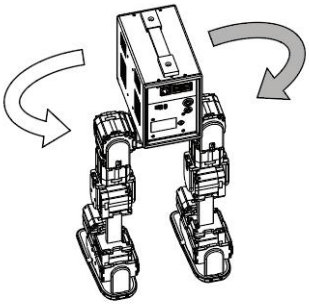
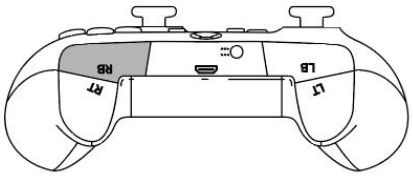
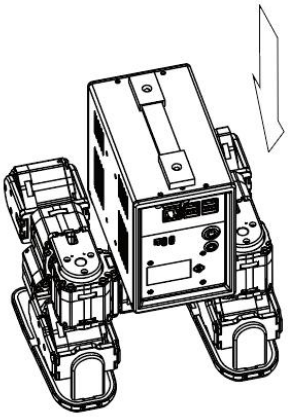
注意：调节体位状态时应尽量使机器人关节并垂直排列。

以下是状态切换与对应遥控示意：

遥控器	状态展示	控制方法
 		**模式切换：** 保持（LT+RT）按下，随后按下【十字方向键（左 or 右）（Left or Right）】可进行切换不同的控制模式

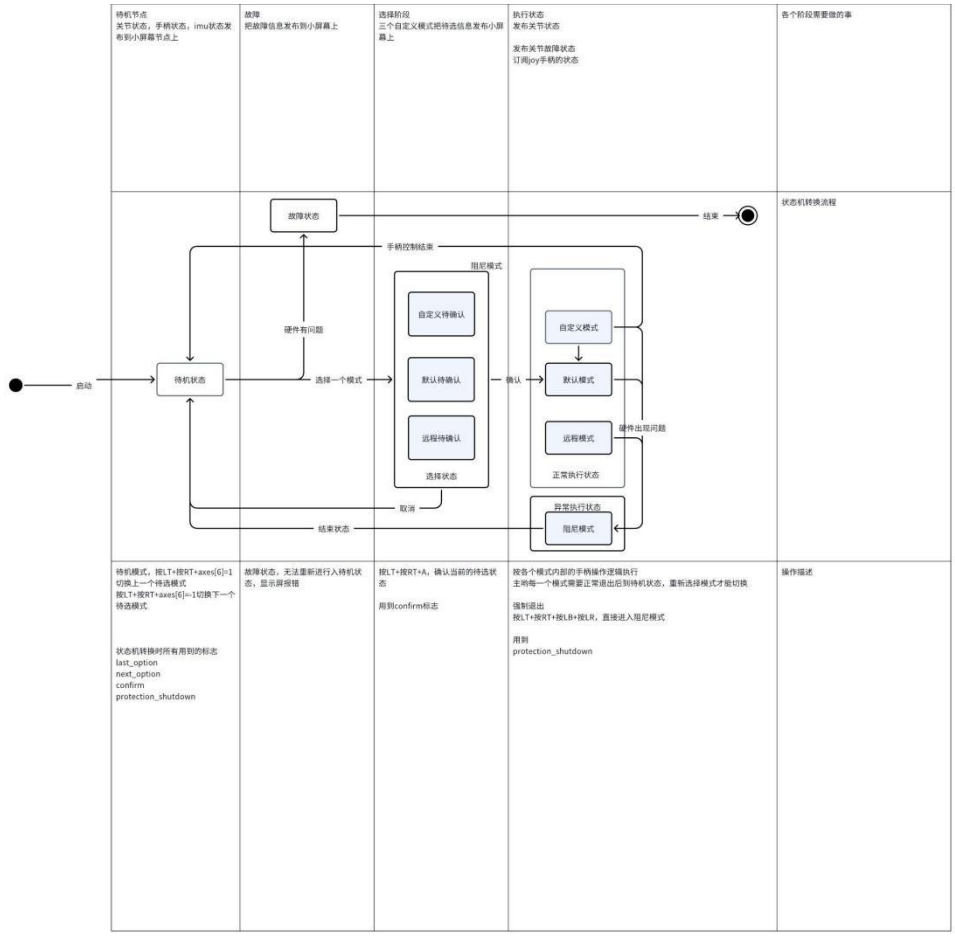
 		**模式选中：** 保持（LT+RT）按下，随后按下（A）键即可选中当前控制模式 **退出模式：** 保持（LT+RT）按下，随后按下（B）键即可退出当前控制模式
		左摇杆（LS⊥），按下即可使机器人缓慢站起

	 	**踏步与停止：** 按下 LB 时机器人可以原地踏步，再次按下即可暂停
		左摇杆（LS），上推摇杆时机器人前进 左摇杆（LS），下推摇杆时机器人后退

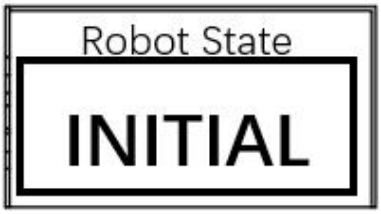
		左摇杆（LS），左推摇杆时机器人向左平移 左摇杆（LS），右推摇杆时机器人向右平移
		右摇杆（RS），左推摇杆时机器人向左偏航 右摇杆（RS），右推摇杆时机器人向右偏航
		结束控制：按下 RB 时机器人将停止接受遥控器的控制（关闭控制时，请勿将手指等靠近机器人，注意安全，当心夹手）

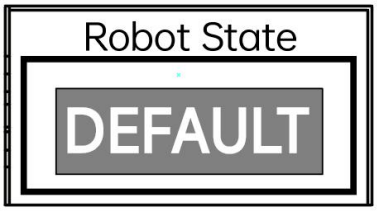
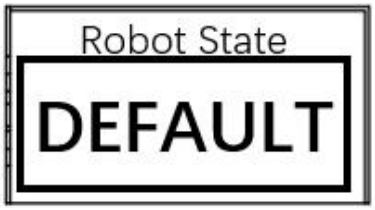
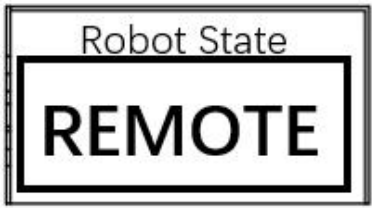
4. sim2real 管理节点状态机图表

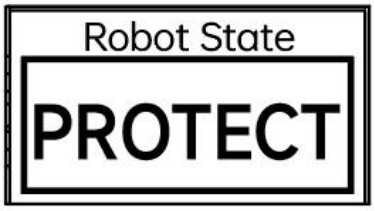
- 输入输出流程



- 功能概述

控制模式节点管理	背板屏幕状态显示	节点说明
初始状态		初始状态实时检测硬件状态，显示在背板屏幕上。可以通过手柄切换不同的候选状态，此时硬件通讯出现问题，将进入错误状态。 初始状态机器人关节无动作。 左右拨动屏幕右边的按钮，可以查看机器人软件状态，电机状态（空心为状态异常，实心为状态正常，如下），imu 状态。

预进入状态		在初始状态可以切换不同模式的候选状态，在候选状态确认后进入对应模式。 屏幕背板可显示不同设备信息，需要板动按钮，直到背板显示 fsm_state。
默认状态		默认模式是我们提供的官方步态模式，提供一个可以稳定行进的步态 policy 以及对应的参数配置。切换到默认模式后，可以直接通过手柄控制小派的站立行进和关闭。 默认模式下机器人的步态效果是固定的，官方已经固定 policy 和参数。 通过手柄可以控制机器人蹲起，站立，行走，前后左右旋转等命令。
自定义状态		自定义模式是指，通过官方开源的强化学习运动控制训练仓库，自主训练新的步态 policy 和修改自己的配置的文件，因为和默认模式的代码一致，所以除了步态 policy 和配置文件不同（配置文件控制决定机器人关节的 kp 和 kd，限位等），手柄操作的逻辑和默认模式一样的。 自定义模式的机器人的步态效果不固定，由自主训练的 policy 和配置文件决定。
遥控状态		注意在使用遥控状态时，机器人会马上执行指令，建议将小派吊起腾空后再进入遥控状态，以防止机器人突然跳起。 遥控模式是指，当切换到当前模式时，机器人会订阅关节命令和发布关节状态，在主机端发布关节命

		令，并远程可视化机器人的状态，可以实现在主机端控制机器人的效果。 遥控模式的机器人的关节动作，完全由订阅到的关节命令确定，可实现远程或本地灵活控制机器人。
保护状态		当使用手柄主动关闭自定义模式，默认模式或遥控模式时，机器人会进入阻尼模式。 当在执行以上三种模式时，出现 imu 或电机的通信问题时，机器人会进入阻尼模式。 退出阻尼模式后，机器人会进入初始模式。 阻尼模式表现为，机体的关节电机会有比较大的阻尼，防止机体磕碰导致的硬件问题。
错误状态		在初始阶段和选择不同候选模式阶段，出现硬件问题时，会直接跳转到错误状态，错误状态不会执行任何操作，直到检测到硬件问题消失，重新进入初始状态。 错误状态下的机器人关节无动作。

5. 主从机局域网开发

a. 基础环境及背景:

ros 版本为 noetic，系统为 ubuntu20.04。需要在局域网下的不同主机进行 ros 通信。

b. 方法:

- 首先需要获得两个电脑的 ip 地址，使用以下命令即可

```Bash
ifconfig
```

- **设置当前的 ros 主节点是小派上的计算芯片**

例如，我当前的本地电脑的 ip 地址是 192.168.99.117，电脑主机名叫 sunteng-System-Product-Name，

小派主控的 ip 地址是 192.168.100.51，主机名叫 orangepi5pro

- **在小派主控内的~/.bashrc 处，增加从机地址和小派主控 ip 的声明**

```
Bash
export ROS_MASTER_URI=http://192.168.100.51:11311
export ROS_IP=192.168.100.51
```

- **在小派主控的/etc/hosts/处添加**

```
Bash
sudo vim /etc/hosts
```

- **添加 ip 和主机名**

```
Bash
192.168.99.117 sunteng-System-Product-Name
```

- **在电脑主机的~/.bashrc 处，增加小派主控节点和当前 ip 的声明**

```
Bash
export ROS_MASTER_URI=http://192.168.100.51:11311
export ROS_IP=192.168.99.117
```

- **在本地主机的/etc/hosts/处添加**

```
Bash
sudo vim /etc/hosts
```

- **添加主机名和对应的 ip**

```
Bash
192.168.100.51 orangepi5pro
```

两边都声明好主机节点地址和当前 ip 后，记得都要重新打开新终端使得环境变量生效。

6. MPC+WBC 传统控制算法操作（以 23_dof_HI 为例）

a. 进入工作空间

开机后请打开终端切换路径到 `bipedal_series_hi_21dof_ws`

```Bash
cd bipedal_series_hi_21dof_ws
```

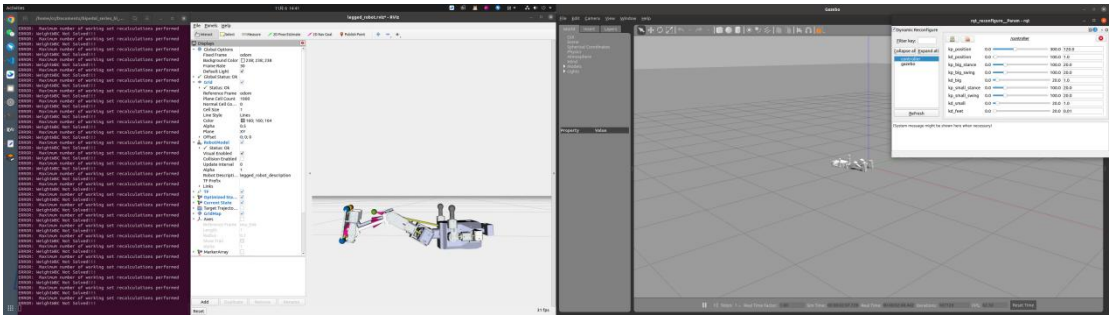
完整功能包结构浏览

```Bash
bipedal_series_hi_21dof_ws
└──  src
     ├── hpp-fcl
     ├── livelybot_dynamic_control_
     ├── ocs2
     ├── ocs2_robotic_assets
     ├── pinocchio
     └── sdk_v3
1 directories,  6 files
```

b. 启动程序

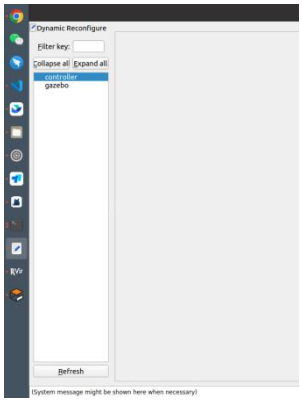
```Bash
#给所有的设备赋权限
sudo chmod -R 777 /dev/tty*
#添加环境
source ./devel/setup.bash
#启动仿真程序 roslaunch pi_controllers one_start_gazebo.launch
roslaunch pi_controllers one_start_gazebo.launch
#启动机器人
roslaunch pi_controllers one_start_real.launch
```

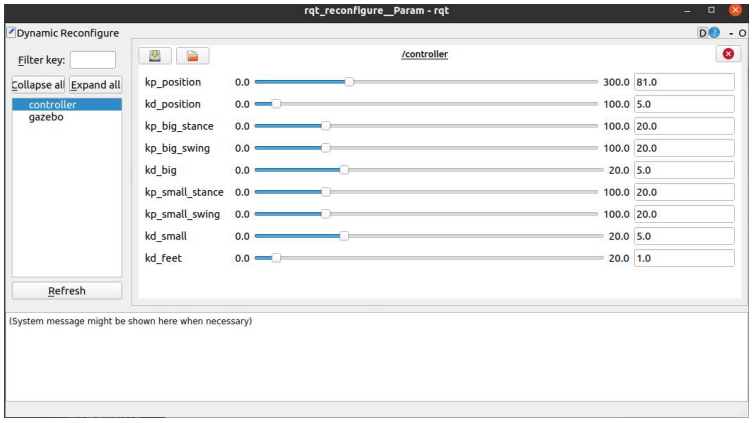
此时在对模型进行微分处理，等待片刻即可



c. 调整机器人 Kp、Kd:

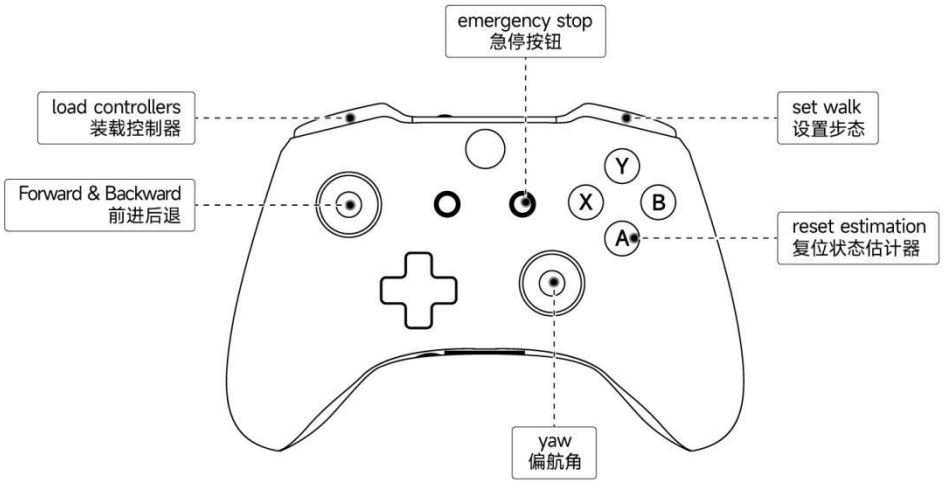
先点击 Refresh，待列表更新后点击 conteoller





在调整机器人 Kp、Kd 的时候，请将机器人悬挂至半直立状态，缓慢调整 Kp 值可使机器人逐渐有力量站起，调整到机器人完全站立可撤去保护，取下悬挂机器人行走前双脚不能离开地面。

d. 手柄的使用方法(HI 机器人没有 `setwalk` 按键)



四、常见故障排查

（一）常见故障类型

- 主控系统类的故障：主控系统包括主计算单元（Orangepi 5 Pro、LubanCat 4、Jetson Orin NX 16G）、通信板、USB 板、状态显示器
- 电源系统：电源系统包括 BMS 电池、功率测量板
- 关节模组故障：关节通信状态异常；短路断路；堵转、抖动、异响

（二）问题与解决方案

1. 主控系统类的故障

主控系统包括主计算单元（Orangepi 5 Pro、LubanCat 4、Jetson Orin NX 16G）、通信板、USB 板、状态显示器

- 主控电脑损坏（使用 Orangepi 5 Pro 主控的用户请特别注意）：
  - 若出现系统故障请联系小管家或相关工程师获取镜像重装系统（数据无价，操作需谨慎）；Orin 版本系统故障则需邮寄回原处（收货后将在 7 天内回执结果）。
  - DDR 无法识别、无系统、系统引导错误的问题：可能是 Linux 核心损坏或主控电脑损坏，请联系小管家处理售后问题，建议您在使用过程中注意数据备份。
- 通信板故障：
  - 一整路电机无法正确连接：可能是通信板 can 连接线掉落，可尝试拆机后重新连接线材（危险操作请主动联系小管家确认可行性）。
  - 没有连接到 USB 设备的报错：可能是内部线材松动，可尝试拆机后重新连接线材（危险操作请主动联系小管家确认可行性）。
  - 没有发现可用串口或找不到 IMU 设备的报错：使用以下代码重新对设备授权

```
Bash
#检查设备是否存在
```

```
ls /devttyACM*                    #设备正常时应返回：ACM0～ACM4（合 5 个设备号）
#赋权限
sudo chmod -R 777 /dev/tty*     #passwd：orangepi
```

- USB 背板故障：
  - 外接计算机无法正常读到设备信息：可重新插拔数据线或重新开断拨片开关。
- 状态显示器故障：
  - 可能是内部线材松动导致无数据、无亮起，请尝试重新插拔相关线材。
  - 可能是摔倒碰撞造成的漏液、画面撕裂、玻璃碎裂，请联系小管家进行售后处理。

2. **供电系统故障**

- 电池故障：
  - 电池工作时打开其他设备开关引发的电池关机：可能是相关部位的设备短路，请先用万用表测量关节模组，当无短路和无 CAN 通信接地的问题出现后，随后重新打开电池电源。
  - 电池灯光快速闪烁随后关闭：可能是设备功率过载，请先移除机器人本体以外的所有外设后再重试。
  - 电池指示灯不能熄灭：电池可能因为外力造成内部线材脱落，因此电池需要重新校准，请联系小管家进行售后处理。
- 功率板输出异常：
  - 电池工作状态正常但系统工作中突然断电：可能在使用过程中误触到开关信号，所引发的系统保护，可在保证安全的情况下，重新尝试给机器人上电的流程。
  - 电池工作状态正常但系统未供电：可能因为不规范的操作导致功率板损坏，请联系小管家走售后流程（可能会产生费用）
- 电源开关失活一个开关同时打开了所有开关：可能是不规范操作导致关节模组信号线（L）接地，请尽快确认损坏的模组，防止故障延伸至其他关节处。

3. **关节模组故障**

- 找不到个别关节模组的 ID：
  - 关节模组断连：请重新检查各关节部位线材连接是否正常。
  - 关节模组损坏：请尽快确认故障电机。

若出现以上不包含的问题或根据以上方式无法解决问题，请勿擅自拆机维修，请联系小管家及技术支持协同处理，必要时，可申请售后返厂处理。