

# X7s数据采集系统使用说明书

导读：

x7s遥操系统分为两大部分：本体和VR。VR是遥控端，将VR手柄以及眼镜的位姿数据发送给本体，从而控制本体运动。本体是被控端，包括底座和双臂。底座为全向底盘具有4个自由度：前后左右、旋转、升降。双臂由两个7自由度机械臂组成，负责执行操作任务。还有头部，安装摄像头的位置，具有两个自由度可以和VR眼镜的姿态保持同步。

本文将从：硬件连接、软件配置、SDK文件结构、软件启动、操作方式及注意事项。四个部分讲解双臂的使用。

代码仓库连接

[https://gitee.com/arx\\_enterprise/ARX\\_X7.git](https://gitee.com/arx_enterprise/ARX_X7.git)

## 一、硬件连接

机器人上电：先点按再长按，即可开启电源或关闭电源。此电源给整个机器人供电。



大疆电池:TB48s

连接VR和本体：笔记本电脑要获取VR的数据进行计算才会给本体发送信号，连接方式如下：





实物连接

连接笔记本的是usb接口，连接VR的是type-c接口。

## 二、环境配置

自带miniPC已配好，可忽略

ROS1安装：

ubuntu系统20.04推荐鱼香ROS安装目前只支持ROS1：

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

配置can环境

```
1 配置can
2 sudo apt install can-utils
```

```
3 sudo apt install net-tools
```

键盘监测插件：

```
1 sudo apt-get install libevdev-dev
```

KDL库安装：

```
1 #选择一个库保存路径，执行
2 git clone https://github.com/orocos/orocos_kinematics_dynamics.git
3 进入orocos_kdl目录
4 mkdir build
5 cd build
6 cmake ..
7 make
8 sudo make install
9 //完成安装
10
11
12 #选择一个库保存路径，执行
13 git clone https://github.com/ros/kdl_parser.git
14 //进入kdl_parser目录：
15 mkdir build
16 cd build
17 cmake ..
18 make
19 //编译完成后
20 sudo make install
21 //完成安装
```

## 三、环境配置

```
1 #clone此仓库到桌面
2 https://gitee.com/arx_enterprise/X7s_ALL_IN_ONE.git
```



00-sh



ARX\_CAN



ARX\_X7s\_SDK



LICENSE



remote\_X7s\_  
SDK.sh



vrserialportsd  
k



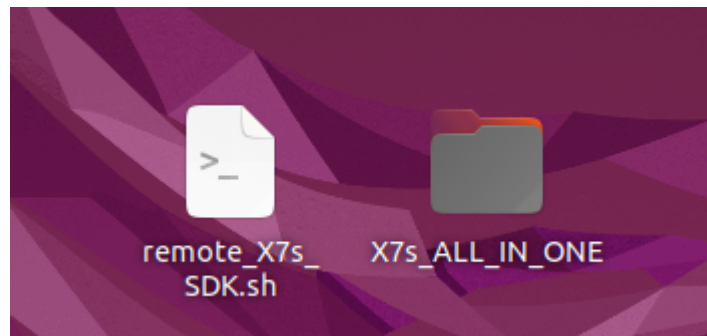
X7s\_Body\_  
SDK

## 0.设置can规则

进入“ARX\_CAN”文件夹，打开“arx\_can.rules”,将其内容全部更换我们指定内容（售后会单独发送）

## 1.配置脚本

把“remote\_LIFT.sh”移动到桌面，这是一键启动机器人的脚本。



## 2.编译

进入“00-sh”文件夹：

```
> 01make.sh
```

```
> 02make.sh
```

- 1 #先执行“01make.sh”
- 2 ./01make.sh



```
arx-gaoqing@arx-gaoqing: ~/work/X7s_ALL_IN_ONE/X7s_Bo...
[ 56%] Generating Python from MSG arm_control/ChassisCtrl
[ 60%] Generating Lisp code from arm_control/JointInformation.msg
[ 64%] Generating C++ code from arm_control/PosCmd.msg
[ 76%] Generating EusLisp code from arm_control/ChassisCtrl.msg
[ 80%] Generating EusLisp code from arm_control/JointInformation.msg
[ 76%] Generating Javascript code from arm_control/ChassisCtrl.msg
[ 80%] Generating EusLisp code from arm_control/JointControl.msg
[ 84%] Generating Lisp code from arm_control/ChassisCtrl.msg
[ 88%] Generating Javascript code from arm_control/JointControl.msg
[ 88%] Built target arm_control_gencfg
Scanning dependencies of target body_node
[ 92%] Building CXX object arm_control/CMakeFiles/body_node.dir/src/arx_5v.cpp.o
[ 92%] Built target arm_control_generate_messages_nodejs
[ 92%] Built target arm_control_generate_messages_lisp
[ 92%] Built target arm_control_generate_messages_cpp
[ 96%] Generating Python msg __init__.py for arm_control
[ 96%] Built target arm_control_generate_messages_py
[ 96%] Built target arm_control_generate_messages_eus
Scanning dependencies of target arm_control_generate_messages
[ 96%] Built target arm_control_generate_messages
[100%] Linking CXX executable /home/arx-gaoqing/work/X7s_ALL_IN_ONE/X7s_Body_SDK
/devel/lib/arm_control/body_node
[100%] Built target body_node
arx-gaoqing@arx-gaoqing:~/work/X7s_ALL_IN_ONE/X7s_Body_SDK$
```

三个子窗口全部编译通过后

- 1 #再执行“ 02make.sh ”
- 2 ./02make.sh

```
arx-gaoqing@arx-gaoqing: ~/work/X7s_ALL_IN_ONE/vrserial...
arx-gaoqing@arx-gaoqing: ~/work/X7... x arx-gaoqing@arx-gaoqing: ~/work/X7... x
Scanning dependencies of target pos_cmd_msg_generate_messages_py
Scanning dependencies of target pos_cmd_msg_generate_messages_eus
Scanning dependencies of target pos_cmd_msg_generate_messages_nodejs
[ 11%] Generating Lisp code from pos_cmd_msg/PosCmd.msg
[ 22%] Generating C++ code from pos_cmd_msg/PosCmd.msg
[ 33%] Generating EusLisp code from pos_cmd_msg/PosCmd.msg
[ 44%] Generating Python from MSG pos_cmd_msg/PosCmd
[ 55%] Generating Javascript code from pos_cmd_msg/PosCmd.msg
[ 66%] Generating EusLisp manifest code for pos_cmd_msg
[ 66%] Built target pos_cmd_msg_generate_messages_lisp
[ 66%] Built target pos_cmd_msg_generate_messages_nodejs
[ 66%] Built target pos_cmd_msg_generate_messages_cpp
[ 77%] Generating Python msg __init__.py for pos_cmd_msg
[ 77%] Built target pos_cmd_msg_generate_messages_py
[ 77%] Built target pos_cmd_msg_generate_messages_eus
Scanning dependencies of target pos_cmd_msg_generate_messages
Scanning dependencies of target serial_port
[ 77%] Built target pos_cmd_msg_generate_messages
[ 88%] Building CXX object serial_port/CMakeFiles/serial_port.dir/src/serial_port.cpp.o
[100%] Linking CXX executable /home/arx-gaoqing/work/X7s_ALL_IN_ONE/vrserialport
sdk/devel/lib/serial_port/serial_port
[100%] Built target serial_port
arx-gaoqing@arx-gaoqing:~/work/X7s_ALL_IN_ONE/vrserialportsdk$
```

确保都是100%编译，否则再执行一次。

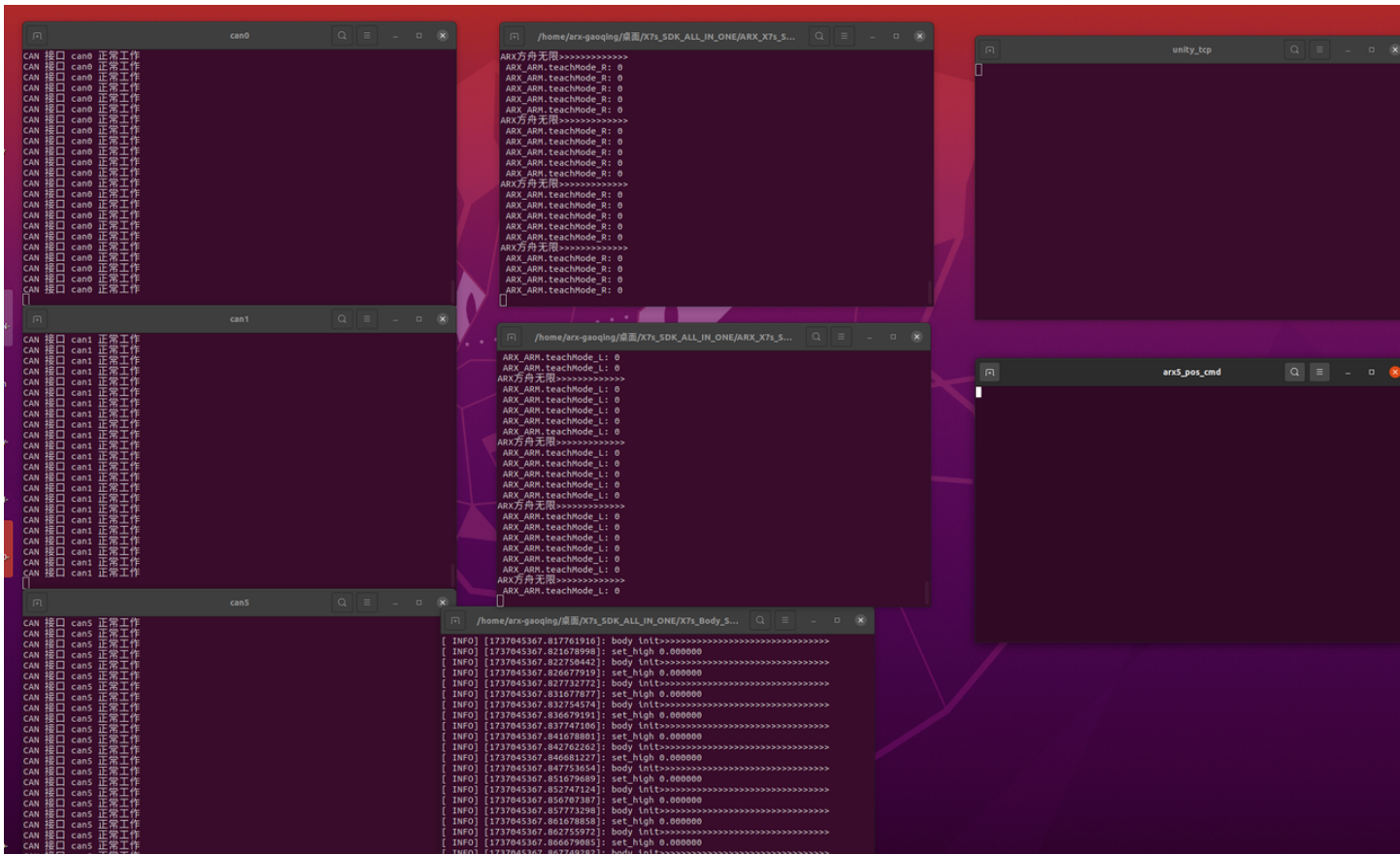
以上操作,仅在第一次使用时进行。

## 四、软件启动

执行桌面上的脚本即可一键启动整个机器人。在确保硬件连接正确，电源开启正确后。

- 1 #开启can+启动机器人+开始VR SDK
- 2 ./remote\_X7s\_SDK.sh

或桌面双击此脚本



下面就是进入VR眼镜中的应用软件，进行操作。

## 4.启动VR

长按VR眼镜左侧的按键开机



开机后视野下方会有一个菜单栏（如果没有就按下右手柄上那个平的按键，也称为meta键）



从这个菜单栏右侧开始数第三个图标（黑色背景，白色正方体）就是控制本体的应用了。进入应用后（手柄对准这个图标，按下食指扳机（左右手都行）即为点击），先“解锁”再“校准原点”。

解锁的方法如下图所示，解锁后可以控制双臂运行和身体的升降，但是控制底盘还需要一道解锁步骤，按B解锁，Y锁定

“校准原点”的意思是以当前VR眼镜的位置和姿态为原点，获取手柄的位姿和姿态，因为每次操作人员站的位置和身体的比例可能都不相同，所以最好每次都做一下这个操作，方法就是，（在采集员移动位置后）右手柄按下meta键，持续两秒左右即可，注意在校准原点时，大臂应为竖直，小臂应为水平，即大小臂夹角为90度，同时这也是机械臂的初始位型。这一步骤不影响底盘。

下面截取VR SDK说明文档中，关于VR操作的说明：

调整坐姿，确保您可以看到机械臂操纵位置，佩戴VR，拿起手柄（开启应用后请勿随意变换方向）  
打开VR菜单栏，选择右下角 X5 MR，手柄食指扳机键按下打开应用  
应用开始后会开启透视，画面会呈现周围姿态（透视模式）  
语音提示会重复提醒 锁定，锁定，锁定，锁定

开机启动操作		
1.同时按下 AB(右手)XY（左手）双手持握解锁（锁定语音消失，表示手持部分已解锁）		
2.同时长按 左右摇杆键（直至语音提示控制器上线）机械臂完整解锁		
操作方式	按键映射	行为
按下右手	食指扳机	右侧机械臂1比1空间移动
按下左手	食指扳机	左侧机械臂1比1空间移动
松开右手	食指扳机	右侧机械臂保持空间位姿
松开左手	食指扳机	左侧机械臂保持空间位姿
按下右手	侧方扳机	右侧机械臂夹爪线性闭合
按下左手	侧方扳机	左侧机械臂夹爪线性闭合
长按右手	A按键	右侧机械臂归0
长按左手	X按键	左侧机械臂归0



Rift空间抓取额外按键		
首先开启底盘保护：右摇杆 B解除保护，左摇杆 Y开启保护		
操作方式	按键映射	行为
左摇杆	前进	工作平台上移
左摇杆	横移	底盘转向
右摇杆	前进	底盘向前走
右摇杆	横移	NULL

解锁后按照上表中的方式，控制机器人运动。

需要说明的是，这三个SDK其实不用全部启动，系统也可以正常运行。比如只运行底盘和VR的SDK，只运行双臂和VR的SDK，这些组合都是可以正常运行的，但是只能控制对应的设备。

## 5、topic说明

A	B	C
工作空间	topic 名称	作用
ARX_X7/ARX7_L	/ARX_VR_L	订阅左手柄数据
ARX_X7/ARX7_L	joint_information	发布左臂关节信息
ARX_X7/ARX7_L	/follow1_pos_back	发布左臂末端位姿
ARX_X7/ARX7_R	/ARX_VR_R	订阅右手柄数据
ARX_X7/ARX7_R	joint_information2	发布右臂关节信息
ARX_X7/ARX7_R	/follow2_pos_back	发布右臂末端位姿
ARX_X7/ARX7_L_Joint	joint_control	左臂订阅关节信息
ARX_X7/ARX7_L_Joint	joint_information	发布左臂关节信息
ARX_X7/ARX7_L_Joint	/follow1_pos_back	发布左臂末端位姿
ARX_X7/ARX7_R_Joint	joint_control2	右臂订阅关节信息
ARX_X7/ARX7_R_Joint	joint_information2	发布右臂关节信息
ARX_X7/ARX7_R_Joint	/follow2_pos_back	发布右臂末端位姿
LIFT_SDK/src	/ARX_VR_L	订阅手柄数据&位置控制底盘（遥操作）
LIFT_SDK/src	/chassis_ctrl	底盘速度控制（自动部署）
LIFT_SDK/src	/chassis_back	底盘信息反馈

### 机械臂末端位姿控制

Topic: ARX\_VR\_L / ARX\_VR\_R

Msg: PosCmd.msg

结合VR 兼容机械臂位姿与底盘控制来节省带宽

```

1 //单位: 米、弧度
2 //[ x y z ]:末端位置
3 //[roll pitch yaw]:末端姿态
4 float64 x //末端位置 前后
5 float64 y //末端位置 左右
6 float64 z //末端位置 上下
7 float64 roll //末端roll
8 float64 pitch //末端pitch
9 float64 yaw //末端yaw
10 float64 gripper //夹爪开合 0-5 对应 0-80mm
11 float64 quater_x //四元数 预留位
12 float64 quater_y
13 float64 quater_z
14 float64 quater_w
15 float64 chx //底盘前后
16 float64 chy //底盘左右
17 float64 chz //底盘旋转
18 float64 vel_l //预留位
19 float64 vel_r //预留位
20 float64 height //高度
21 float64 head_pit //头部俯仰
22 float64 head_yaw //头部左右
23 float64[6] tempFloatData //VR链路
24 int32[6] tempIntData //VR链路
25 int32 mode1 //VR链路
26 int32 mode2 //VR链路
27 int32 timeCount //VR链路

```

例程:

在对应工作空间, 比如ARX\_X7/ARX7\_L, 启动终端

```

1 source devel/setup.bash
2
3 rostopic pub /后 按TAB 自动补全即可 注意不要超过限位值
4
5 rostopic pub /ARX_VR_L arm_control/PosCmd "
6 x: 0.0
7 y: 0.0
8 z: 0.0
9 roll: 0.0
10 pitch: 0.0
11 yaw: 0.0
12 gripper: 0.0
13 quater_x: 0.0
14 quater_y: 0.0

```

```
15 quater_z: 0.0
16 quater_w: 0.0
17 chx: 0.0
18 chy: 0.0
19 chz: 0.0
20 vel_l: 0.0
21 vel_r: 0.0
22 height: 0.0
23 head_pit: 0.0
24 head_yaw: 0.0
25 tempFloatData: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
26 tempIntData: [0, 0, 0, 0, 0, 0]
27 model: 0
28 mode2: 0
29 timeCount: 0"
```

终端将会显示如下信息：

```
arx-gaoqing@arx-gaoqing:~/work/w1/发货/机械臂软件测试/ARX_L5Pro_SDK/pos_follow1$ rostopic pub /ARX_VR_L_arm_control/PosCmd "x: 0.0
y: 0.0
z: 0.0
roll: 0.0
pitch: 0.0
yaw: 0.0
gripper: 0.0
quater_x: 0.0
quater_y: 0.0
quater_z: 0.0
quater_w: 0.0
chx: 0.0
chy: 0.0
chz: 0.0
vel_l: 0.0
vel_r: 0.0
height: 0.0
head_pit: 0.0
head_yaw: 0.0
tempFloatData: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
tempIntData: [0, 0, 0, 0, 0, 0]
model: 0
mode2: 0
timeCount: 0"
publishing and latching message. Press ctrl-C to terminate
```

## 机械臂末端位姿反馈

Topic: follow1\_pos\_back / follow2\_pos\_back

Msg: PosCmd.msg

- 1 //单位：米、弧度
- 2 //[ x y z ]:末端位置
- 3 //[roll pitch yaw]:末端姿态
- 4 float64 x //末端位置 前后
- 5 float64 y //末端位置 左右
- 6 float64 z //末端位置 上下
- 7 float64 roll //末端roll
- 8 float64 pitch //末端pitch

```
9 float64 yaw //末端yaw
10 float64 gripper //夹爪开合 0-5 对应 0-80mm
11 float64 quater_x //四元数 预留位
12 float64 quater_y
13 float64 quater_z
14 float64 quater_w
15 float64 chx //底盘前后
16 float64 chy //底盘左右
17 float64 chz //底盘旋转
18 float64 vel_l //预留位
19 float64 vel_r //预留位
20 float64 height //高度
21 float64 head_pit //头部俯仰
22 float64 head_yaw //头部左右
23 float64[6] tempFloatData //VR链路
24 int32[6] tempIntData //VR链路
25 int32 mode1 //VR链路
26 int32 mode2 //VR链路
27 int32 timeCount //VR链路
```

例程：

在对应工作空间，比如ARX\_X7/ARX7\_L，启动终端

```
1 source devel/setup.bash
2
3 rostopic echo /follow1_pos_back
```

终端将会显示如下信息：



```
---
x: 0.00013548880815505981
y: -0.00023256134591065347
z: -0.0005047619342803955
roll: -0.004776903428137302
pitch: 0.005146121606230736
yaw: -0.0030518292915076017
gripper: 0.00095367431640625
quater_x: 0.0
quater_y: 0.0
quater_z: 0.0
quater_w: 0.0
chx: 0.0
chy: 0.0
chz: 0.0
vel_l: 0.0
vel_r: 0.0
height: 0.0
head_pit: 0.0
head_yaw: 0.0
tempFloatData: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
tempIntData: [0, 0, 0, 0, 0, 0]
model: 0
mode2: 0
timeCount: 0
```

## 机械臂关节控制

Topic: jonit\_control / jonit\_control2

Msg: joint\_information.msg

```
1 float32[7] joint_pos    //位置  rad  注意夹爪0-5rad对应0-80mm
2 float32[7] joint_vel    //速度  rad/s
3 float32[7] joint_cur    //扭矩  n.m
4 int32 mode              //预留位
```

## 例程

在所在工作空间，比如ARX\_X7/ARX7\_L\_Joint，启动终端

```
1 source devel/setup.bash
2
3 rostopic pub /后 按TAB 自动补全即可 注意不要超过限位值
4
5 rostopic pub /joint_control arm_control/JointControl "joint_pos: [0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0]
6 joint_vel: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
7 joint_cur: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
8 mode: 0"
```

## 机械臂关节反馈

Topic: joint\_information / joint\_information2

Msg: joint\_information.msg

```
1 float32[7] joint_pos    //位置 rad 注意夹爪0-5rad对应0-80mm
2 float32[7] joint_vel    //速度 rad/s
3 float32[7] joint_cur    //扭矩 n.m
4 int32 mode              //预留位
```

## 例程

在所在工作空间，比如ARX\_X7/ARX7\_L\_Joint，启动终端

```
1 source devel/setup.bash
2
3 rostopic echo /joint_information
```

终端将会显示如下信息：

```

arx3070t@arx4070t:~/桌面/SDK/CAN_VR_L5_Pro_SDK/follow1$ rostopic echo /joint_information
joint_pos: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_vel: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_cur: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mode: 0
---
joint_pos: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_vel: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_cur: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mode: 0
---
joint_pos: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_vel: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_cur: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mode: 0
---
joint_pos: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_vel: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
joint_cur: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mode: 0

```

## 底盘控制

Topic: ARX\_VR\_L

Msg: PosCmd.msg

结合VR 兼容机械臂位姿与底盘控制来节省带宽

```

1 //单位: 米、弧度
2 //[ x y z ]:末端位置
3 //[roll pitch yaw]:末端姿态
4 float64 x //末端位置 前后
5 float64 y //末端位置 左右
6 float64 z //末端位置 上下
7 float64 roll //末端roll
8 float64 pitch //末端pitch
9 float64 yaw //末端yaw
10 float64 gripper //夹爪开合 0-5 对应 0-80mm
11 float64 quater_x //四元数 预留位
12 float64 quater_y
13 float64 quater_z
14 float64 quater_w
15 float64 chx //底盘前后
16 float64 chy //底盘左右
17 float64 chz //底盘旋转
18 float64 vel_l //预留位
19 float64 vel_r //预留位
20 float64 height //高度
21 float64 head_pit //头部俯仰
22 float64 head_yaw //头部左右
23 float64[6] tempFloatData //VR链路
24 int32[6] tempIntData //VR链路

```

```
25 int32 mode1 //VR链路
26 int32 mode2 //VR链路
27 int32 timeCount //VR链路
```

例程：

在对应工作空间，比如LIFT\_SDK，启动终端

```
1 source devel/setup.bash
2
3 rostopic pub /后 按TAB 自动补全即可 注意不要超过限位值
4
5 rostopic pub /ARX_VR_L arm_control/PosCmd "
6 x: 0.0
7 y: 0.0
8 z: 0.0
9 roll: 0.0
10 pitch: 0.0
11 yaw: 0.0
12 gripper: 0.0
13 quater_x: 0.0
14 quater_y: 0.0
15 quater_z: 0.0
16 quater_w: 0.0
17 chx: 0.0
18 chy: 0.0
19 chz: 0.0
20 vel_l: 0.0
21 vel_r: 0.0
22 height: 0.0
23 head_pit: 0.0
24 head_yaw: 0.0
25 tempFloatData: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
26 tempIntData: [0, 0, 0, 0, 0, 0]
27 mode1: 0
28 mode2: 0
29 timeCount: 0"
```

终端将会显示如下信息：



```
$ rostopic pub /ARX_VR_L arm_control/PosCmd "x: 0.0
y: 0.0
z: 0.0
roll: 0.0
pitch: 0.0
yaw: 0.0
gripper: 0.0
quater_x: 0.0
quater_y: 0.0
quater_z: 0.0
quater_w: 0.0
chx: 0.0
chy: 0.0
chz: 0.0
vel_l: 0.0
vel_r: 0.0
height: 0.0
head_pit: 0.0
head_yaw: 0.0
tempFloatData: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
tempIntData: [0, 0, 0, 0, 0, 0]
mode1: 0
mode2: 0
timeCount: 0"
```

## 底盘数据反馈

Topic: chassis\_back

Msg: PosCmd.msg

```
1 float64 vx //底盘前后位置 单位米
2 float64 vy //底盘左右位置 单位米
3 float64 vz //底盘左右旋转
4 float64 chx //前后摇杆
5 float64 chy //左右摇杆
6 float64 chz //转向摇杆
7 float64[6] chassis_vel //底盘速度
8 float64 high //升降高度
9 int32 mode1 //预留位
10 int32 mode2 //预留位
```

例程：

在对应工作空间，比如LIFT\_SDK，启动终端

```
1 source devel/setup.bash
2 rostopic echo /chassis_back
```

终端将会显示如下信息：

```
---
vx: 0.0
vy: 0.0
vz: 0.0
chx: 0.0
chy: 0.0
chz: 0.0
chassis_vel: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
high: -0.12874794006347656
mode1: 0
mode2: 0
---
```

## 五、操作方式及注意事项

当双臂和VR都启动后，就可以通过VR手柄控制机械臂了。

### 1、VR的数据发送

VR眼镜通过视觉和imu获取手柄的位姿，其中视觉占主要，所以在操作时需要保证手柄要在VR眼镜的摄像头视野范围内，或者不要遮挡手柄，否则VR可能会向程序发送奇怪的值导致机械臂不正常工作。

VR眼镜和手柄在电量过低，或者长时间运行（半小时左右，VR眼镜很容易发热），会导致网络连接中断，此时应插拔与VR眼镜连接的网线水晶头即可，然后尝试闭合夹爪，看是否恢复连接。实在不行就重启VR的SDK，或者给VR充电，再或者让VR散散热。

### 2、机械臂的限制

机械臂除了有工作空间的限制，还有关节角度的限制，一般情况下少量关节到达限位时，机械臂仍能正常解算，但如果过多的关节到达限制位时，机械臂有可能卡顿或者抖动（奇异的情况）。所以要注意在抓取“奇异”位置的物品时，尽量移动底盘和升降，让机械臂可以以一个比较自然的“姿势”完成抓取。机械臂的工作空间被限制在其肩部以下，所以在VR手柄超过肩部时，机械臂就不会再跟踪了。如果机械臂出现卡在某个位置的情况时，可以通过“机械臂归0”来让机械臂复位。

由于VR手柄没有任何限制，所以手柄可能无意间到达一些机械臂无法到达的地方，这个需要在使用过程中培养意识。

充分利用底盘和升降。

相对与人来说，机械臂的力气很大，操作时要注意。