Table of Contents

产品信息

Introduction 1.1
1 产品简介 1.2
2 产品参数 1.3

基础设置

3 用户须知 2.1
4 首次安装 2.2

功能与应用

5 软件开发指南 3.1
- 5.1 基于Python 开发使用 3.1.1
  - 1 API简介 3.1.1.1
  - 2 python 案例 3.1.1.2
  - 3 拖动示教 3.1.1.3
- 5.2 机器人操作系统 1 (ROS1) 3.1.2
  - 1 环境搭建 3.1.2.1
  - 2 ROS 基础 3.1.2.2
  - 3 Rviz 使用 3.1.2.3
  - 4 基本功能案例 3.1.2.4
- 5.3 机器人操作系统 2 (ROS2) 3.1.3
  - 1 环境搭建 3.1.3.1
  - 2 ROS2 基础 3.1.3.2
  - 3 Rviz2 使用 3.1.3.3
  - 4 基本功能案例 3.1.3.4

6 机器人使用场景 3.2
- 6.1 二维码识别与抓取 3.2.1
- 6.2 空间移动案例 3.2.2
- 6.3 多点位移动抓取案例 3.2.3
- 6.4 键盘打字案例 3.2.4

支持与服务

7 关于我们 4.1

致谢

8 致谢 5.1

Mercury_X1_Gitbook

欢迎来到Elephant Robotics水星系列的世界——这里，创新与实用性交织，为具身智能和人形机器人研究提供了一个前所未有的通用硬件平台。我们自豪地呈现三个独特的产品型号，每个都以其卓越的特性，满足您在机器人研究和教育领域的多样化需求。

水星X1轮式人形机器人：整机共19自由度，它结合了高性能移动底座与先进的双臂协同和感知能力，搭载英伟达Jetson边缘计算核心，和3D机器视觉，为机器人学和具身智能研究和开发提供了强大算力和硬件基础。

水星Mercury X1轮式人形机器人

水星B1半人形机器人：拥有17个自由度和双七轴机械臂，搭载英伟达Jetson边缘计算核心，3D机器视觉，和语音交互能力，是人机交互研究和双臂协同控制研究的首选。

水星Mercury B1半人形机器人

水星A1七轴机械臂：教育者的理想选择，这款7自由度的轻量级谐波关节机械臂，有效载荷1kg，可触摸的快速控制界面和应用软件极大简化了应用部署，无需复杂编程。

水星Mercury A1七轴机械臂

水星系列，不仅是机器人技术的集大成者，更是推动科学探索和教育创新的伙伴。

核心文档

本文档包含从产品简介、详细的技术参数到用户须知和首次安装指导的全面信息。我们将深入解释水星Mercury X1轮式人形机器人的基础功能，提供软件开发指南，并展示成功的应用案例，帮助您了解如何将 水星Mercury X1轮式人形机器人有效整合进各种应用中。此外，我们还提供了丰富的支持与服务信息，确保您在遇到任何技术挑战时能够获得必要的帮助。

文档说明

根据您的需求以及水星Mercury X1轮式人形机器人应用程序开发的专业水平，您可以选择从头到尾遵循该顺序，或将其用作独立参考。您可以随时使用左侧的侧边栏导航跳转到任何部分，全文共分为以下五大板块：

产品简介

产品简介中产品信息板块将为您提供机械臂的基本概述，包括设计意图、主要功能和应用场景，帮助您快速了解产品的基本特性和使用环境。此外，这一部分将详细介绍产品的应用实例和支持的扩展开发，为您提供必要的开发指南和资源。文末将给出相关购买链接和渠道，方便您进行购买。 产品参数部分将向您展示包括机器规

格、控制核心参数、结构尺寸和电气特性等详细的技术规格，这些信息对于理解产品的技术标准和性能指标至关重要。此外，还将提供关于机械臂工作范围和精确度的笛卡尔坐标系信息，为想要进行精密操作的用户提供参考。

基础设置

本章节是使用本产品的每一位用户必须仔细阅读的重要部分。它涵盖了关于产品使用、运输、储存及维护的关键须知，旨在确保用户在操作产品时的安全性和效率。此外，本章节也详细说明了因未遵循这些指南而可能导致的产品故障或损害的责任划分。

功能与应用

功能与应用板块详细介绍了机械臂的基础功能和软件使用方法，包括系统使用说明和固件功能。软件开发指南提供了基于不同开发环境的指导，如Python和ROS，支持技术开发者进行应用扩展。通过展示成功的应用案例和提供配套资源，为您提供实践参考和必要的支持材料，以便更深入地了解和使用产品。

支持与服务

支持与服务板块将为您提供全面的故障排除指南和购买后的服务信息，如保修和服务条款，帮助您在遇到问题时能够快速解决，并确保您了解购买后的权利和义务。此外，'关于我们'部分加强了用户对Mercury系列产品设计及制造商的了解，旨在建立信任和品牌忠诚。

致谢

我们非常感谢您花时间阅读水星Mercury X1轮式人形机器人用户手册。我们希望本文档能够帮助您更好地了解并有效使用这款机器人，从而激发您的创造力。如果您有任何疑问或需要进一步帮助，请随时联系我们的客户支持团队。我们期待看到您使用水星Mercury X1轮式人形机器人完成创新项目，并欢迎您加入我们快速发展的开发者社区。

文档目录

产品信息

- 1 产品介绍
- 2 产品参数

基础设置

- 3 用户须知
- 4 首次安装

功能与应用

- 5 软件开发指南
  - 5.1 基于Python 开发使用
    - 1 API简介
    - 2 python 案例
    - 3 拖动示教
  - 5.2 机器人操作系统 1 (ROS1)
    - 1 环境搭建
    - 2 ROS 基础
    - 3 Rviz 使用
    - 4 基本功能案例
  - 5.3 机器人操作系统 2 (ROS2)
    - 1 环境搭建
    - 2 ROS2 基础
    - 3 Rviz2 使用
    - 4 基本功能案例
- 6 机器人使用场景
  - 6.1 二维码识别与抓取
  - 6.2 空间移动案例
  - 6.3 多点位移动抓取案例

支持与服务

- 7 关于我们

致谢

- 8 致谢

1 产品概述

产品简介

水星Mercury X1是一款面向具身智能应用的轮式双臂人形机器人共拥有19自由度，由水星Mercury B1双臂机器人和高性能移动底座组合而成。整机搭载全新自研力源系列谐波模组。全新模块化设计模组标配电磁制动刹车，中空走线。具有高精度、高惯量、低自重等特点。同时配备英伟达Jetson Xavier主控，与独立四个副控协同工作。移动底座配备高性能激光雷达，超声波传感器和2D视觉等丰富感知；采用直驱电机驱动，最大运行速度达1.2m/s； 最大爬坡高度2CM；最大爬坡角度15度。整机最大续航高达8小时，满足个人及商业应用的需求。

特征	描述
名称	水星Mercury X1
类型	轮式双臂人形机器人
自由度	19自由度
组成	水星Mercury B1双臂机器人 + 高性能移动底座
续航	最大8小时
应用	科研、教育、服务、娱乐、远程操作

设计理念

水星Mercury X1轮式人形机器人的设计理念源于对当前科技教育和研究领域需求的深刻理解。机器学习、人工智能等高科技领域快速发展，要求实验设备不仅要有通用功能性，还需具备灵活性和精确性。设计旨在满足这些复杂需求，提供一个既灵活精确又能适应各种应用场景的解决方案。

组件	功能
谐波模组	提供高精度、高惯量、低自重
中空走线	便于维护和升级
英伟达Jetson Xavier	主控协同副控工作
激光雷达、超声波传感器、2D视觉	丰富感知能力
直驱电机	高性能驱动，最大速度1.2m/s

设计目标

设计目标	描述	应用场景及特点
通用多功能平台	水星Mercury X1轮式人形机器人适用于教育、研究和商业展示等多种应用场景。	其多自由度和 底盘移动导航支持在各种工作环境中执行复杂的运动控制，如精准定位和路径规划。
科研和教育支持	水星Mercury X1轮式人形机器人特别适合机器学习和人工智能研究，能执行高精度的实验和技术演示。	支持端到端数据训练与复现，以及基础的视觉任务，是实验室中的理想设备。
可编程性与扩展性	水星Mercury X1轮式人形机器人的高度可编程性允许用户根据新兴技术自定义和编程，以适应未来技术的需求。	通过用户自定义编程，设备能实现优化操作和实验结果，满足研究与发展的不断变化。
技术创新与知识传播	水星Mercury X1轮式人形机器人在商业展示中可以作为展示最新科技成果的平台，旨在提高公众对科技的理解和兴趣，并推动科技创新向商业化转化。	通过展示和演示最新的科技成果，增加公众参与度，促进科技知识的普及和科技产品的市场接受度。

产品特点

特点	优势
高性能移动底座	爬坡能力强，移动灵活
长续航	满足长时间工作需求
多感知能力	环境适应性强
模块化设计	易于维护和升级

产品价值

价值	影响
教育与科研	促进智能机器人领域的教育和研究
商业服务	提升商业活动的互动性和效率
个人娱乐	丰富个人娱乐体验，激发创意

行业贡献

贡献	描述
科研工具	为人形机器人研究提供实验平台
教育资源	作为教学工具，培养机器人技术人才
服务创新	推动服务行业向智能化转型

产品应用

应用场景	功能
科研	双臂协同控制、运动规划
教育	图像识别、机器人控制教学
服务	展会服务、公司展示
娱乐	类人动作模拟
远程操作	VR/Aloha遥控

3. 支持的扩展开发

水星Mercury X1轮式人形机器人在教育和科研领域中极具价值，特别是在 Python 和 ROS（Robot Operating System）这两个广泛使用的开发环境中。这些环境提供了强大的支持，使得 mercury 系列产品能够广泛应用于机器学习、人工智能研究、复杂运动控制以及视觉处理任务中。

Python	- 提供标准化的机器人驱动库，可通过 PyPI 进行下载和安装 - 支持查询关节角度数据、笛卡尔坐标数据等 - 这些功能支持使得 mercury 适合于基本的教育和研究场景特别是在数据采集和初步机器人控制方面
ROS	- 支持 ROS1 和 ROS2 双版本，提供 RVIZ 仿真环境支持。 - 允许用户实时显示机械臂和采集机械臂的状态信息，使得 mercury 适合 ROS 初学者和教育用途。
硬件接口	- 包括 IO、USB 等，方便连接各种传感器和执行器。
软件库	- 提供丰富的开源库和 API 来简化开发过程。

下一章 →

机器人参数说明

> 第一章中，我们探讨了产品的卖点及其设计理念，为您提供了对产品高层次理解的全景视角。现在，让我们进入第二章——机器人参数说明。这一章节将是您理解产品技术细节的关键。详细了解这些技术参数，不仅可以帮助您充分认识到我们产品的先进性和实用性，而且还能够确保您能够更有效地利用这些技术来满足您的具体需求。

1 机器规格参数

产品型号	Mercury X1
整机高度	1.18m
工作电压	24V
自由度	19自由度
续航时间	8小时
机械臂最大负载	1KG
机械臂重复定位精度	± 0.05mm
整机净重	55KG
底盘驱动电机	高性能直驱电机
最大运行速度	1.2m/s
最大爬坡角度	15°
主控	Nvidia Jetson Xavier NX
主控算力	21 TOPS
屏幕	9英寸触摸屏
移动地盘传感器	激光雷达，超声波雷达，2D视觉
3D 相机	奥比中光 Deeyea
麦克风	线性4麦，5米180°拾音
IO	24V 6 Input, 6 Output
通信方式	CAN 总线/WIFI/网口/蓝牙/USB/串口

2 软件基本功能支持

功能/开发环境	使用情况
自由移动	支持
关节运动	支持
笛卡尔运动	支持
轨迹录制	支持
无线控制	支持
紧急停止	支持
Windows	支持
Linux	支持
MAC	支持
ROS 1	支持
Python	支持
C++	支持
C#	支持
JavaScript	支持
myblockly	支持
Arduino	支持
mystudio	支持
串口控制协议	支持
TCP/IP	支持
MODBUS	支持

3 控制核心参数

主控制器规格表

指标	参数
主控	Jetson Xavier
主控型号	Jetson Xavier NX
CPU	6 核 NVIDIA Carmel ARM®v8.2 64 位 CPU 6MB L2 + 4MB L3
GPU	搭载48个Tensor核心的384核NVIDIA Volta™ GPU
AI 性能	21 TOPS
存储	16 GB eMMC 5.1
CSI 摄像头	2个CSI摄像头
网络	10/100/1000 BASE-T以太网
USB 接口	1 个 USB 3.2 2.0 (10 Gbps) 2 个 USB 2.0 接口
其他 I/O	2个UART串口

左右臂副控制器规格表

指标	参数
副控	左右臂副控
副控型号	ESP32
核心参数	240MHz dual core. 600 DMIPS, 520KB SRAM. Wi-Fi, dual mode Bluetooth
辅控Flash	4MB
LED显示	5X5 RGB

4 结构尺寸参数

> 本章以毫米为距离单位，以度为角度单位。

产品尺寸及工作空间

选择机器人安装位置时，必须考虑机器人正上方和正下方的圆柱体空间，尽可能避免将工具移向圆柱体空间。因为这样会造成工具运动较慢时，关节却转动过快，从而导致机器人工作效率低下，风险评估难以进行。

双臂末端法兰尺寸

图2.4.1 末端尺寸

DH值

图2.4.2 DH值

手臂部分

图2.4.3 手臂部分

5 电气特性参数

底座接口总览

Figure 1 底座正视图

Figure 2 底座左视图

Figure 3 底座右视图

Figure 4 底座内部图

底座接口说明

编号	接口	定义	功能	备注
1	电源输入接口	DC24V输入	DC24V 输入	
2	开关	电源开关	控制输入电源通断	带灯（通电灯亮）
3	急停接口	STOP	急停回路接口	
4	网口			
5	USB3.0	USB3.0*2	可外接设备或U盘	

1 电源输入接口：本接口与DC24V电源适配器接口连接

2 电源开关：控制总电源输入的通断，关闭时，控制器也断电

3 急停回路端子：与急停按钮盒连接，可用于控制机器人紧急停止

> **注**: 机器人使用中必须接上急停开关，并确保急停开关回路处于连通状态。

4 以太网

5 USB3.0接口：以串口总线标准3.0进行数据连接的接口；用户可以使用USB接口拷贝程序文件，也可以使用USB接口连接鼠标、键盘等外设

末端接口总览

Figure 5 左臂末端图

Figure 6 右臂末端图

末端接口说明

编号	接口	定义	功能	备注
6	4pinUSB端子	对外接口	连接摄像头	
7	M8航空插座	末端工具IO接口	与外部设备交互	

如图所示是M8航空插座I/O图，Mercury X1机器人提供了一路输入和两路输出。

FRONT VIEW

各个工具I/O端口的定义如下表所示，注意的是，工具I/O无论是输入还是输出都是PNP类型，接线方式同底部输出接口一致。

编号	信号	解释	配套M8线颜色
1	GND	DC24V 负极	白
2	OUT1	工具输出接口1	褐
3	OUT2	工具输出接口2	绿
4	485A	预留, 未开发	黄
5	24V	DC24V 正极	灰
6	IN1	工具输入接口1	粉
7	IN2	具输入接口2	蓝
8	485B	预留, 未开发	紫

USB端子：用于连接摄像头

如果您已经阅读了本章的所有内容，可以继续阅读下一章。

← 上一章 | 下一章 →

用户须知

> 本章节是使用本产品的每一位用户必须仔细阅读的重要部分。它涵盖了关于产品使用、运输、储存及维护的关键须知，旨在确保用户在操作产品时的安全性和效率。此外，本章节也详细说明了因未遵循这些指南而可能导致的产品故障或损害的责任划分。

1 安全须知

概要

本章详细介绍了对大象机器人进行安装、维护和修理工作的人员的一般安全信息。在搬运、安装、使用前，请仔细阅读并理解本章内容及注意事项。

危险识别

协作机器人的安全性基于机器人的正确配置和使用。此外，即使遵守了所有安全说明，操作员也可能会造成伤害或损坏。因此，了解机器人使用的安全风险以预防它们非常重要。

表1-1至表3列出了机器人使用过程中可能出现的常见安全风险：

表1-1 风险等级 安全风险

1 因机器人操作不当造成的人身伤害或机器人损坏。
2 如果机器人没有按照要求固定，例如螺丝缺失或螺丝不紧，或者底座的锁定能力不足以支撑机器人高速移动，机器人就会翻倒，导致人身伤害或机器人损坏。
3 由于安全功能配置不正确或缺乏安全防护工具，导致机器人安全功能未能发挥作用。

表1-2 安全风险提示

1 调试程序时，不要停留在机器人的运动范围内。安全配置不当可能无法避免碰撞，造成人身伤害。
2 机器人与其他设备之间的连接可能会带来新的危险，需要进行全面的风险评估。
3 小心工作环境中的其他设备或机器人末端执行器等尖锐表面造成的划伤和刺伤。
4 机器人是精密机器；踩到它们可能会造成损坏。运输过程中放置不当可能会导致振动，影响内部零件并造成损坏。因此，请确保在所有情况下的稳定性和机械结构完整性。
5 如果在机器人断电前（夹紧不牢固时）未移除被夹物，可能会导致末端执行器损坏或被夹物因断电而掉落受伤等危险。
6 存在机器人意外移动的风险。在任何情况下都不要站在机器人的任何轴下！
7 与普通机械设备相比，机器人具有更多的自由度和更大的运动范围。未能保持在运动范围内可能会导致意外碰撞。

表1-3 可能导致触电的安全隐患

1 使用非原装电缆可能会产生未知的危险。
2 电气设备接触液体可能引起漏电危险。
3 电气连接错误可能会导致触电。
4 更换前请务必关闭控制器及相关设备的电源并拔下电源插头。如果带电进行操作，可能会导致触电或故障。

安全注意事项

使用机械手时应遵守以下安全规则：

- 机械臂属于带电设备。非专业人员不得随意改变电路，否则可能对设备或人体造成损害。
- 操作机械臂时，请遵守当地法律法规。本手册中描述的安全注意事项和危险、警告和注意事项仅是对当地安全法规的补充。
- 请在规定的环境下使用机械臂。超过机械臂的规格和负载条件会缩短产品的使用寿命，甚至损坏设备。
- 无论如何，安装、操作和维护 mercury 手臂的人员必须接受有关安全预防措施以及操作和维护机器人的正确方法的严格培训。
- 无论如何，请勿在潮湿环境中长时间使用本产品。本产品为精密电子元件，长期处于潮湿环境中会损坏设备。
- 无论如何，请勿在潮湿环境中长时间使用本产品。本产品为精密电子元件，长期处于潮湿环境中会损坏设备。
- 高腐蚀性清洁不适合清洁机械臂，阳极氧化部件不适合浸泡清洁。
- 不知不觉中，请勿在未安装底座的情况下使用设备，以免损坏设备或发生事故，应在没有障碍物的固定环境中使用设备。
- 请勿使用其他电源适配器供电。如果因使用非标准适配器而导致设备损坏，则不包含售后服务。
- 请勿拆卸、拆解或拧开机械手的螺钉或外壳。若自行拆卸，则不提供保修服务。
- 未经专业培训的人员不得擅自修理故障产品和拆卸机械臂。如果产品出现故障，请及时联系mercury技术支持工程师。
- 如果产品被丢弃，请遵守相关法律，妥善处理工业废物，保护环境。
- 孩子在某个时刻使用设备，迫使某人监视该过程并在完成后将其关闭。
- 机器人运动时，请勿将手伸入机器人手臂的运动范围内，以免发生碰撞。
- 严禁更改、拆除或修改机械手及相关设备的铭牌、说明、图标和标志。
- 请小心搬运和安装。按照包装箱上的说明轻轻放好机器人，并按照箭头方向正确放置。否则，机器可能会损坏。

- **请勿从 Atom 终端刻录其他产品驱动程序，或使用非官方建议刻录固件。如果因用户烧录其他固件而导致设备损坏，则不属于售后服务范围。**

- 电源规格：**使用官方电源**
- USB Type-C 使用规范：**不要连接到电源板**

如果您对本手册内容有任何疑问或建议，请登录大象机器人官方网站并提交相关信息：

https://www.elephantrobotics.com

请不要将机械臂用于以下用途：

- 生命攸关的应用中的医疗保健成本。
- 购买公共汽车可能会导致环境爆炸。
- 四旬斋直接使用，无需进行风险评估。
- 使用低级别安全功能的成本。
- Lo-fi不符合机器人使用性能参数。

免责声明

请在使用产品前仔细阅读并理解以下免责声明：

- **安全使用：** 本产品专为特定应用场景而设计。确保在使用过程中遵循所有安全准则和操作手册。用户应接受有关产品使用的适当培训，并了解并遵守所有相关的安全法规。

- **责任限制：** 对于因使用或误用产品或与产品相关的任何事项而导致的任何直接、间接、偶然、特殊或后果性损害，制造商概不负责。本免责声明不涵盖或排除法律不允许排除的责任。

- **技术支持：** 请在安装和使用过程中仔细阅读产品文档，必要时寻求制造商的技术支持。有关技术支持问题，请参阅厂家提供的官方文档或联系相关支持渠道。

- **软件更新：** 制造商可能会提供产品固件或软件的更新。用户应定期检查并应用这些更新，以确保产品性能和安全性。

- **定期维护：** 用户应按照制造商提供的定期维护指南检查和维护产品。定期维护和检查有助于确保产品的长期性能。

- **定制和修改：** 未经制造商明确许可，不得定制、修改或变更产品。任何未经授权的修改都可能导致产品保修失效，并可能对安全和性能产生不可预测的影响。

- **法律合规性：** 用户应确保其使用符合所有适用的法律和法规。在某些地区，产品的使用可能受到特定法规的限制。

使用水星X1轮式双臂人形机器人，即表示您同意并接受这些免责声明。制造商保留更改产品规格、功能和免责声明的权利，恕不另行通知。

2 运输与保存

物流运输要求

温度	0°C~50°C
相对湿度	20%~70%
运输时的方向	机器人头部朝上，双臂垂下
运输时的外部条件	外部使用木架固定，防止挤压
木架尺寸	57*57*120

内部包装

外部包装

2 设备储藏

温度	0°C~50°C
相对湿度	20%~70%
运输时的方向	机器人头部朝上，双臂垂下
叠压要求	不可叠压
储藏环境	室内
其他环境要求	- 远离灰尘、油烟、盐分、铁屑等。 - 远离易燃性、腐蚀性液体与气体。 - 不得与水接触。 - 不传递冲击与振动等。 - 远离强电磁干扰源。

3 维护和保养

作为一家机器人制造商，我们重视确保客户能够正确、安全地维护和升级他们的机器人设备。为此，我们提供以下详细的维护和保养指南，包括常见维护项目及维修或升级硬件的部分，请您认真阅读。

常见维护项目及推荐周期

维护项目	描述	推荐周期
视觉检查	检查机器人有无明显的损坏、异物堆积或磨损	日常
结构清洁	使用干净、干燥的布料清洁机器人结构部件，避免水分和侵蚀性清洁剂	日常
紧固件检查	检查并紧固所有螺栓和连接件	日常
润滑	对关节和移动部件进行润滑，使用制造商推荐的润滑油	每3个月
电缆和接线检查	检查电缆和接线，确保无损坏或磨损	每月
电气连接检查	确保所有电气连接牢固，无腐蚀或损坏	每月
软件更新	检查并更新控制软件和应用程序	每次有更新时
软件数据备份	定期备份关键软件配置和数据	每季度
固件更新	定期检查并更新固件，以获取最新的功能和安全补丁	每次有更新时
传感器和器件检查	检查传感器和其他关键器件，确保正常工作	每月
紧急停止功能测试	定期测试紧急停止功能，确保其可靠性	每月
环境条件监控	监控工作环境的温度、湿度、灰尘等，确保符合机器人的操作规格。	持续监控
安全配置复查	定期检查和确认机器人的安全配置，如限速和工作范围设定	每月
预防性维护计划执行	按制造商的维护计划执行定期检查和维护	按制造商指南

独立更改机器人硬件的指南

我们理解客户可能会有自行升级或维修机器人硬件的需求。在进行任何升级操作之前，请务必详细阅读产品的相关参数，并与我们的官方人员确认是否被允许进行此类操作。未经官方允许的操作可能导致产品故障，且不在保修范围内。

物料要求

官方制造或推荐的物料：所有维修和升级所需的配件和组件必须是由我们官方制造或明确推荐的。这包括但不限于电子组件、传感器、电机、连接件和任何其他可更换部件。

物料获取：客户可通过我们的官方渠道购买所需的维修和升级物料。这确保了配件的质量和兼容性。

维修或升级流程

客户自行维修：客户应负责完成维修工作。我们将提供详细的维修指导和手册，以指导客户完成维修步骤。

遵循官方指导：维修操作应严格遵循我们提供的官方指导。任何偏离官方指导的操作都可能导致设备损坏。

责任和保修政策

责任划分：

制造商：提供维修和升级的官方指导、官方制造或推荐的物料，并处理由制造缺陷导致的问题。

客户：负责按照官方指导完成维修，使用官方配件。

保修政策：

保修有效：只有当维修操作完全遵循我们的指导，且使用官方配件时，保修才有效。

保修无效：若客户未按官方指导操作，或使用非官方配件进行维修或升级，导致的任何损坏都将不在保修范围内。

注意事项

安全第一：在进行任何维修或升级操作前，请确保遵循所有安全指南，包括断电和使用适当的防护装备。

技术支持：如在维修过程中遇到问题，建议停止操作并联系我们的技术支持团队寻求帮助。

我们强烈建议客户严格遵循这些指南，以确保机器人设备的安全、有效运行。不当的维修操作可能导致设备损坏并影响保修状态。如需进一步的指导或支持，请及时联系我们的专业技术团队。

4 常见问题

如何优雅地提问

1 在各地提问时，你会发现几种现象：

- 提问后没有得到答复。
- 提的问题过了很久才用人回答。
- 对方老是嫌弃自己太菜。

2 在提问之前，请确保已经研究过该手册。

很多问题会在这个过程中解决，切忌一开始就到 QQ 群、论坛 、issue， 或者邮件随便提问，很多在文档中一开始就说明了的问题可能不会得到社区的及时的回答，节省大家的时间，也为了更好的社区环境，大家共同更好地成长，请互相理解。

3 提问时，尽量做到以下几点，这将大大增加快速解决问题的机会：

要弄清楚发生了什么事，我做了什么，包括：

- 我想实现什么样的效果、什么功能？
- 为了达到这种效果，我是怎么做的，具体过程是怎样的？
- 在实施过程中，出现了什么错误，有什么现象（例如，报告的错误是什么，**完整**的错误内容是什么？）
- 我是否认真看了报错信息，报错信息中是否提示了错误原因和解决方法？
- 根据这些错误信息，仔细想想，我能解决问题吗？
- 搜索文档、issue，以及用搜索引擎能不能搜索到问题的解决方法？

4 如果自己实在解决不了问题，需要请人帮忙，需要考虑：

- 请教谁，在哪里请教，谁会更有几率回答我的问题？以及实时性如何？
- 我应该给他提供什么数据和现象他才能愿意快速帮我解决问题？
    - 提供我的目的（方便让回答者知道你在做什么事情）
    - 提供完整的实施过程，以及过程中出现的现象（方便回答者跟着你的流程去做一遍，即问题复现）
    - 给出错误的地方，指明哪里出现的现象或结果和自己预期的不一样！（让回答者知道，哪里没达到预期）
    - 提供出现的错误信息，需要完整，尽可能更多地截图，更多的日志，千万别小气地截一张小图，或者之给出日志的一部分（因为回答者可能很久没做这件事情了，忘记了一些细节，需要靠截图和完整日志来快速回忆起来;而且根据详细的日志可以快速定位到问题处在哪里）
- 怎么提问态度显得比较诚恳，就算我再小白，大家也愿意回答

5 提问模板

提问尽量做到提问优雅，不需要添加多余的语气词，抱怨的词汇，斟酌每一个词和标点符号，站在回答人的角度思考问题，如何让回答的人快速帮助自己解决问题，字数太少描述不清楚，字数太多则让人没有耐心。

6 标题

无论在哪里提问（包括 `QQ 群`），给自己的问题拟定一个 30 个字左右标题，阐明问题的中心思想，包括：

- 问题类别，是问题请教，还是 BUG 提交，还是经验分享等等。让大家在满是文字的屏幕上可以立马定位到你想做什么

- 一句话阐明问题的中心你思想，比如 `运行摄像头示例程序，报错 reset fail，可能是硬件问题`

所以综合后标题可以是这样：

- `【mercury 提问】 运行摄像头示例程序，报错 reset fail，可能是硬件问题吗`

这样的标题尽量**不要**出现:

- `啊啊啊啊啊为什么我的板子又不行了`
- `为什么我的代码不能运行了`
- `为什么我的屏幕黑屏了`
- `【mercury 提问】 收到了开发板，开发板屏幕是红的，一行小字，为什么呢`
- `我运行xxx程序，出了问题`

可以这样问：

- `【mercury 提问】 我的板子在我接反电源后，现在板子无法启动，怎么判断板子哪里烧毁了，如果可以，怎么救活呢`

7 内容

首先站在回答者的角度，如果被问问题:

- 首先要知道对方要做什么，实现什么目标
- 为了实现这个目标，他是参考了哪里的步骤做的
- 实际上采用了哪些具体的步骤，然后在那一步出现了问题，方便我照着他的步骤去尝试复现一下现象。如果这个问题看起来不好解决，又没有复现的步骤，那可能要花很多时间去复现，先放一放先解决其它问题吧
- 具体出现的问题是什么，如果他只说出了问题，我怎么知道他出了什么问题，也许是身体不适？所以这很重要，需要让他说明出问题时的现象，并指明与预期有什么不同，不然我还得去猜去对比和预期有什么不同，解决问题的时长又增加了
- 出问题了，我可能需要他的日志文件，方便我根据日志去看源码等进行分析，不然可能会很难解决问题，那这个问题可以放到后面慢慢看吧

综上，可以这么提问：

- 详细阐明自己的目标，想做什么事情，现象应该是什么样的
- 我有没有参考哪里的文档、代码或者教学
- 如何复现错误：具体怎么做的，每一个步骤写详细，直到问题出现
- 详细阐述错误出现时的现象，以及和期望的相比哪里不一样，需要证明问题确实出现了
- 附带日志文件，以及截图，甚至是视频，日志和截图一定要完整，不要只截取一小部分，回答者可能会从你的完整日志和截图中发现一些你没注意到的问题，这很重要
- 另外，粘贴代码注意格式，别粘贴后显示很乱，无法入目，尽量能复制后直接能运行
- 最后，对解答问题的社区朋友们表达感谢

驱动器相关

1 关于python

问： send_base_coords（[x，y，z，rx，ry，rz]， speed） 此 API 中的参数是什么意思？rx、ry 和 rz 对应于欧拉角什么？欧拉角的旋转顺序是什么？每个参数的取值范围是多少？

- 答：前面数组中的参数是 mercury X1 末端的坐标，speed 是速度。rx、ry 和 rz 应对应 RPY，即分别对应滚动、俯仰和偏航。欧拉角的阶数是 zyx，zyx 是它自己的坐标。x、y、z的取值范围。为-350~350，-350~350，-41~523.9（取值范围未定义，如果超过该范围，将返回逆运动学无解提示），rx、ry、rz的取值范围为-180~180。

问：不同版本的机械臂的 python api 是否相同？

- 答：API 是一样的。

2 关于ROS

问：您能提供 rviz 模型的文件和编程示例吗？

- 答：它可以在我们的 github 上找到。“https://github.com/elephantrobotics/mercury_x1_ros”

问：为什么使用 ROS 启动 rviz 模型文件时，报错权限“/dev/ttyUSB0”？

- 答：这是因为没有给出串口权限。您应该在终端中键入 sudo chmod 777 端口名称。 例如：

```
sudo chmod 777 /dev/ttyUSB0
```

问：为什么在运行 ROS 的滑块控件和模型遵循命令时，错误 _init_() takes exactly 2 arguments (3 given)？

- 答：pymercury 库未安装和启动。

问：使用 ROS 时，为什么打开 rviz 模型后 mercury_X1 角度与模型角度不一致？

- 答：很有可能mercury_X1的零位没有校准，mercury_X1的零位需要校准。

软件问题

关于ROS1

问:终端切换到~/catkin_ws/src，使用git安装更新mercury_x1_ros时，目标路径“mercury_x1_ros”已经存在。原因是什么？

- 答:这意味着在~/catkin_ws/src中已经有一个“mercury_x1_ros”包了。需要提前删除，然后重新执行git操作。

问: rosrun运行时，终端错误提示“could not open port /dev/ttyUSB0: Permission: '/dev/ttyUSB0'”。为什么？

- 答:串口权限不足。在终端输入“sudo chmod 777 /dev/ttyUSB0”，授予权限。

问:为什么ROS程序不能在VSCode中运行？

- 答:由于VSCode终端无法加载到ROS环境中，需要在系统终端中运行。

问: rosrun运行时，终端提示“无法注册主节点[http://localhost:11311]:主节点可能尚未运行”。我会继续努力的。”原因是什么?

- 答:运行ROS程序前，需要打开ROS节点，在终端中输入“roscore”。

问: rosrun运行时，终端错误提示“could not open port /dev/ ttyusb0: No such file or directory: '/dev/ttyUSB1'”。为什么?

- 答:串口错误。需要确认当前机械臂的实际串口。可以通过 `ls /dev/tty*` 查看。

问:在Ubuntu18.04中，' catkin_make '构建代码失败，终端提示' Project 'cv_bridge'指定'/usr/include/opencv'作为包含目录，没有找到。和其他错误信息

- 答:配置文件中的OpenCV路径与系统实际路径不匹配。你需要使用sudo命令修改配置文件(路径为“/opt/ros/melodic/share/cv_bridge/cmake/cv_bridgeConfig.cmake”)。系统的实际OpenCV路径位于“/usr/include/”路径下。

问:只需克隆mercury_x1_ros包，然后直接运行rosrun程序。出现诸如“package” mercury_x1_ros“not found”之类的错误或诸如无法找到文件之类的错误?

- 答:新克隆的mercury_x1_ros需要构建ROS环境编译的代码。终端输入

```
bash
cd ~/catkin_ws/
catkin_make
source devel/ setup.bash
```

关于机械臂控制

问:给机械臂发送角度或坐标机械臂没有运动

- 答：使用 `get_angles()` 读取机械臂的角度，若角度返回为空，则检查机械臂是否上电，使用 `power_on()` 对机械臂进行上电使能；检查端口号是否使用正确。若有角度，查看角度是否超出运动范围见表1，如果超出范围，使用 `release_all_servos()` 放松所有关节（注意关节放松后会下坠，需要接住），对齐1~5、7关节零刻度线，6关节与零刻度线90度垂直。再使用 `focus_all_servos()` 锁定关节，再使用 `set_servo_calibration(1)~set_servo_calibration(7)` 依次校准各关节零点，再使用 `get_angles()` 读取当前关节角度，如返回数据为[0, 0, 0, 0, 0, 90, 0]则为校准成功，否则重复之前的校准步骤。

表1-1 各关节角度可运动范围

关节	范围
J1	-175 ~ +175
J2	-65 ~ +115
J3	-175 ~ +175
J4	-180 ~ +10
J5	-175 ~ +175
J6	-20 ~ + 173
J7	-180 ~ +180

问:摄像头无法打开

- 答：打开本地相机查看是否可以切换到左右臂摄像头，如有某个摄像头无法查看，则重新拔插USB口或者更换USB口。 若可以切换到左右摄像头，检查摄像头端口是否发生变化(在重启后，端口可能发生变化)

问:自适应夹爪无法控制

- 答：查看自适应夹爪的电源指示灯，正常时指示灯应为常亮状态，若指示灯出现闪烁情况，请重新拔插夹爪与机械臂的连接线，指示灯恢复常亮状态则正常。 若指示灯为常亮状态仍无法控制，使用 `set_gripper_mode(0)` 更改夹爪使用模式。 **问:底座无法移动**

在终端输入

```
roslaunch turn_on_mercury_robot turn_on_mercury_robot.launch
rostopic echo /PowerVoltage
```

查看电池电量，若电量低于21V，底座无法使用，请充电后继续使用

硬件问题

1 关于结构

问:Atom在机械臂中的作用是什么?

- 答: Atom主要控制机械臂的运动学算法，包括正逆运动学、解选择、加减速、速度同步、多功率插补、坐标转换等。与atom相关的程序还不是开源的。

问:使用过程中电机自动切断电源。为什么?

- 答:电机长时间使用有过热保护。这种现象属于正常现象，几分钟后可以继续使用。

2 关于参数

问:机械臂的速度单位是多少?

- 答:运行速度为180度/秒。

问:自适应夹持器可以不完全关闭吗?

- 答:下巴之间会有一定的间隙，它们并不是完全闭合的。您可以通过增加它们之间的间隔的厚度来调整它。

问:机械臂末端USB摄像头如何固定?

- 答:需要用法兰固定，可以单独购买。

如果您有购买意向或任何参数问题，请发送电子邮件至此邮箱。 邮箱 : "sales@elephantrobotics.com"

如果列出的问题不能帮您解决，您有更多售后问题，请发送邮件至此邮箱。 邮箱 : "support@elephantrobotics.com"

← 上一章 | 下一章 →

1 产品标准列表

产品标准清单对照表

序列号	产品
1	水星Mercury X1轮式人形机器人
2	产品手册
3	合格证书
4	使用说明
5	发货清单
6	电源(24V)

注意: 包装框到位后，请确认机器人包装完整。如果有任何损失，请及时联系物流公司和您所在地区的供应商。解开包装后，请根据项目列表检查框中的实际内容。

---

2 产品开箱指南

产品开箱图形指南

为什么您需要按照以下步骤移出产品

在本节中，我们强烈建议按照指定的步骤移出产品。这不仅有助于确保产品在运输过程中不会损坏，还可以将意外故障的风险降至最低。请仔细阅读以下图形指南，以确保您的产品在开箱过程中是安全的。

- **1** 检查箱体是否损坏。如有损坏，请及时与您所在地区的物流公司和供应商联系。
- **2** 打开包装盒，取出使用说明、海绵包装盖、水星Mercury X1轮式人形机器人、配套电源、发货清单。
- **3** 在进行下一个步骤之前，请确保每一步都已完成，以防止不必要的损坏或遗漏。

注意: 取下产品后，请仔细检查每件物品的外观。请将箱子里的实际项目与项目清单核对一下。

开箱图片

3 夹爪、摄像头安装指南

4 开机检测指南

外部电缆连接

操作前请仔细阅读 **章节安全说明** ，确保操作安全。同时将电源适配器与底座连接。

将电源适配器连接上机械臂，也可通过内置电源供电，并将机器人移至空旷区域防止碰撞。

底座后部电源接口位置

电源开关位置

急停开关位置

在按下急停开关后机器人将立即停止运动，复位时需将急停开关顺时针旋转，急停开关将自动弹起，之后可上电恢复控制。

点击电源开关，并确保急停处于复位状态，开始使用机械臂。

底部LED灯亮起

进入开机界面

屏幕点亮

进入登录界面后输入开机密码**Elephant**

基础功能检测

检测双臂是否可以正常运动：

1. 打开终端进入python环境
2. 导入pymycobot包，初始化左右臂并上电
3. 使用get_angle()获取关节角度,若能获取到角度则上电成功

4. 校准零点，再次使用get_angle()获取关节角度，当返回为 [0.0, 0.0, 0.0, 0.0, 0.0, 90.0, 0.0]时零点校准成功

5. 对左右臂进行控制看是否能运动，若成功运动则机械臂正常

```
er@er-desktop:~$ python
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pymycobot import Mercury
>>> mr = Mercury("/dev/ttyACM0")
>>> ml = Mercury("/dev/ttyTHS0")
>>> ml.power_on()
1
>>> mr.power_on()
1
>>> ml.get_angles()
[-362.31, -98.48, -254.57, 15.59, -31.44, 3.67, 1.29]
>>> mr.get_angles()
[-7.07, -95.34, -111.56, 19.93, -713.43, -71.2, 3.35]
```

上电后获取关节角度

检测移动底座是否可以正常运动：

1. 在终端输入下列指令启动底座控制

```
roslaunch turn_on_mercury_robot turn_on_mercury_robot.launch
```

```
er@er-desktop:~$ roslaunch turn_on_mercury_robot turn_on_mercury_robot.launch
... logging to /home/er/.ros/log/dc68a2e6-fe33-11ee-bacd-e8fb1c355a09/roslaunch-e
r-desktop-6122.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://er-desktop:40689/

SUMMARY
========

PARAMETERS
 * /if_akm_yes_or_no: no
 * /mercury_robot/gyro_frame_id: gyro_link
 * /mercury_robot/odom_frame_id: odom
 * /mercury_robot/robot_frame_id: base_footprint
 * /mercury_robot/serial_baud_rate: 115200
 * /mercury_robot/usart_port_name: /dev/wheeltec_con...
 * /robot_description: <?xml version="1....
 * /robot_pose_ekf/base_footprint_frame: base_footprint
 * /robot_pose_ekf/freq: 30.0
 * /robot_pose_ekf/imu_used: True
 * /robot_pose_ekf/odom_data: odom
 * /robot_pose_ekf/odom_used: True
 * /robot_pose_ekf/output_frame: odom
 * /robot_pose_ekf/sensor_timeout: 2.0
 * /robot_pose_ekf/vo_used: False
 * /rosdistro: noetic
 * /rosversion: 1.16.0
```

1. 再打开一个新的终端，输入下列指令启动键盘控制移动底座。根据提示进行按键控制，若小车正常运动则移动底座正常。

```
roslaunch mercury_x1_teleop keyboard_teleop.launch
```

检测激光雷达是否正常

1. 打开终端输入

```
roslaunch turn_on_mercury_robot mapping.launch
```

2. 打开一个新的终端输入

```
roslaunch turn_on_mercury_robot cobotx_rviz.launch
```

若自动打开的rviz中有如图白色区域，则激光雷达扫描正常

← 上一章 | 下一章 →

软件开发指南

1 使用环境

水星Mercury X1轮式人形机器人 是基于内置ubuntu系统开发和使用的。

2 开发环境

为了满足机器人在不同场景下的多样化应用需求，我们对机器人进行了多种编程语言的适配。到目前为止，我们已经适配了以下主流编程语言，我们认为您可以使用以下任何一种语言进行开发。请务必严格按照说明进行操作。任何遗漏的步骤都可能导致相应语言无法成功运行。祝您顺利使用机器人。

- Python
  我们的机器人支持 Python，Python API 库的开发也日趋完善。机器人的关节角度、坐标、抓手和其他方面都可以通过 Python 进行控制。
- ROS1
  ROS（Robot Operating System）作为一个开源的机器人操作系统，为机器人的开发和控制提供了无限的可能性。我们的机器人可以通过ROS丰富的控制功能，通过模块化的方式对机器人进行控制。无论是关节控制、路径规划还是感知反馈，ROS都提供了相应的工具和库，使得控制过程更加灵活和高效。
- ROS2
  ROS 2（Robot Operating System 2）是一个灵活的软件框架，专为机器人软件开发而设计。我们的机器人可以通过它提供的硬件抽象、设备驱动、库函数、可视化工具、消息传递以及包管理等一系列服务和功能，使得应用程序的开发变得更为高效和模块化。

← 上一章 | 下一页 →

环境配置

> pymycobot是Elephant Robot开发的Python库，用于机器人控制。

Linux

系统出厂时默认安装了Python 3.8.10，并且已经安装了 `pymycobot` 控制库，用户无需自行安装。

pymycobot 安装

通过终端输入命令即可安装pymycobot

```
pip install pymycobot
```

pymycobot 卸载

您可以通过终端输入命令来卸载pymycobot

```
pip uninstall pymycobot
```

pymycobot 更新

您可以通过终端输入命令来更新pymycobot

```
pip install pymycobot -U
```

窗口

1.1 安装Python

注意： 安装前，请检查PC的操作系统。 在“我的电脑”图标上按右键，然后选择“属性”。安装相应的Python。

- **前往http://www.python.org/download/下载Python。**
- **点击“下载”，然后开始下载。 勾选“将 Python 3.10 添加到 PATH”。 单击“立即安装”，然后开始安装。**

- **下载并安装完成。**

1.2 运行Python

打开命令提示符窗口（Win+R，输入“cmd”并按“Enter”）。 输入“Python”。

安装成功：

屏幕上出现此指示意味着 Python 已成功安装。 提示符“>>>”表示Python交互环境。 如果输入一段Python代码，立即得到执行结果。

错误报告：

如果输入了错误的指令，例如“pythonn”，系统可能会报告错误。

> **注意：** 一般情况下，该错误是由于环境配置不足造成的。 参考**1.3 环境配置**解决问题。

1.3 环境变量配置

Windows 遵循 Path 环境变量设置的路径来搜索 **python.exe** 。 否则会报错。 如果安装时没有勾选“Add Python 3.9 to PATH”，则需要手动将python.exe所在路径添加到环境变量中或者重新下载python。 请记住勾选“将 Python 3.9 添加到 PATH”。

按照以下步骤手动将 python 添加到环境变量中。

- 右键单击“我的电脑”图标 --> 属性 -> 高级系统设置 -> 环境变量

- 环境变量包括用户变量和系统变量。 对于用户变量，用户可以通过“cmd”命令使用自己下载的程序。 将目标程序的绝对路径写入用户变量中。

- 配置完成后，打开命令提示符窗口（Win+R；输入“cmd”并按“Enter”），然后输入“Python”。

2 PyCharm的安装

PyCharm 是一款功能强大、具有跨平台性质的 Python 编辑器。 请按照以下步骤下载并安装 PyCharm。

前往 **PyCharm** 下载 PyCharm。

2.1 下载与安装

官网查看：

建议安装免费版本。

- 点击“下一步”：

- 根据您的需要选择选项，然后选择“下一步”：

PyCharm Community Edition Setup
Installation Options
Configure your PyCharm Community Edition installation
Create Desktop Shortcut
PyCharm Community Edition
Update PATH Variable (restart needed)
Add "bin" folder to the PATH
Update Context Menu
Add "Open Folder as Project"
Create Associations
.py
< Back
Next >
Cancel

- 点击“安装”：

PyCharm Community Edition Setup
Choose Start Menu Folder
Choose a Start Menu folder for the PyCharm Community Edition shortcuts.
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.
JetBrains
#LancsBox 6.0.0
360安全中心
ABBYY Aligner 2.0
Accessibility
Accessories
Administrative Tools
Chrome 应用
EndNote
Git
IBM SPSS Statistics
Java
JetBrains
< Back
Install
Cancel

- 安装：

- 点击“完成”

2.2 创建新项目

- 点击“+新建项目”：
- `Interpreter` 用于解释Python程序。 选择“添加解释器”->“新建”以添加基本解释器。

- `Location` 是指保存python文件的位置。 选择一个文件来放置您的程序。

- 点击“创建”，会出现一个示例：
- 右键单击红色箭头所指的选项，然后创建一个新的 python 文件。
- 键入新文件的名称。

3 准备工作

- pymycobot 安装。 通过终端（Win+R）“cmd”命令输入“pip install pymycobot --upgrade --user”。

```
pip install pymycobot --upgrade --user
```

- 源码安装。 打开终端（Win+R，输入 `cmd` ），然后输入以下命令进行安装。

```
git clone https://github.com/elephantrobotics/pymycobot.git <your-path>
#<your-path>中填写你的安装地址，不要选择当前默认路径。

cd <your-path>/pymycobot
#进入下载包的pymycobot文件夹。

#根据您的python版本运行以下命令之一。
# install
 python2 setup.py install
# or
 python3 setup.py install
```

- 更新pymycobot

```
pip install pymycobot --upgrade
```

> **注意:**
>
> 1. 如果代码下方没有出现红色波浪线，则说明pymycobot安装成功。
> 2. 如果出现红色波浪线，请前往地址 **https://github.com/elephantrobotics/pymycobot** 手动下载 pymycobot 并将其放入 python 库中。

Python的基本使用

```python
from pymycobot import Mercury

ml = Mercury('/dev/left_arm')
mr = Mercury('/dev/right_arm')

ml.power_on()
mr.power_on()

print(ml.get_angles())
print(mr.get_angles())
```

← 上一页 | 下一页 →

6.1 Python API

6.1.1 API使用说明

API（ApplicationProgrammingInterface），也称为应用程序编程接口函数，是预定义的函数。 使用以下功能接口时，请在开始时输入以下代码导入我们的API库，否则无法成功运行：

```
# 例子
from pymycobot import Mercury

ml = Mercury('/dev/left_arm')
mr = Mercury('/dev/right_arm')

ml.power_on()
mr.power_on()

print(ml.get_angles())
print(mr.get_angles())
```

1.系统状态

`get_system_version()`

- **功能：** 获取系统版本
- **返回值：** 系统版本

`get_robot_type()`

- **功能：** 获取机器人id
- **返回值：** 定义规则：实际机器型号。 例如水星A1型号为4500

`get_atom_version()`

- **功能：** 获取最终版本号
- **返回值：** 结束参数(`float`)

`get_robot_status()`

- **功能：** 上位机错误安全状态
- **返回值：** 0 - 正常。 其他 - 机器人触发碰撞检测

2.总体状况

`power_on()`

- **功能：** atom开放通讯（默认开放）
  - 注意： 执行关机或按紧急停止后，需要 7 秒才能上电并恢复供电
- **返回值：**
  - `1` - 开机完成。
  - `0` - 开机失败

`power_off()`

- **功能：** 机械臂断电
- **返回值：**
  - `1` - 开机完成。
  - `0` - 开机失败

`is_power_on()`

- **功能：** 判断机械臂是否通电
- **返回值：**
  - `1`：开机
  - `0`： 关闭电源
  - `-1`：错误

`release_all_servos()`

- **功能：** 释放所有机械臂
  - 注意：关节禁用后，需在1秒内启用才能控制
- **参数**： `data` （可选）： 放松关节的方式。 默认为阻尼模式，如果提供了'data'参数，则可以指定为非阻尼模式（1-Undamping）。
- **返回值：**
  - `1` - 发布完成。
  - `0` - 释放失败

`focus_all_servos()`

- **功能：** 打开机器人扭矩输出
  - **返回值：**
    - `1`：完成
    - `0`： 失败
    - `-1`：错误

3.MDI模式及操作

`get_angles()`

- **功能：** 获取全关节角度
- **返回值**： `list` 所有级别的浮点列表

`get_angle()`

- **功能：** 获取单关节角度
- **参数：** joint_id (int): 1 ~ 7
- **返回值：** 关节对应的角度数组

`send_angle(id, degree, speed)`

- **功能：** 控制单关节旋转
- **参数：**
  - `id`：关节id(`genre.Angle`)，范围 int 1-7 or 1-6
  - `degree`：角度值(`float`)

7轴版本手臂关节活动范围

关节id	角度
1	-165 ~ 165
2	-55 ~ 95
3	-165 ~ 165
4	-180 ~ 10
5	-165 ~ 165
6	-20 ~ 273
7	-180 ~ 180

6轴版本手臂关节活动范围

关节id	角度
1	-165 ~ 165
2	-55 ~ 95
3	-180 ~ 5
4	-165 ~ 165
5	-20 ~ 273
6	-180 ~ 180

身体关节的活动范围。 第 11 关节是下巴摄像头。 第 12 关节是颈部。 第13关节是腰椎关节

关节id	角度
11	-60 ~ 0
12	-138 ~ 188
13	-118 ~ 118

- `speed`： 机械臂运动速度和幅度 1~100

`send_angles(angles, speed)`

- **功能：** 控制全关节旋转
- **参数：**
  - `angles`： 角度值列表(`List[float]`)，长度 7 or 6
  - `速度`：(`int`) 1 ~ 100

`get_base_coords()`

- **功能：** 获取机器人Base坐标
- **返回值：** 坐标的浮点列表：[x, y, z, rx, ry, rz]

`send_base_coord(axis, coord, speed)`

- **功能：** 控制机械臂末端沿指定轴运动到目标位置
- **参数：**
  - `axis`： 轴号，1-6对应[x, y, z, rx, ry, rz]
  - `coord`：坐标值(`float`)

坐标id	范围
1	-328.03 ~ 554.71
2	-637.68 ~ 637.68
3	-206.52 ~ 751.37
4	-180 ~ 180
5	-180 ~ 180
6	-180 ~ 180

- `speed`：(`int`) 1-100

`send_base_coords(coords, speed, mode)`

- **功能：**： 控制机械臂末端运动到指定坐标
- **参数：**
  - coords:： 坐标值列表 `[x,y,z,rx,ry,rz]` ,length6
  - 速度 `(int)`：1 ~ 100

`pause()`

- **参数:**
  - `1` - 缓停
  - `0` - 急停
- **功能:** 控制指令暂停核心并停止所有运动指令

`is_paused()`

- **功能:** 检查程序是否暂停了移动命令
- **返回值:**
  - `1` - 暂停
  - `0` - 未暂停
  - `-1` - 错误

`resume()`

- **功能:** 恢复机器人运动并完成之前的命令

`stop()`

- **参数:**
  - `1` - 缓停
  - `0` - 急停
- **功能:** 停止机器人的所有动作
- **返回值**:
  - `1` - 停止
  - `0` - 不停止
  - `-1` - 错误

`is_in_position(data, flag)`

- **功能**：判断是否在位。
- **参数:**
  - data：提供一组数据，可以是角度或坐标值。 如果输入角度长度范围为7，如果输入坐标值长度范围为6
  - 标志数据类型（值范围0或1）
    - `0`：角度
    - `1`：坐标
- **返回值**:
  - `1` - 是
  - `0` - 否
  - `-1` - 错误

`is_moving()`

- **功能:** 判断机器人是否在移动
- **返回值:**
  - `1` 移动

- `0` 静止
- `-1` 错误

4. JOG 模式及操作

`jog_angle(joint_id, direction, speed)`

- **功能：** 点动控制角度
- **参数**：
  - `joint_id`： 代表机械臂的关节，用关节ID表示，范围为1到7
  - `direction(int)`： 控制机械臂运动方向，输入 `0` 为负值运动，输入 `1` 为正值运动
  - `速度`： 1 ~ 100

`jog_base_coord(coord_id, direction, speed)`

- **功能：** 点动控制坐标。
- **参数：**
  - `coord_id`：(`int`) 机械臂坐标范围：1~6
  - `direction`:(`int`) 控制机械臂运动方向， `0` - 负值运动， `1` - 正值运动
  - `速度`： 1 ~ 100

`jog_increment_angle(joint_id, increment, speed)`

- **功能：** 单关节角度增量控制
- **参数**：
  - `joint_id`： 1-7 or 1-6
  - `increment`： 根据当前位置角度增量移动
  - `速度`： 1 ~ 100

`jog_increment_coord(coord_id, increment, speed)`

- **功能：** 单关节角度增量控制
- **参数**：
  - `joint_id`： 轴ID 1 - 6。
  - `increment`： 基于当前位置坐标的增量运动
  - `速度`： 1 ~ 100

5.零空间运动

`get_solution_angles()`

- **功能：** 获取零空间偏转角度值
- **返回值**：零空间偏转角度值

`set_solution_angles(angle, speed)`

- **功能：** 获取零空间偏转角度值
- **参数：**

- `angle` ：输入关节1的角度范围，角度范围-90到90
- `速度` ： 1 - 100。

6.软件限位

`get_joint_min_angle(joint_id)`

- **功能：** 读取最小关节角度
- **参数：**
  - `joint_id` ： 输入关节 ID（范围 1-7 or 1-6）
- **返回值：** `float` 角度值

`get_joint_max_angle(joint_id)`

- **功能：** 读取最大关节角度
- **参数：**
  - `joint_id` ： 输入关节 ID（范围 1-7 or 1-6）
- **返回值：** `float` 角度值

`set_joint_min(id, angle)`

- **功能：** 设置最小关节角度限制
- **参数：**
  - `id` ： 输入关节 ID（范围 1-7 or 1-6）
  - `angle` ： 参考send_angle()接口中对应关节的限制信息，不得小于最小值

`set_joint_max(id, angle)`

- **功能：** 设置最小关节角度限制
- **参数：**
  - `id` ： 输入关节 ID（范围 1-7 or 1-6）
  - `angle` ： 参考send_angle()接口中对应关节的限制信息，不得大于最大值

7.关节电机控制

`is_servo_enable(servo_id)`

- **功能：** 检测关节连接状态
- **参数：** `舵机id` 1-7 or 1-6
- **返回值：**
  - `1` ：连接成功
  - `0` ： 未连接
  - `-1` ：错误

`is_all_servo_enable()`

- **功能：** 检测所有关节连接的状态
- **返回值：**
  - `1` ：连接成功

- `0`： 未连接
- `-1`：错误

`set_servo_calibration(servo_id)`

- **功能：** 校准关节执行器当前位置为角度零点
- **参数**:
  - `servo_id`： 1 - 7 or 1-6

`release_servo(servo_id)`

- **功能：** 设置指定关节扭矩输出关闭
- **参数**:
  - `servo_id`：1 ~ 7 or 1-6
- **返回值：**
  - `1`：释放成功
  - `0`：释放失败
  - `-1`：错误

`focus_servo(servo_id)`

- **功能**：设置指定关节扭矩输出开启
- **参数**： `servo_id`： 1 ~ 7 or 1-6
- **返回值：**
  - `1`： 聚焦成功
  - `0`： 焦点失败
  - `-1`：错误

`set_break (joint_id, value)`

- **功能：** 设置断点
- **参数**:
  - `joint_id`： int。 关节 ID 1 - 7 or 1-6
  - `值`： 整数。 0 - 禁用，1 - 启用
- **返回值：** 0：失败； 1：成功

`get_servo_speeds()`

- **功能**：获取所有关节的运动速度
- **返回值**： 单位step/s

`get_servo_currents()`

- **功能**：获取所有关节的运动电流
- **返回值**： 0 ~ 5000 mA

`get_servo_status()`

- **功能**：获取所有关节的运动状态
- **返回值**：值为0表示没有错误

`servo_restore(joint_id)`

- **功能**：清除关节异常
- **参数**：
  - `joint_id`： int。 关节 ID 1 - 7 or 1-6

8.机械臂端IO控制

`set_digital_output(pin_no, pin_signal)`

- **功能：** 设置IO状态
- **参数**
  - `pin_no` (int): PIN 码
  - `pin_signal` (int): 0 / 1

`get_digital_input(pin_no)`

- **功能：** 读取IO状态
- **参数**： `pin_no` (int)
- **返回值**：信号

9. 机械臂末端夹具控制

`set_gripper_state(flag, speed, _type_1=None)`

- **功能**：自适应夹具启用
- **参数**：
  - `flag (int)`：0 - 打开 1 - 关闭，254 - 释放
  - `速度（整数）`： 1 ~ 100
  - `_type_1 (int)`：
    - `1`： 自适应夹具（默认状态为 1）
    - `2`：一只灵活的手，有 5 个手指
    - `3`： 平行夹具
    - `4`： 柔性夹具

`set_gripper_value(gripper_value, speed, gripper_type=None)`

- **功能**：设置夹具值
- **参数**：
  - `gripper_value (int)`：0 ~ 100
  - `speed (int)`： 1 ~ 100
  - `gripper_type (int)`：

        - `1`：自适应夹具（默认状态为 1）
        - `2`：一只灵活的手，有 5 个手指
        - `3`：平行夹具
        - `4`：柔性夹具

`set_gripper_calibration()`

- **功能**：将夹具的当前位置设置为零

`set_gripper_enabled(value)`

- **功能**：自适应夹具启用设置
- **参数**：
    - `value` 1：启用 0：释放

`set_gripper_mode(mode)`

- **功能**：设置夹具模式
- **参数**：
    - `值`：
        - 0：透传模式
        - 1：正常模式

`get_gripper_mode()`

- **功能**：获取抓手模式
- **返回值**：
    - 0：透传模式
    - 1：正常模式

10.机械臂末端的按钮功能

`is_btn_clicked()`

- **功能**：获取机械臂末端按钮的状态
- **返回值**：
    - 0：没有点击
    - 1：点击

`set_color(r, g, b)`

- **功能**：设置机械臂末端灯颜色
- **参数**：
    - `r (int)`：0 ~ 255
    - `g (int)`：0 ~ 255
    - `b (int)`：0 ~ 255

11.拖动示教

`drag_teach_save()`

- **功能：** 放松关节并开始录制点位。
  - 注意：为了显示最佳运动效果，录制时间不应超过90秒

`drag_teach_pause()`

- **功能：** 暂停采样

`drag_teach_execute()`

- **功能：** 执行拖动示教。

`drag_teach_clean()`

- **功能：** 清除采样。

12.笛卡尔空间坐标参数设置

`set_tool_reference(coords)`

- **功能：** 设置工具坐标系。
- **参数：** `coords`：(`list`) [x, y, z, rx, ry, rz]。
- **返回值：** NULL

`get_tool_reference(coords)`

- **功能：** 获取工具坐标系。
- **返回值：** `oords`：(`list`) [x, y, z, rx, ry, rz]

`set_world_reference(coords)`

- **功能：** 设置世界坐标系。
- **参数：** `coords`：(`list`) [x, y, z, rx, ry, rz]。
- **返回值：** NULL

`get_world_reference()`

- **功能：** 获取世界坐标系。
- **返回值：** `list` [x, y, z, rx, ry, rz]。

`set_reference_frame(rftype)`

- **功能：** 设置基础坐标系。
- **参数：** `rftype`：0 - 基础 1 - 工具。

`get_reference_frame()`

- **功能：** 设置基础坐标系。
- **返回值：**
  - `0` - 基数
  - `1` - 工具。

`set_movement_type(move_type)`

- **功能：** 设置运动类型。
- **参数：**
  - `move_type`： 2 - fusion 1 - movel，0 - moveJ。

`get_movement_type()`

- **功能：** 获取运动类型。
- **返回值：**
  - `2` - 速度融合
  - `1` - 移动
  - `0` - moveJ

`set_end_type(end)`

- **功能：** 获取末端坐标系
- **参数：**
  - `end (int)`： `0` - 法兰， `1` - 工具

`get_end_type()`

- **功能：** 获取末端坐标系
- **返回值：**
  - `0` - 法兰
  - `1` - 工具

13.圆周运动

`write_move_c(transpoint, endpoint, speed)`

- 功能：圆弧轨迹运动
- 参数： `transpoint(list)`： 经过点坐标的圆弧 `endpoint(list)`： 圆弧端点坐标 `速度(int)`： 1 ~ 100

14.设置底层IO输入/输出状态

`set_basic_output(pin_no, pin_signal)`

- **功能**：设置基本IO输出
- **参数**：
  - `pin_no` (`int`) Pin 端口号，范围 1 ~ 6
  - `pin_signal` (`int`)：0 - 低。 1 - 高

`get_basic_input(pin_no)`

- **功能**：设置基本IO输出
- **参数**：
  - `pin_no` (`int`) Pin 端口号，范围 1 ~ 6
  - `pin_signal` (`int`)：0 - 低。 1 - 高

`tool_serial_restore()`

- **功能**：485恢复出厂设置

`tool_serial_ready()`

- **功能：** 设置485通讯
- **返回值：** 0：未设置 1：设置完成

`tool_serial_available()`

- **功能：** 设置485通讯
- **返回值：** 0-正常 1-机器人触发碰撞检测

`tool_serial_read_data()`

- **功能：** 读取固定长度数据。 读取之前，先读取缓冲区长度。 读取后数据将被清除
- **参数**： data_len (int)：要读取的字节数，范围1 ~ 45
- **返回值：** 0：未设置 1：设置完成

`tool_serial_write_data()`

- **功能：** 结束485发送数据，数据长度范围为1 ~ 45字节
- **返回值：** 0-正常 1-机器人触发碰撞检测

`tool_serial_flush()`

- **功能：** 清除485缓冲区
- **返回值：** 0-正常 1-机器人触发碰撞检测

`tool_serial_peek()`

- **功能：** 查看缓冲区中的第一条数据，数据不会被清除
- **返回值：** 1字节数据

`tool_serial_set_baud(baud)`

- **功能：** 设置485波特率，默认115200
- **参数**：波特率（int）：波特率
- **返回值：** NULL

`tool_serial_set_timeout(max_time)`

- **功能：** 设置 485 超时时间（以毫秒为单位），默认 30ms
- **参数**
  - `max_time`: (int): 超时
- **返回值：** NULL

6.1.2 底盘基础控制接口

API使用说明

```
# 示例
from pymycobot import ChassisControl

mc = ChassisControl()

# 获取电池电压
voltage = mc.get_power_voltage()
print(voltage)

# 设置底盘RGB灯带为绿色
mc.set_color(255, 0, 0)
```

get_power_voltage()

- **功能：** 获取电池电压。
- **返回值：** 电池电压（浮点型），单位：伏特

get_ultrasonic_value()

- **功能：** 获取所有超声波数据。
- **返回值：** 所有超声波数据列表，单位：毫米。

go_straight(speed)

- **功能：** 控制底盘前进。
- **参数：**
  - speed：（浮点型）底盘运动的速度，范围是 `0 ~ 1`，单位：米/秒。

go_back(speed)

- **功能：** 控制底盘后退。
- **参数：**
  - speed：（浮点型）底盘运动的速度，范围是 `-1 ~ 0`，单位：米/秒。

turn_left(speed)

- **功能：** 控制底盘向左旋转。
- **参数：**
  - speed：（浮点型）底盘运动的速度，范围是 `0 ~ 1`，单位：米/秒。

turn_right(speed)

- **功能：** 控制底盘右旋转。
- **参数：**
  - speed：（浮点型）底盘运动的速度，范围是 `-1 ~ 0`，单位：米/秒。

stop()

- **功能：** 停止运动。

set_color(r, g, b)

- **功能：** 设置底盘RGB灯带颜色。
- **参数：**
  - r (int)：0 ~ 255
  - g (int)：0 ~ 255
  - b (int)：0 ~ 255

← 上一页 | 下一页 →

案例 1

将末端灯的颜色设置为蓝色

```
from pymycobot import Mercury

ml = Mercury("/dev/left_arm")
mr = Mercury("/dev/right_arm")
# 机器人上电
ml.power_on()
mr.power_on()

ml.set_color(0,0,255)
mr.set_color(0,0,255)
```

案例 2

角度控制

```
from pymycobot import Mercury
import time
ml = Mercury("/dev/left_arm")
mr = Mercury("/dev/right_arm")
# 机器人上电
ml.power_on()
mr.power_on()
# 单角度控制
ml.send_angle(1, 90, 40)
mr.send_angle(1, 90, 40)
time.sleep(3)
ml.send_angle(1, 0, 40)
mr.send_angle(1, 0, 40)
time.sleep(3)

# 所有角度控制
ml.send_angles([0, 0, 90, 0, 0, 90, 0], 40)
mr.send_angles([0, 0, 90, 0, 0, 90, 0], 40)
time.sleep(3)
ml.send_angles([0, 0, 0, 0, 0, 90, 0], 40)
mr.send_angles([0, 0, 0, 0, 0, 90, 0], 40)
```

← 上一页 | 下一节 →

拖动教学

拖动*教学*保存()

开始录制并拖动教学点。

- 注意：为了显示最佳运动效果，录制时间不应超过90秒

拖动*教学*暂停()

暂停采样。暂停采样并再次启用机械臂

拖动*教学*执行()

开始拖动示教点，仅执行一次。

案例

```
from pymycobot import Mercury
import time
ml = Mercury("/dev/left_arm")
mr = Mercury("/dev/right_arm")

ml.power_on()
mr.power_on()
# 左臂开始轨迹记录
ml.drag_teach_save()
# 记录时长为10秒
time.sleep(10)
# 停止记录
ml.drag_teach_pause()
time.sleep(1)
# 开始执行记录的运动轨迹，仅执行一次
ml.drag_teach_execute()
```

← 上一页 | 下一章 →

ROS

ROS是用于机器人的开源的元操作系统。它提供了操作系统应有的服务，包括硬件抽象、低级设备控制、常用功能的实现、进程之间的消息传递以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。

ROS 运行时的“graph”是一种基于ROS通信基础结构的松耦合点对点进程网络。ROS实现了几种不同的通信方式，包括基于同步RPC样式通信的服务（services）机制，基于异步流媒体数据的话题（topics）机制以及用于数据存储的参数服务器。

ROS并不是一个实时的框架，但ROS可以嵌入实时程序。Willow Garage的PR2机器人使用了一种叫做 pr2_etherCAT 的系统来实时发送或接收 ROS 消息。ROS还可以与Orocos 实时工具包无缝集成。

ROS 图标：

1 ros 的设计目标和特点

很多人都在问“ROS与其它机器人软件平台有什么不同？”这是一个很难解答的问题。因为ROS不是一个集成了大多数功能或特征的框架。事实上，ROS 的主要目标是为机器人研究和开发提供代码复用的支持。ROS是一个分布式的进程（也就是节点）框架，这些进程被封装在易于被分享和发布的程序包和功能包集中。ROS也支持一种类似于代码储存库的联合系统，这个系统也可以实现工程的协作及发布。这个设计可以使一个工程的开发和实现从文件系统到用户接口完全独立决策（不受ROS限制）。同时，所有的工程都可以被ROS的基础工具整合在一起。

为了支持共享和协作这一主要目标，ROS 框架还有其他几个特点：

- 精简：ROS尽可能设计的精简，以便为ROS编写的代码可以与其他机器人软件框架一起使用。由此得出的必然结论是ROS可以轻松集成在其它机器人软件平台：ROS已经可以与OpenRAVE，Orocos和Player集成。
- ROS不敏感库：ROS的首选开发模型都是用不依赖ROS的干净的库函数编写而成。
- 语言独立：ROS框架可以简单地使用任何的现代编程语言实现。ros已经实现了Python版本，C++版本和 Lisp版本。同时也拥有Java 和 Lua版本的实验库。
- 松耦合:ROS中功能模块封装于独立的功能包或元功能包，便于分享，功能包内的模块以节点为单位运行，以ROS标准的IO作为接口，开发者不需要关注模块内部实现，只要了解接口规则就能实现复用,实现了模块间点对点的松耦合连接
- 方便测试：ROS内建一个了叫做rostest的单元/集成测试框架，可以轻松安装或卸载测试模块。
- 可扩展：ROS可以适用于大型运行时系统和大型开发进程。
- 免费且开源：开发者众多，功能包多

2 为什么使用ROS

通过ROS，我们能够在虚拟环境中实现对机械臂的仿真控制。

我们将通过 **rviz** 平台实现对机械臂的可视化，并使用多种方式对我们的机械臂进行操作；

我们将在接下来的章节中学习如何通过ros中的平台对我们产品的控制进行控制。

← 上一节 | 下一页 →

Linux系统环境搭建

系统出厂自带Ubuntu（V-20.04）系统，内置开发环境，无需搭建和管理，更新 `mercury_x1_ros` 包即可。

`mercury_x1_ros` 是大象机器人推出的适用于其Mercury X1系列机械臂的ROS1包。

ROS1项目地址: http://github.com/elephantrobotics/mercury_x1_ros

机械臂API驱动库地址: https://github.com/elephantrobotics/pymycobot

1 更新 mercury_x1_ros 包

为了保证用户能及时使用最新的官方包，可以通过文件管理器进入 `/home/er/` 文件夹，打开ROS1环境终端，然后运行命令更新：

```
# 克隆github上的代码
git clone https://github.com/elephantrobotics/mercury_x1_ros.git # 在决定是否执行此命令
cd ~/mercury_x1_ros     # 回到工作区
catkin_make # 在工作区中构建代码
source devel/setup.bash # 添加环境变量
```

注意: 如果在 `/home/er` 目录下已经存在 `mercury_x1_ros` 文件夹，则需要先删除原来的 `mercury_x1_ros` ，然后再执行上述命令。 其中，目录路径中的 `er` 为系统的用户名。 如有不一致，请修改。

至此ROS1环境搭建完成，您可以学习 ROS的基础知识 或者 ROS使用案例。

← 上一页 | 下一页 →

1 ROS工程结构

1.1 catkin工作空间

Catkin工作空间是创建、修改、编译catkin软件包的目录。catkin的工作空间，直观的形容就是一个仓库，里面装载着ROS的各种项目工程，便于系统组织管理调用。

- **创建工作空间**：

```
mkdir -p ~/catkin_ws/src  # 创建文件夹
cd ~/catkin_ws/src        # 进入文件夹
catkin_init_workspace     # 把当前目录初始化为一个ROS工作空间
cd ..                     # 返回上级目录
catkin_make               # 构建工作区中的代码。
```

catkin的结构十分清晰，它包括了src、build、devel三个路径，在有些编译选项下也可能包括其他。但这三个文件夹是catkin编译系统默认的。它们的具体作用如下：

```
src/: ROS的catkin软件包（源代码包）

build/: catkin（CMake）的缓存信息和中间文件

devel/: 生成的目标文件（包括头文件，动态链接库，静态链接库，可执行文件等）、环境变量
```

一个简单的工作空间如下所示：

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

1.2 ROS软件包

Package不仅是Linux上的软件包，也是catkin编译得基本单元，我们使用catkin_make 编译的对象就是每 个ROS的package。

```
+-- PACKAGE
    +-- CMakeLists.txt
    +-- package.xml
    +-- src/
    +-- include/
    +-- scripts/
    +-- msg/
    +-- srv/
    +-- urdf/
    +-- launch/
```

- **CMakeLists.txt:** 定义package的包名、依赖、源文件、目标文件等编译规则，是package不可少的成 分
- **package.xml:** 描述package的包名、版本号、作者、依赖等信息，是package不可少的成分
- **src/:** 存放ROS的源代码，包括C++的源码和(.cpp)以及Python的module(.py)
- **include/:** 存放C++源码对应的头文件
- **scripts/:** 存放可执行脚本，例如shell脚本(.sh)、Python脚本(.py)
- **msg/:** 存放自定义格式的消息(.msg)
- **srv/:** 存放自定义格式的服务(.srv)
- **urdf/:** 存放机器人的模型描述(.urdf或.xacro)、3D模型文件(.sda, .stl, .dae等)
- **launch/:** 存放launch文件(.launch或.xml)

创建自己的软件包:

- 指令格式:

catkin_create_pkg命令会要求你输入package_name，如有需要还可以在后面添加一些需要依赖的其它软件包:

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

- 例如:

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

2 ROS通信架构

2.1 Master与node

1 Master

节点管理器，每个node启动前要向master注册，管理node之间的通信。

2 roscore

启动master，也会启动rosout（日志管理）和parameter server（参数管理器）

3 node

ROS的进程、pkg里可执行文件运行的实例。

```
$rosrun [pkg_name] [node_name] #启动
$rosnode list  #列出当前运行的node信息
$rosnode info [node_name] #显示某个node的详细信息
$rosnode kill [node_name] #结束某个node
```

4 launch

启动master和多个node。

```
$roslaunch [pkg_name] [file_name.launch]
```

2.2 Service 与 Topic

我们提供一些 service 和 topic， 用以和 mycobot 交互。

1 Service

命令行中输入:

```
source ~/catkin_ws/devel/setup.bash # 添加环境变量
roslaunch mycobot_320_communication communication_service.launch
```

支持参数：

- port：连接串口字符串
- baud：波特率

打开新的命令行：

```
# 显示活动的服务信息
rosservice list

#/get_joint_angles
#/get_joint_coords
#/set_joint_angles
#/set_joint_coords
#/switch_gripper_status
#/switch_pump_status
```

相关的命令与说明：

命令	详细说明
rosservice list	显示活动的服务信息
rosservice info [服务名称]	显示指定服务的信息
rosservice type [服务名称]	显示服务类型
rosservice find [服务类型]	查找指定服务类型的服务
rosservice uri [服务名称]	显示ROSRPC URI服务
rosservice args [服务名称]	显示服务参数
rosservice call [服务名称] [参数]	用输入的参数请求服务

2 Topic

命令行中输入：

```
source ~/catkin_ws/devel/setup.bash
roslaunch mycobot_320_communication communication_topic.launch
```

支持参数：

- port：连接串口字符串
- baud：波特率

打开新的命令行：

```
# 显示活动的服务信息
rostopic list

#/mycobot/angles_goal
#/mycobot/coords_goal
#/mycobot/angles_real
#/mycobot/coords_real
#/mycobot/pump_status
#/mycobot/gripper_status
```

相关的命令与说明：

命令	详细说明
rostopic list	显示活动的话题目录
rostopic echo [话题名称]	实时显示指定话题的消息内容
rostopic find [类型名称]	显示使用指定类型的消息的话题
rostopic type [话题名称]	显示指定话题的消息类型
rostopic bw [话题名称]	显示指定话题的消息带宽（bandwidth）
rostopic hz [话题名称]	显示指定话题的消息数据发布周期
rostopic info [话题名称]	显示指定话题的信息
rostopic pub [话题名称] [消息类型] [参数]	用指定的话题名称发布消息

service与topic的区别：

	service	topic
同步性	异步	同步
通信模型	发布/订阅	服务器/客户端
底层协议	ROSTCP/ROSUDP	ROSTCP/ROSUDP
反馈机制	无	有
缓冲区	有	无
实时性	弱	强
节点关系	多对多	一对多
适用场景	数据传输	逻辑处理

您可以前往service和topic深入了解这两项功能的使用

2.3 msg和srv简介

- msg：msg文件是描述ROS消息字段的简单文本文件。它们用于为不同语言（c++或者python等）的消息生成源代码。
- srv：srv文件用来描述服务。它由两部分组成：请求（request）和响应（response）。 msg文件存储在包的msg目录中，而srv文件存储在srv目录中。

1 rosmsg

rosmsg是用于显示有关 ROS消息类型的 信息的命令行工具。

rosmsg 演示：

```
rosmsg show       # 显示消息描述
rosmsg info       # 显示消息信息
rosmsg list       # 列出所有消息
rosmsg md5        # 显示 md5 加密后的消息
rosmsg package    # 显示某个功能包下的所有消息
rosmsg packages   # 列出包含消息的功能包
```

- rosmsg list 会列出当前 ROS 中的所有 msg
- rosmsg packages 列出包含消息的所有包
- rosmsg package 列出某个包下的所有msg

```
//rosmsg package  # 包名
rosmsg package turtlesim
```

- rosmsg show 显示消息描述

```
//rosmsg show # 消息名称
rosmsg show turtlesim/Pose
# 结果:
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

- rosmsg info 作用与 rosmsg show 一样
- rosmsg md5 一种校验算法，保证数据传输的一致性

2 rossrv

rossrv是用于显示有关ROS服务类型的信息的命令行工具，与 rosmsg 使用语法高度雷同。

```
rossrv show       # 显示服务消息详情
rossrv info       # 显示服务消息相关信息
rossrv list       # 列出所有服务信息
rossrv md5        # 显示 md5 加密后的服务消息
rossrv package    # 显示某个包下所有服务消息
rossrv packages   # 显示包含服务消息的所有包
```

- rossrv list 会列出当前 ROS 中的所有 srv 消息
- rossrv packages 列出包含服务消息的所有包
- rossrv package 列出某个包下的所有msg

```
//rossrv package # 包名
rossrv package turtlesim
```

- rossrv show 显示消息描述

```
//rossrv show # 消息名称
rossrv show turtlesim/Spawn
# 结果:
float32 x
float32 y
float32 theta
string name
---
string name
```

- rossrv info 作用与 rossrv show 一致
- rossrv md5 对 service 数据使用 md5 校验(加密)

3 URDF介绍

- Unified Robot Description Format，统一机器人描述格式，简称为URDF。ROS中的urdf功能包包含一个URDF的C++解析器，URDF文件使用XML格式描述机器人模型。
- URDF 不能单独使用，需要结合 Rviz 或 Gazebo，URDF 只是一个文件，需要在 Rviz 或 Gazebo 中渲染成图形化的机器人模型。

3.1 urdf文件描述

代码示例：

本处只截取部分代码进行展示：

```
<?xml version="1.0"?>
<robot  xmlns:xacro="http://www.ros.org/wiki/xacro" name="firefighter" >

<xacro:property name="width" value=".2" />

  <link name="base_footprint"/>

  <!-- base -->
  <link name="base_link">
    <visual>
      <geometry>
       <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/base.dae"/>
      </geometry>
      <origin xyz = "-0.04 0 0 " rpy = " 1.5708 0 0"/>
    </visual>
    <collision>
      <geometry>
       <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/base.dae"/>
        </geometry>
        <origin xyz = "-0.04 0 0 " rpy = " 1.5708 0 0"/>
    </collision>
  </link>

  <!-- body -->
  <link name="link_body">
    <visual>
      <geometry>
       <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/body.dae"/>
      </geometry>
      <origin xyz = "0 0 -0.11 " rpy = " 0 0 0"/>
    </visual>
    <collision>
     <geometry>
       <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/body.dae"/>
      </geometry>
      <origin xyz = "0 0 -0.11 " rpy = " 0 0 0"/>
    </collision>
  </link>

  <!-- screen head -->
  <link name="link_head">
    <visual>
      <geometry>
       <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/head.dae"/>
      </geometry>
      <origin xyz = "0 0 -0.1 " rpy = " 0 0 0"/>
    </visual>
    <collision>
```

```
    <geometry>
      <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/head.dae"/>
     </geometry>
     <origin xyz = "0 0 -0.1 " rpy = " 0 0 0"/>
  </collision>
</link>

<!-- camera -->
<link name="link_eye">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/head_eye.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.0 " rpy = " 0 0 -0.6981"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/head_eye.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.1 " rpy = " 0 0 -0.6981"/>
  </collision>
</link>

<link name="link1_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint1_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.084 " rpy = " 0 0 0"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint1_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.084 " rpy = " 0 0 0"/>
  </collision>
</link>

<link name="link2_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint2_L.dae"/>
    </geometry>
    <origin xyz = "0 0 0.0 " rpy = " 3.14159 0 -1.5708"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint2_L.dae"/>
```

```
    </geometry>
    <origin xyz = "0 0 0.0 " rpy = " 3.14159 0 -1.5708"/>
  </collision>
</link>

<link name="link3_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint3_L.dae"/>
    </geometry>
    <origin xyz = "0 0 0.038 " rpy = " 0 0 0"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint3_L.dae"/>
    </geometry>
    <origin xyz = "0 0 0.038 " rpy = " 0 0 0"/>
  </collision>
</link>

<link name="link4_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint4_L.dae"/>
    </geometry>
    <origin xyz = "0 0 0 " rpy = " 0 0 -1.5708"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint4_L.dae"/>
    </geometry>
    <origin xyz = "0 0 0 " rpy = " 0 0 -1.5708"/>
  </collision>
</link>

<link name="link5_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint5_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.069 " rpy = " 0 0 0"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint5_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.069 " rpy = " 0 0 0"/>
  </collision>
```

```
</link>

<link name="link6_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint6_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.025 " rpy = " 0 0 -1.5708"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint6_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.025 " rpy = " 0 0 -1.5708"/>
  </collision>
</link>

<link name="link7_L">
  <visual>
    <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint7_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.002 " rpy = " 0 0 -1.5708"/>
  </visual>
  <collision>
   <geometry>
     <mesh filename="package://turn_on_mercury_robot/urdf/mercury_x1/joint7_L.dae"/>
    </geometry>
    <origin xyz = "0 0 -0.002 " rpy = " 0 0 -1.5708"/>
  </collision>
</link>



 <joint name="joint1_L" type="revolute">
  <axis xyz=" 0 0 1"/>
  <limit effort = "1000.0" lower = "-3.1066" upper = "3.1066" velocity = "0"/>
  <parent link="link_body"/>
  <child link="link1_L"/>
  <origin xyz= " 0.075 0.186 0.085   " rpy = "-1.5708 0 -0.5235"/>
</joint>

<joint name="joint2_L" type="revolute">
  <axis xyz=" 0 0 1"/>
  <limit effort = "1000.0" lower = "-1.4311" upper = "2.2689" velocity = "0"/>
  <parent link="link1_L"/>
  <child link="link2_L"/>
  <origin xyz= " 0 0 0  " rpy = "-1.5708 0 0"/>
</joint>
```

```
  <joint name="joint3_L" type="revolute">
    <axis xyz=" 0 0 1"/>
    <limit effort = "1000.0" lower = "-3.1066" upper = "3.1066" velocity = "0"/>
    <parent link="link2_L"/>
    <child link="link3_L"/>
    <origin xyz= " 0 -0.11 0  " rpy = "1.5708 0 0"/>
  </joint>

  <joint name="joint4_L" type="revolute">
    <axis xyz=" 0 0 1"/>
    <limit effort = "1000.0" lower = "-3.0543" upper = "0.2617" velocity = "0"/>
    <parent link="link3_L"/>
    <child link="link4_L"/>
    <origin xyz= " 0.03 0 0.095" rpy = "-1.5708 0 3.14159"/>
  </joint>

  <joint name="joint5_L" type="revolute">
    <axis xyz=" 0 0 1"/>
    <limit effort = "1000.0" lower = "-3.1066" upper = "3.1066" velocity = "0"/>
    <parent link="link4_L"/>
    <child link="link5_L"/>
    <origin xyz= " 0.03 -0.126 0" rpy = "1.5708 3.14159 0"/>
  </joint>

  <joint name="joint6_L" type="revolute">
    <axis xyz=" 0 0 1"/>
    <limit effort = "1000.0" lower = "-0.0349" upper = "2.9321" velocity = "0"/>
    <parent link="link5_L"/>
    <child link="link6_L"/>
    <origin xyz= " 0 0 0.08" rpy = "1.5708 0 0"/>
  </joint>

  <joint name="joint7_L" type="revolute">
    <axis xyz=" 0 0 1"/>
    <limit effort = "1000.0" lower = "-3.1066" upper = "3.1066" velocity = "0"/>
    <parent link="link6_L"/>
    <child link="link7_L"/>
    <origin xyz= " 0.038 -0.046 0" rpy = "1.5708 0 0"/>
  </joint>

</robot>
```

可以看出，urdf文件并不复杂，主要是由 `link` 和 `joint` 两个部分不断重复而成。

3.2 link部分

link元素描述具有惯性、可视特征和碰撞属性的刚体

3.2.1 属性

name： 用来描述链接本身的名称

3.2.2 元素

- `<inertial>` (可选)
  - 连杆的惯性特性
  - `<origin>` (可选，defaults to identity if not specified)
    - 定义相对于连杆坐标系的惯性参考系的参考坐标，该坐标必需定义在连杆重心处，其坐标轴可与惯性主轴不平行。
    - xyz (可选，默认为零向量) 表示 x , y , z x,y,zx,y,z 方向的偏置，单位为米。
    - rpy(可选: defaults to identity if not specified) 表示坐标轴在RPY方向上的旋转，单位为弧度。
  - `<mass>` 连杆的质量属性
  - `<inertia>` 3×3旋转惯性矩阵，由六个独立的量组成：ixx, ixy, ixz, iyy, iyz, izz。
- `<visual>` (可选)
  - 连杆的可视化属性。用于指定连杆显示的形状（矩形、圆柱体等），同一连杆可以存在多个visual元素，连杆的形状为多个元素两个形成。一般情况下模型较为复杂可以通过soildwork绘制后生成stl调用，简单的形状如添加末端执行器等可以直接编写。同时可以在此处可根据理论模型和实际模型差距调整几何形状的位置。
  - `<namel>` (可选) 连杆几何形状的名字。
  - `<origin>` (可选，defaults to identity if not specified)
    - 相对于连杆的坐标系的几何形状坐标系。
    - xyz (optional: defaults to zero vector) 表示x , y , z x,y,zx,y,z 方向的偏置，单位为米。
    - rpy (optional: defaults to identity if not specified) 表示坐标轴在RPY方向上的旋转，单位为弧度。

- `<geometry>` （必需）
  - 可视化对象的形状，可以是下面的其中一种：
  - `<box>` 矩形，元素包含长、宽、高。原点在中心。
  - `<cylinder>` 圆柱体，元素包含半径、长度。原点中心。
  - `<sphere>` 球体，元素包含半径。原点在中心。
  - `<mesh>` 网格，由文件决定，同时提供 scale ，用于界定其边界。推荐使用 Collada .dae 文件， 也支持.stl文件，但必须为一个本地文件。
- `<material>` (可选)
  - 可视化组件的材料。可以在link标签外定义，但必需在robot标签内，在link标签外定义时，需引用link的名字。
  - `<color>` (可选) 颜色，由 red/green/blue/alpha 组成，大小范围在 [0,1] 内。
  - `<texture>` (可选) 材料属性，由文件定义。

- `<collision>` (可选)

- 连杆的碰撞属性。碰撞属性和连杆的可视化属性不同，简单的碰撞模型经常用来简化计算。同一个连杆可以有多个碰撞属性标签，连杆的碰撞属性表示由其定义的几何图形集构成。
- `<name>` (可选) 指定连杆几何形状的名称
- `<origin>` (可选，defaults to identity if not specified)
  - 碰撞组件的参考坐标系相对于连杆坐标系的参考坐标系。
  - xyz (可选， 默认零向量) 表示x , y , z x,y,zx,y,z 方向的偏置，单位为米。
  - rpy (可选， defaults to identity if not specified) 表示坐标轴在RPY方向上的旋转，单位为弧度。
- `<geometry>` 与上述geometry元素描述相同

详细元素以及各个元素的作用可以前往官方文档进行查看

3.3 joint部分

joint部分描述了关节的运动学和动力学，并指定了关节的安全限值。

3.3.1 joint的属性：

name：

指定关节的唯一名称

type：

指定关节的类型，其中类型可以是下列类型之一：

- revolute - 沿轴线旋转的铰链接头，其范围由上限和下限指定。
- 连续 - 一种连续铰链接头，围绕轴旋转，没有上限和下限。
- 棱柱形 - 沿轴滑动的滑动接头，其范围由上限和下限指定。
- 固定 - 这不是真正的关节，因为它不能移动。所有自由度都被锁定。这种类型的接头不需要轴，校准，动力学，极限或safety_controller。
- 浮动 - 此接头允许所有 6 个自由度的运动。
- 平面 - 此接头允许在垂直于轴的平面上运动。

3.3.2 joint的元素

- `<origin>` (可选，defaults to identity if not specified) 从parent link到child link的变换，joint位于child link的原点，修改该参数可以调整连杆的位置，可用在调整实际模型与理论模型误差，但不建议大幅度修改，因为该参数影响连杆stl的位置，容易影响碰撞检测效果。
  - xyz (可选: 默认为零向量) 代表x , y , z x,y,zx,y,z轴方向上的偏移，单位米。
  - rpy (可选: 默认为零向量) 代表绕着固定轴旋转的角度：roll绕着x轴,pitch绕着y轴，yaw绕着z轴，用弧度表示。
- `<parent>` (必需)
  - parent link的名字是一个强制的属性。
  - link parent link的名字，是这个link在机器人结构树中的名字。
- `<child>` (必需)

    - child link的名字是一个强制的属性。
    - link child link的名字，是这个link在机器人结构树中的名字。

- `<axis>` (可选: 默认为(1,0,0))
    - joint的axis轴在joint的坐标系中。这是旋转轴(revolute joint)，prismatic joint移动的轴，是planar joint的标准平面。这个轴在joint坐标系中被指定。修改该参数可以调整关节的旋转所绕着的轴，常用于调整旋转方向，若模型旋向与实际相反，只需乘-1即可。固定(fixed)和浮动(floating)类型的joint不需要用到这个元素。
    - xyz(必需) 代表轴向量的x , y , z x,y,zx,y,z分量，为标准化的向量。
- `<calibration>` (可选)
    - joint的参考点，用来矫正joint的绝对位置。
    - rising (可选) 当joint正向运动时，参考点会触发一个上升沿。
    - falling (可选) 当joint正向运动时，参考点会触发一个下降沿。
- `<dynamics>` (可选)
    - 该元素用来指定joint的物理性能。它的值被用来描述joint的建模性能，尤其是在仿真的时候。

`<limit>` (当关节为旋转或移动关节时为必需)

- 该元素为关节运动学约束。
- lower (可选, 默认为0) 指定joint运动范围下界的属性(revolute joint的单位为弧度，prismatic joint的单位为米)，连续型的joint忽略该属性。
- upper (可选, 默认为0) 指定joint运动范围上界的属性(revolute joint的单位为弧度，prismatic joint的单位为米)，连续型的joint忽略该属性。
- effort (必需) 该属性指定了joint运行时的最大的力。
- velocity (required) 该属性指定了joint运行时的最大的速度。

`<mimic>` (可选)

- 这个标签用于指定已定义的 joint 来模仿已存在的 joint 。这个joint的值可以用以下公式计算： `value = multiplier * other_joint_value + offset`
- joint(必填) 需要模仿的joint的名字。
- multiplier(可选) 指定上述公式中的乘数因子。
- offset(可选) 指定上述公式中的偏移项。默认值为0

`<safety_controller>` (可选)

- 该元素为安全控制限制，此元素下数据会读入到move_group中，但实际上时无效，move_group会跳过此处限制直接读取limit下的参数内容，同时设置该元素有几率会导致规划失败。
- soft_lower_limit (可选, 默认为0) 该属性指定了joint安全控制边界的下界，是joint安全控制的起始限制点。这个值需要大于上述的limit中的lower值。
- soft_upper_limit (可选, 默认为0) 该属性指定了joint安全控制边界的上界，是joint安全控制的起始限制点。这个值需要小于上述的limit中的upper值。
- k_position(可选, 默认为0) 本属性用于说明位置和速度之间的关系。
- k_velocity(必需) 本属性用于说明力和速度之间的关系。

详细元素以及各个元素的作用可以前往 http://wiki.ros.org/urdf/XML/joint 进行查看

4 常用命令工具

在ROS中，有许多常用的命令行工具，这些工具可以帮助你进行开发、调试、管理ROS节点等。以下是一些常用的ROS命令行工具：

4.1 编译工作空间

```
caktin_make
```

4.2 roscore

启动ROS主节点。在运行ROS节点之前，通常需要先启动roscore

```
roscore
```

4.3 rosrun

运行指定的ROS节点。

```
rosrun package_name node_name
```

4.4 roslaunch

使用Launch文件启动一个或多个ROS节点。

```
roslaunch package_name launch_file.launch
```

4.5 rosnode

查看正在运行的ROS节点信息。

```
rosnode list
rosnode info node_name
```

4.6 rostopic

查看正在运行的ROS话题信息。

```
rostopic list
rostopic echo topic_name
```

4.7 rosservice

查看和调用ROS服务。

```
rosservice list
rosservice call service_name
```

4.8 rosparam

获取和设置ROS参数。

```
rosparam get parameter_name
rosparam set parameter_name value
```

4.9 rosmsg

查看ROS消息类型。

```
rosmsg show message_type
```

4.10 rosdep

安装ROS包的依赖项。

```
rosdep install package_name
```

4.11 环境变量

查看ROS_PACKAGE_PATH环境变量

```
echo $ROS_PACKAGE_PATH
```

← 上一页 | 下一页 →

rviz的简单介绍及使用

rviz是ROS中一款三维可视化平台，一方面能够实现对外部信息的图形化显示，另外还可以通过 rviz 给对象发布控制信息，从而实现对机器人的监测与控制。

1 rviz界面简介

请先打开一个新的ROS1环境终端(快捷键`Ctrl+Alt+T`),输入如下指令：

```
roscore
```

然后再打开一个ROS1环境终端(快捷键`Ctrl+Alt+T`)输入命令打开rviz

```
rosrun rviz rviz
# 或
rviz
```

打开rviz,显示如下界面：

1.1 各个区域介绍

- 左侧为显示器列表，显示器是在3D世界中绘制某些内容的东西，并且可能在显示列表中具有一些可用的选项。
- 上方是工具栏，允许用户用各种功能按键选择多种功能的工具
- 中间部分为3D视图: 它是可以用三维方式查看各种数据的主屏幕。3D视图的背景颜色、固定框架、网格等可以在左侧显示的全局选项（Global Options）和网格（Grid）项目中进行详细设置。
- 下方为时间显示区域，包括系统时间和ROS时间等。
- 右侧为观测视角设置区域，可以设置不同的观测视角。

本部分我们只进行粗略的介绍，如果您想了解更多详细的内容，可以前往用户指南进行查看。

如果您想了解更多rviz的相关资料信息，您可以前往官方文档进行查看

← 上一页 | 下一页 →

Mercury X1的控制

这里主要介绍如何通过一系列相关指令对Mercury X1进行移动控制。

1.底盘底层通信

首先启动底盘的底层通信、地图构建程序。 打开ROS1环境终端，然后运行命令：

```
roslaunch turn_on_mercury_robot mapping.launch
```

输出如下信息：

```
... logging to /home/er/.ros/log/34bcf3be-0606-11ef-8293-e8fb1c355a09/roslaunch-er-des
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://er-desktop:43117/

SUMMARY
========

PARAMETERS
 * /LD14/disable_max: [180]
 * /LD14/disable_min: [120]
 * /LD14/flag_parted: True
 * /if_akm_yes_or_no: no
 * /lsm10_v2/baud_rate: 460800
 * /lsm10_v2/disable_max: [260]
 * /lsm10_v2/disable_min: [100]
 * /lsm10_v2/frame_id: laser
 * /lsm10_v2/max_distance: 30.0
 * /lsm10_v2/min_distance: 0.0
 * /lsm10_v2/scan_topic: scan
 * /lsm10_v2/serial_port: /dev/wheeltec_lidar
 * /lsm10_v2/truncated_mode: 1
 * /lsn10/disable_max: [10, 40, 70, 180]
 * /lsn10/disable_min: [0, 30, 60, 90]
 * /lsn10/max_distance: 30.0
 * /lsn10/min_distance: 0.0
 * /lsn10/truncated_mode: 0
 * /mercury_robot/gyro_frame_id: gyro_link
 * /mercury_robot/odom_frame_id: odom
 * /mercury_robot/robot_frame_id: base_footprint
 * /mercury_robot/serial_baud_rate: 115200
 * /mercury_robot/usart_port_name: /dev/wheeltec_con...
 * /robot_description: <?xml version="1....
 * /robot_pose_ekf/base_footprint_frame: base_footprint
 * /robot_pose_ekf/freq: 30.0
 * /robot_pose_ekf/imu_used: True
 * /robot_pose_ekf/odom_data: odom
 * /robot_pose_ekf/odom_used: True
 * /robot_pose_ekf/output_frame: odom
 * /robot_pose_ekf/sensor_timeout: 2.0
 * /robot_pose_ekf/vo_used: False
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /rplidarNode/angle1_end: 50.0
 * /rplidarNode/angle1_start: 40.0
 * /rplidarNode/angle2_end: 140.0
```

```
 * /rplidarNode/angle2_start: 130.0
 * /rplidarNode/angle3_end: 230.0
 * /rplidarNode/angle3_start: 220.0
 * /rplidarNode/angle4_end: 320.0
 * /rplidarNode/angle4_start: 310.0
 * /rplidarNode/angle_end: 360.0
 * /rplidarNode/angle_start: 0.0
 * /rplidarNode/distance_max: 30.0
 * /rplidarNode/distance_min: 0.0
 * /rplidarNode/is_parted: False
 * /slam_gmapping/angularUpdate: 0.0436
 * /slam_gmapping/astep: 0.05
 * /slam_gmapping/base_frame: base_footprint
 * /slam_gmapping/delta: 0.05
 * /slam_gmapping/iterations: 5
 * /slam_gmapping/kernelSize: 3
 * /slam_gmapping/lasamplerange: 0.005
 * /slam_gmapping/lasamplestep: 0.005
 * /slam_gmapping/linearUpdate: 0.05
 * /slam_gmapping/llsamplerange: 0.01
 * /slam_gmapping/llsamplestep: 0.01
 * /slam_gmapping/lsigma: 0.075
 * /slam_gmapping/lskip: 0
 * /slam_gmapping/lstep: 0.05
 * /slam_gmapping/map_update_interval: 0.01
 * /slam_gmapping/maxRange: 5.0
 * /slam_gmapping/maxUrange: 4.0
 * /slam_gmapping/minimumScore: 30
 * /slam_gmapping/odom_frame: odom
 * /slam_gmapping/ogain: 3.0
 * /slam_gmapping/particles: 8
 * /slam_gmapping/resampleThreshold: 0.5
 * /slam_gmapping/sigma: 0.05
 * /slam_gmapping/srr: 0.01
 * /slam_gmapping/srt: 0.02
 * /slam_gmapping/str: 0.01
 * /slam_gmapping/stt: 0.02
 * /slam_gmapping/temporalUpdate: -1.0
 * /slam_gmapping/xmax: 5.0
 * /slam_gmapping/xmin: -5.0
 * /slam_gmapping/ymax: 4.0
 * /slam_gmapping/ymin: -4.0

NODES
  /
    base_to_camera (tf/static_transform_publisher)
    base_to_gyro (tf/static_transform_publisher)
    base_to_laser (tf/static_transform_publisher)
```

```
    base_to_link (tf/static_transform_publisher)
    joint_state_publisher (joint_state_publisher/joint_state_publisher)
    lsm10_v2 (lsm10_v2/lsm10_v2)
    mercury_robot (turn_on_mercury_robot/mercury_robot_node)
    robot_pose_ekf (robot_pose_ekf/robot_pose_ekf)
    robot_state_publisher (robot_state_publisher/robot_state_publisher)
    save_map (world_canvas_msgs/save)
    slam_gmapping (gmapping/slam_gmapping)

auto-starting new master
process[master]: started with pid [7403]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 34bcf3be-0606-11ef-8293-e8fb1c355a09
process[rosout-1]: started with pid [7414]
started core service [/rosout]
process[lsm10_v2-2]: started with pid [7421]
process[save_map-3]: started with pid [7422]
process[slam_gmapping-4]: started with pid [7427]
port = /dev/wheeltec_lidar, baud_rate = 460800
open_port /dev/wheeltec_lidar  OK !
process[mercury_robot-5]: started with pid [7434]
process[base_to_link-6]: started with pid [7437]
process[base_to_laser-7]: started with pid [7447]
[ INFO] [1714380923.257953390]: Data ready
[ INFO] [1714380923.271190031]: tringai_robot serial port opened
process[base_to_camera-8]: started with pid [7455]
process[base_to_gyro-9]: started with pid [7461]
process[joint_state_publisher-10]: started with pid [7464]
process[robot_state_publisher-11]: started with pid [7476]
process[robot_pose_ekf-12]: started with pid [7488]
[ INFO] [1714380923.815804921]: output frame: odom
[ INFO] [1714380923.822399818]: base frame: base_footprint
[ INFO] [1714380924.077547181]: Initializing Odom sensor
[ INFO] [1714380924.582680406]: Odom sensor activated
[ INFO] [1714380925.085858055]: Initializing Imu sensor
[ INFO] [1714380925.087556508]: Kalman filter initialized with odom measurement
[ INFO] [1714380925.091287721]: Imu sensor activated
[ INFO] [1714380925.206262818]: Laser is mounted upwards.
 -maxUrange 4 -maxUrange 5 -sigma     0.05 -kernelSize 3 -lstep 0.05 -lobsGain 3 -aste
 -srr 0.01 -srt 0.02 -str 0.01 -stt 0.02
 -linearUpdate 0.05 -angularUpdate 0.0436 -resampleThreshold 0.5
 -xmin -5 -xmax 5 -ymin -4 -ymax 4 -delta 0.05 -particles 8
[ INFO] [1714380925.222595401]: Initialization complete
update frame 0
update ld=0 ad=0
Laser Pose= 0.0368556 0.000564671 3.14
m_count 0
```

```
Registering First Scan
update frame 4089
update ld=0.0500112 ad=4.53872e-06
Laser Pose= 0.0383474 0.001601 3.14
```

2.加载URDF模型

底层通信程序启动后，接着通过launch文件加载Mercury X1的URDF模型，打开一个ROS1环境终端，然后运行命令。

```
roslaunch turn_on_mercury_robot slider_control.launch
```

效果图如下：

接着你可以通过拖动滑块来控制 rviz 中的关节模型移动。

3.关节控制

URDF模型加载成功之后，如果你想让真实的 Mercury X1 跟着一起运动，需要再打开一个ROS1环境终端，然后运行命令：

```
rosrun turn_on_mercury_robot slider_control.py
```

请注意：由于在命令输入的同时机械臂会移动到模型目前的位置，在您使用命令之前请确保rviz中的模型没有出现穿模现象 不要在连接机械臂后做出快速拖动滑块的行为，防止机械臂损坏

4 底盘控制

关节控制程序启动之后，如果你想让真实的Mercury X1中的底盘小车跟着一起运行，需要启动底盘小车的键盘控制程序，再打开一个ROS1环境终端，然后运行命令：

```
rosrun turn_on_mercury_robot mercury_keyboard.py
```

ROS2 介绍

ROS2的前身是ROS，ROS就是机器人操作系统（Robot Operating System）。 但ROS本身并不是一个操作系统，而是一个软件库和工具集。 Ros的出现解决了机器人各个部件的通信问题。 后来，越来越多的机器人算法被集成到ROS中。 ROS2继承了ROS，比ROS更强大更好用。

1 ROS2的设计目标和特点

ROS2肩负着改变智能机器人时代的历史使命。 在设计之初，就考虑到满足各种机器人应用的需求。

- **多机器人系统：** 未来机器人不再是独立的个体，机器人之间也需要交流和协作。 ROS2为多机器人系统的应用提供了标准的方法和通信机制。
- **跨平台：** 机器人应用场景不同，使用的控制平台也会有很大差异。 为了让所有的机器人都能运行ROS2，ROS2可以跨平台运行在Linux、Windows、MacOS、RTOS上。
- **实时：** 机器人运动控制和许多行为策略都要求机器人是实时的。 例如，机器人必须在 100 毫秒内可靠地检测到前方的行人，或在 1 毫秒内完成运动学和动力学计算。 ROS2 是像这样实时提供基本要求的。
- **产品化：** 大量的机器人已经进入我们的生活，未来还会越来越多，ROS2不仅可以用于机器人研发阶段，还可以直接安装在 产品并进入消费市场。 这也对ROS2的稳定性和鲁棒性提出了巨大的挑战。
- **项目管理：** 机器人开发是一项复杂的系统工程。 设计、开发、调试、测试、部署全过程的项目管理工具和机制也将在ROS2中得到体现，方便我们开发机器人。

2 发行版本

ROS2和Ubuntu对应的发行版本和维护周期。

ROS2 版本	发布日期	维护期限	Ubuntu 版本
Dashing	2019.5	2021.5	Ubuntu 18.04 (Bionic Beaver)
Eloquent	2019.11	2020.11	Ubuntu 18.04 (Bionic Beaver)
Foxy	2020.6	2023.5	Ubuntu 20.04(Focal Fossa)
Galactic	2021.5	2022.11	Ubuntu 20.04(Focal Fossa)
Humble	2022.5	2027.5	Ubuntu 22.04(Jammy Jellyfish)

3 ROS和ROS2的比较

ROS2重新设计了系统架构。 两代ROS的架构变化如下：

- **OS Layer：** OS层。在ROS2中，它可以构建在linux或其他系统上，甚至是没有操作系统的裸机。
- **Middleware Layer：** 中间件层。ROS1的通信系统基于TCPROS/UDPROS，而ROS2的通信系统基于DDS。 DDS是分布式实时系统中数据发布/订阅的标准解决方案。
- **Application Layer：** 应用层。ROS1依赖于ROS Master，而在ROS2中，节点之间使用了一种名为“Discovery”的发现机制来帮助彼此建立连接。

ROS设计了一套完整的通信机制（主题、服务、参数、动作）来简化机器人开发。通过这种机制，可以连接机器人的各个部件。 这种机制设计了一个叫做Ros Master的节点，所有其他组件的通信都必须经过master节点。 一旦主节点挂掉，就会导致整个机器人系统的通信崩溃！ 所以不能利用Ros的不稳定性来做一些自动驾驶等高风险的机器人。 此外，还有以下缺点：

- 基于TCP的通信实时性差，系统开销大
- 对 python3 支持不友好
- 消息机制不兼容
- 无加密机制，安全性低

ROS2首先移除ROS中存在的master节点。 去掉主节点后，各个节点可以通过DDS节点相互发现，各个节点是平等的，可以实现一对一、一对多、多对多的通信。 使用DDS进行通信后，可靠性和稳定性得到了增强。

与只支持Linux系统的**ROS**相比，**ROS2**还支持windows、mac甚至RTOS平台。

← 上一节 | 下一页 →

Linux系统环境:

系统出厂自带Ubuntu（V-20.04）系统，内置 **ROS2 Galactic** 开发环境，无需搭建和管理，只需更新 `mercury_x1_ros2` 包即可。

`mercury_x1_ros2` 是大象机器人推出的适用于其Mercury X1系列机械臂的ROS2包

ROS2项目地址: http://github.com/elephantrobotics/mercury_x1_ros2

机械臂API驱动库地址: https://github.com/elephantrobotics/pymycobot

1 更新 mercury_x1_ros2 包

为了保证用户能及时使用最新的官方包，可以通过文件管理器进入 `/home/er` 文件夹，打开ROS2环境终端，然后运行命令更新：

```
# 克隆github上的代码
git clone --depth 1 https://github.com/elephantrobotics/mercury_x1_ros2.git
cd ~/mercury_x1_ros2       # 返回工作区
colcon build --symlink-install # 构建工作区中的代码，--symlink-install：避免每次调整 pyth
source install/setup.bash # 添加环境变量

# 单独编译功能包：
# 如果只编译“turn_on_mercury_robot”，则需要执行命令：
colcon build --packages-select turn_on_mercury_robot
```

注意: 如果在 `/home/er` 目录下已经存在 `mercury_x1_ros2` 文件夹，则需要先删除原来的 `mercury_x1_ros2` ，然后再执行上述命令。 其中，目录路径中的 `er` 为系统的用户名。 如有不一致，请修改。

至此ROS2环境搭建完成，你可以学习ROS2的基础知识 或者ROS2使用案例。

← 上一页 | 下一页 →

1 ROS2工程结构

1.1 colcon工作空间

colocn工作空间是创建、修改、编译软件包的目录。colcon的工作空间，直观的形容就是一个仓库，里面装载着ROS的各种项目工程，便于系统组织管理调用。

- **创建工作空间**：

```
mkdir -p ~/colcon_ws/src  # 创建文件夹
cd ~/colcon_ws/        # 进入文件夹
colcon build               # 构建工作区中的代码。
```

注意： colcon 支持选项 `--symlink-install` 。这允许通过更改 source 空间中的文件（例如 Python 文件或其他未编译的资源）来更改已安装的文件，以加快迭代速度。避免每次修改 python 脚本时都需要重新编译。

```
colcon build --symlink-install
```

ROS2工作空间是一个具有特定结构的目录。通常有一个 `src` 子目录。在该子目录中是 ROS2 包的源代码所在的位置。通常，目录以其他方式为空开始。

colcon 会进行源代码构建。默认情况下，它将创建以下目录作为 `src` 目录的同级目录：

```
src/: ROS2的colcon软件包（源代码包）

build/: 存储中间文件的位置。对于每个包，将创建一个子文件夹，例如在其中调用 CMake。

install/: 每个包的安装位置。默认情况下，每个包都将安装到单独的子目录中。

log/: 包含有关每个 colcon 调用的各种日志记录信息。
```

一个ROS2工作空间目录结构如下所示：

```
WorkSpace --- 自定义的工作空间。
    |--- build: 存储中间文件的目录，该目录下会为每一个功能包创建一个单独子目录。
    |--- install: 安装目录，该目录下会为每一个功能包创建一个单独子目录。
    |--- log: 日志目录，用于存储日志文件。
    |--- src: 用于存储功能包源码的目录。
        |-- C++功能包
            |-- package.xml: 包信息，比如:包名、版本、作者、依赖项。
            |-- CMakeLists.txt: 配置编译规则，比如源文件、依赖项、目标文件。
            |-- src: C++源文件目录。
            |-- include: 头文件目录。
            |-- msg: 消息接口文件目录。
            |-- srv: 服务接口文件目录。
            |-- action: 动作接口文件目录。
        |-- Python功能包
            |-- package.xml: 包信息，比如:包名、版本、作者、依赖项。
            |-- setup.py: 与C++功能包的CMakeLists.txt类似。
            |-- setup.cfg: 功能包基本配置文件。
            |-- resource: 资源目录。
            |-- test: 存储测试相关文件。
            |-- 功能包同名目录: Python源文件目录。
```

1.2 ROS2软件包

Package不仅是Linux上的软件包，也是colcon编译得基本单元，我们使用 `colcon build` 编译的对象就是每个ROS2的package。

创建自己的软件包:

- 使用Python创建软件包的命令语法为:

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

- 例如:

```
ros2 pkg create --build-type ament_python --node-name my_node my_package
```

2 基本工具命令

在本章中，您将了解ROS2的常用命令工具。

2.1 Topics

ROS 2 将复杂的系统分解为许多模块化节点。 Topics是 ROS 图的重要元素，充当节点交换消息的总线。 Topics是数据在节点之间移动的主要方式之一，因此在系统的不同部分之间移动。

具体参考: 官方教程

- **topics 帮助**

```
ros2 topics -h
```

- **启动turtlesim和键盘控制**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **节点关系图**

```
rqt_graph
```

- **了解topics相关命令**

```
ros2 topics -h
```

- **话题列表**

```
ros2 topic list
ros2 topic list -t # 显示相应的消息类型
```

- **查看话题内容**

```
ros2 topic echo <topic_name>
ros2 topic echo /turtle1/cmd_vel
```

- **显示话题相关信息，类型**

```
ros2 topic info <topic_name>
# 输出 /turtle1/cmd_vel 话题接口相关信息
ros2 topic info /turtle1/cmd_vel
```

- **显示接口相关信息**

```
ros2 interface show <msg_type>
# 输出 geometry_msgs/msg/Twist接口相关信息
ros2 interface show geometry_msgs/msg/Twist
```

- **发布命令**

```
ros2 topic pub <topic_name> <msg_type> '<args>'
# 发布速度命令
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0
# 按一定频率发布速度命令
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y:
```

- **查看话题发布的频率**

```
ros2 topic hz <topic_name>
#输出/turtle1/cmd_vel发布频率
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y:
```

2.2 Nodes

ROS 中的每个节点都应该负责一个单一的模块用途（例如，一个节点用于控制车轮电机，一个节点用于控制激光测距仪等）。 每个节点都可以通过主题、服务、操作或参数向其他节点发送和接收数据。 一个完整的机器人系统由许多协同工作的节点组成。 在 ROS 2 中，单个可执行文件（C++ 程序、Python 程序等）可以包含一个或多个节点。

具体参考: 官方教程

- **nodes 帮助**

```
ros2 nodes -h
```

- **启动turtlesim和键盘控制**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **查看节点列表**

```
ros2 node list
```

- **查看节点关系图**

```
rqt_graph
```

- **重映射**

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle
ros2 node list
```

- **查看节点信息**

```
ros2 node info <node_name>
ros2 node info /my_turtle
```

2.3 Services

服务是 ROS 图中节点的另一种通信方法。 服务基于调用和响应模型，而不是主题的发布者-订阅者模型。 虽然主题允许节点订阅数据流并获得持续更新，但服务仅在客户端专门调用时才提供数据。

具体参考: 官方教程

- **services 帮助**

```
ros2 service -h
```

- **启动turtlesim和键盘控制**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **查看服务列表**

```
ros2 service list
# 显示服务列表及消息类型
ros2 service list -t
```

- **查看服务接收到的消息类型**

```
ros2 service type <service_name>
ros2 service type /clear
```

- **找到使用某类消息类型的服务**

```
ros2 service find <type_name>
ros2 service find std_srvs/srv/Empty
```

- **查看服务消息类型定义**

```
ros2 interface show <type_name>.srv
ros2 interface show std_srvs/srv/Empty.srv
```

- **调用服务命令,清除行走轨迹**

```
ros2 service call <service_name> <service_type>
ros2 service call /clear std_srvs/srv/Empty
```

- **生成新乌龟**

```
ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: 'turtle2'
```

2.4 Parameters

参数是节点的配置值。 您可以将参数视为节点设置。 节点可以将参数存储为整数、浮点数、布尔值、字符串和列表。 在 ROS 2 中，每个节点都维护自己的参数。 有关参数的更多背景信息，请参阅概念文档。

具体参考: 官方教程

- **parameters 帮助**

```
ros2 param -h
```

- **启动turtlesim和键盘控制**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **查看服务列表**

```
ros2 param list
```

- **获取参数值**

```
ros2 param get <node_name> <parameter_name>
ros2 param get /turtlesim background_g
```

- **设置参数值**

```
ros2 param set <node_name> <parameter_name> <value>
ros2 param set /turtlesim background_r 150
```

- **导出参数值**

```
ros2 param dump <node_name>
ros2 param dump /turtlesim
```

- **独立导入参数**

```
ros2 param load <node_name> <parameter_file>
ros2 param load /turtlesim ./turtlesim.yaml
```

- **启动节点同时导入参数**

```
ros2 run <package_name> <executable_name> --ros-args --params-file <file_name>
ros2 run turtlesim turtlesim_node --ros-args --params-file ./turtlesim.yaml
```

2.5 Actions

动作是 ROS 2 中的一种通信类型，用于长时间运行的任务。 它们由三部分组成：目标、反馈和结果。

操作基于主题和服务。 它们的功能类似于服务，除了操作是可抢占的（您可以在执行时取消它们）。 他们还提供稳定的反馈，而不是返回单一响应的服务。

操作使用客户端-服务器模型，类似于发布者-订阅者模型（在主题教程中描述）。“动作客户端”节点将目标发送到“动作服务器”节点，该节点确认目标并返回反馈流和结果。

具体参考: 官方教程

- **action 帮助**

```
ros2 action -h
```

- **启动turtlesim和键盘控制**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

按G|B|V|C|D|E|R|T 实现旋转，按F键盘取消l

- **查看节点action的服务端和客户端**

```
ros2 node info /turtlesim
```

- **查看动作列表**

```
ros2 action list
ros2 action list -t # 显示动作类型
```

- **查看动作信息**

```
ros2 action info <action>
ros2 action info /turtle1/rotate_absolute
```

- **查看动作消息内容**

```
ros2 interface show turtlesim/action/RotateAbsolute
```

- **发送动作目标信息**

```
ros2 action send_goal <action_name> <action_type>
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta
# 带反馈信息
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta
```

2.6 RQt

RQt 是一个图形用户界面框架，它以插件的形式实现各种工具和界面。 可以将所有现有的 GUI 工具作为 RQt 中的可停靠窗口运行！ 这些工具仍然可以以传统的独立方式运行，但 RQt 可以更轻松地在单个屏幕布局中管理所有不同的窗口。

具体参考: 官方教程

您可以通过以下方式轻松运行任何 RQt 工具/插件：

```
rqt
```

- **rqt 帮助**

```
rqt -h
```

- **启动turtlesim和键盘控制**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- 动作浏览器: / Plugins -> Actions ->Action Type Browser
- 参数重配置: / Plugins -> configuration ->Parameter Reconfigure
- 节点图: /Node Graph
- 控制转向: /Plugins -> Robot Tools -> Robot Steering
- 服务调用: /Plugins -> Services -> Service Caller
- 服务类型浏览器: Plugins -> Services -> Service Type Browser
- 消息发布: Plugins -> Topics -> Message Publisher
- 消息类型浏览器: Plugins -> Topics -> Message Type Browser
- 话题列表: Plugins -> Topics -> Topic Monitor

- 绘制曲线图: Plugins -> Visualization -> Plot
- **查看日志: rqt_console**

```
ros2 run rqt_console rqt_console
ros2 run turtlesim turtlesim_node
ros2 topic pub -r 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0
```

2.7 TF2

tf2 是变换库，它允许用户随着时间的推移跟踪多个坐标系。 tf2 以时间缓冲的树结构维护坐标系之间的关系，并让用户在任何需要的时间点在任意两个坐标系之间变换点、向量等。

具体参考: 官方教程

让我们从安装演示包及其依赖项开始。

```
sudo apt-get install ros-foxy-turtle-tf2-py ros-foxy-tf2-tools ros-foxy-tf-transformat
```

- **跟随**
- launch启动2个小乌龟，第一个小乌龟自动跟随第二个

```
ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py
```

- 通过键盘控制第一个小乌龟移动

```
ros2 run turtlesim turtle_teleop_key
```

- **查看TF树**

```
ros2 run tf2_tools view_frames.py
evince frames.pdf
```

- **查看两个坐标系之间的关系**

```
ros2 run tf2_ros tf2_echo [reference_frame] [target_frame]
ros2 run tf2_ros tf2_echo turtle2 turtle1
```

- **在rviz上查看TF关系**

```
ros2 run rviz2 rviz2 -d $(ros2 pkg prefix --share turtle_tf2_py)/rviz/turtle_rviz.rviz
```

2.8 URDF

URDF 是统一机器人描述格式，用于指定 ROS 中的机器人几何和组织。

具体参考: 官方教程

- **完整语法**

```
<robot>
    # describe:
    # Parameters: name=""
    #  Child node:
        <link>
            # Description:
            # Parameters: name=""
            # Child node:
                <visual>
                    # describe:
                    # Parameters:
                    # child nodes:
                        <geometry>
                            # description
                            # parameters
                            # Child node:
                                <cylinder />
                                    # Description:
                                    # Parameters:
                                        # length="0.6"
                                        # radius="0.2"
                                <box />
                                    # description
                                    # Parameters:size="0.6 0.1 0.2"
                                <mesh />
                                    #  Description
                                    #Parameters: filename="package://urdf_tutorial/mes
                        <collision>
                            # Description: collision element, prioritized
                            # parameters
                            # child node
                                <geometry>
                        <inertial>
                            # description
                            # parameters
                            # Child nodes:
                                <mass />
                                    # description: mass
                                    # Parameters: value=10
                                <inertia />
                                    # Description: Inertia
                                    # Parameters: i+"Cartesian product of xyz" (9 in t
                        <origin />
                            # Description:
                            # Parameters:
                                # rpy="0 1.5 0"
                                # xyz="0 0 -0.3"
                        <material />
```

```
                        # Description
                        # Parameters: name="blue"
        <joint>
            # Description
            # Parameters:
                # name=""
                # type=""
                    # fixed
                    # prismatic
            # child node
                <parent />
                    # Description
                    # Parameters: link=""
                <child />
                    # Description:
                    # Parameters: link=""
                <origin />
                    # Description:
                    # Parameters: xyz="0 -0.2 0.25"
                <limit />
                    # Description
                    # Parameters:
                        # effort="1000.0"    maximum effort
                        # lower="-0.38"      Joint upper limit (radians)
                        # upper="0"          Joint lower limit (radians)
                        # velocity="0.5"     Maximum velocity
                <axis />
                    # Description:  Press ? axis rotation
                    # Parameters: xyz="0 0 1", along the Z axis
        <material>
            # Description:
            # Parameters: name="blue"
            # child node:
                <color />
                    # description:
                    # Parameters: rgba="0 0 0.8 1"
```

- **安装依赖库**

```
sudo apt install ros-foxy-joint-state-publisher-gui ros-foxy-joint-state-publisher
sudo apt install ros-foxy-xacro
```

- **下载源代码**

```
cd ~/dev_ws
git clone -b ros2 https://github.com/ros/urdf_tutorial.git src/urdf_tutorial
```

- **编译源代码**

```
colcon build --packages-select urdf_tutorial
```

- **运行示例**

```
ros2 launch urdf_tutorial display.launch.py model:=urdf/01-myfirst.urdf
```

2.9 Launch

ROS 2 中的启动系统负责帮助用户描述他们系统的配置，然后按照描述执行。 系统的配置包括要运行的程序、运行它们的位置、传递给它们的参数，以及 ROS 特定的约定，这些约定通过为每个组件提供不同的配置，使得在整个系统中重用组件变得容易。 它还负责监视已启动流程的状态，并报告和/或响应这些流程状态的变化。

用 Python、XML 或 YAML 编写的launch文件可以启动和停止不同的节点，以及触发和处理各种事件。

具体参考: 官方教程

Setup

创建一个新目录来存储您的launch文件：

```
mkdir launch
```

编写启动文件

让我们使用 turtlesim 包及其可执行文件将 ROS 2 启动文件放在一起。 正如刚才提到的。

将完整代码复制并粘贴到 launch/turtlesim_mimic_launch.py 文件中：

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

运行ros2启动文件

要运行上面创建的launch文件，请进入您之前创建的目录并运行以下命令：

语法格式为：

```
ros2 launch <package_name> <launch_file_name>
```

```
cd launch
ros2 launch turtlesim_mimic_launch.py
```

- **launch 帮助**

```
ros2 launch -h
```

- **运行节点**

```
ros2 launch turtlesim multisim.launch.py
```

- **查看launch文件有哪些参数**

```
ros2 launch turtlebot3_fake_node turtlebot3_fake_node.launch.py -s
ros2 launch turtlebot3_fake_node turtlebot3_fake_node.launch.py --show-arguments
ros2 launch turtlebot3_bringup robot.launch.launch.py -s
```

- **运行launch文件带参数**

```
ros2 launch turtlebot3_bringup robot.launch.launch.py usb_port:=/dev/opencr
```

- **运行节点并调试**

```
ros2 launch turtlesim turtlesim_node.launch.py -d
```

- **只输出节点描述**

```
ros2 launch turtlesim turtlesim_node.launch.py -p
```

- **运行组件**

```
ros2 launch composition composition_demo.launch.py
```

2.10 Run

run用于运行单个节点，组件程序。

- **run 帮助**

```
ros2 run -h
```

- **运行节点**

```
ros2 run turtlesim turtlesim_node
```

- **运行节点带参数**

```
ros2 run turtlesim turtlesim_node --ros-args -r __node:=turtle2 -r __ns:=/ns2
```

- **运行组件容器**

```
ros2 run rclcpp_components component_container
```

- **运行组件**

```
ros2 run composition manual_composition
```

2.11 Package

一个包可以被认为是你的 ROS 2 代码的容器。 如果您希望能够安装您的代码或与他人共享，那么您需要将其组织在一个包中。 借助软件包，您可以发布您的 ROS 2 作品并允许其他人轻松构建和使用它。

ROS 2 中的包创建使用 ament 作为其构建系统，并使用 colcon 作为其构建工具。 您可以使用官方支持的 CMake 或 Python 创建包，但确实存在其他构建类型。

具体参数: 官方教程

创建工作空间

为每个新工作区创建一个新目录。 名称并不重要，但它有助于表明工作区的用途。 让我们为“开发工作区”选择目录名称 ros2_ws：

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
```

- **pkg 帮助**

```
ros2 pkg -h
```

- **列出功能包**

```
ros2 pkg executable turtlesim
```

- **输出某个功能包可执行程序**

```
ros2 pkg executable turtlesim
```

- **创建Python功能包**

在运行包创建命令之前，请确保您位于 src 文件夹中。

```
cd ~/ros2_ws/src
```

在 ROS 2 中创建新包的命令语法是：

```
ros2 pkg create --build-type ament_python <package_name>
# 您将使用可选参数 --node-name 在包中创建一个简单的 Hello World 类型可执行文件。
ros2 pkg create --build-type ament_python --node-name my_node my_package
```

- **构建package**

将包放在工作区中特别有价值，因为您可以通过在工作区根目录中运行 colcon build 来一次构建许多包。 否则，您将不得不单独构建每个包。

```
# 返回到工作区目录:
cd ~/ros2_ws
# 现在你可以构建你的包:
colcon build
```

- **Source setup文件**

要使用您的新包和可执行文件，首先打开一个新终端并获取您的主要 ROS 2 安装源。

然后，从 ros2_ws 目录中，运行以下命令来获取您的工作空间：

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

现在您的工作区已添加到您的路径中，您将能够使用新包的可执行文件。

- **使用 package**

要运行您在包创建期间使用 --node-name 参数创建的可执行文件，请输入以下命令：

```
ros2 run my_package my_node
```

← 上一页 | 下一页 →

rviz2的简单介绍及使用

Rviz2是一个可视化工具，用于显示机器人环境中的消息，提供3D视角来查看机器人的状态和活动。它可以帮助开发者更好地理解机器人当前的状态和活动，以及其他可视化消息。Rviz2提供了一系列的可视化工具，可以帮助开发者更好地理解机器人的状态和活动，比如可视化坐标系、激光扫描消息、点云消息、机器人模型等等。使用Rviz2，可以轻松地查看和调试机器人系统，从而更好地实现机器人目标。

1 rviz2的界面简介

打开ROS2环境终端，输入命令打开rviz2：

```
ros2 run rviz2 rviz2
# 或
rviz2
```

打开rviz2,显示如下界面：

1.1 各个区域介绍

- 左侧为显示器列表，显示器是在3D世界中绘制某些内容的东西，并且可能在显示列表中具有一些可用的选项。包括添加、删除、复制、重命名插件，显示插件，以及设置插件属性等功能。
- 上方是工具栏，允许用户用各种功能按键选择多种功能的工具
- 中间部分为3D视图: 它是可以用三维方式查看各种数据的主屏幕。3D视图的背景颜色、固定框架、网格等可以在左侧显示的全局选项（Global Options）和网格（Grid）项目中进行详细设置。
- 下方为时间显示区域，包括系统时间和ROS时间等。
- 右侧为观测视角设置区域，可以设置不同的观测视角。

本部分我们只进行粗略的介绍，如果您想了解更多详细的内容，可以前往用户指南进行查看。

如果您想了解更多rviz的相关资料信息，您可以前往官方文档进行查看

← 上一页 | 下一页 →

Mercury X1的控制

这里主要介绍如何通过一系列相关指令对Mercury X1进行移动控制。

1.底盘底层通信

首先启动底盘的底层通信、地图构建程序，并加载Mercury X1的URDF模型。

打开ROS2环境终端，然后运行命令：

```
ros2 launch slam_gmapping slam_gmapping.launch.py
```

接着你可以通过拖动滑块来控制 rviz2 中的关节模型移动。

2.关节控制

URDF模型加载成功之后，如果你想让真实的 Mercury X1 双臂跟着一起运动，需要再打开一个ROS2环境终端，然后运行命令：

```
ros2 run mercury_x1_control slider_control
```

请注意：由于在命令输入的同时机械臂会移动到模型目前的位置，在您使用命令之前请确保rviz中的模型没有出现穿模现象 不要在连接机械臂后做出快速拖动滑块的行为，防止机械臂损坏

3 底盘控制

关节控制程序启动之后，如果你想让真实的Mercury X1中的底盘小车跟着一起运动，需要启动底盘小车的键盘控制程序，再打开一个ROS2环境终端，然后运行命令：

```
ros2 run mercury_x1_control mercury_keyboard
```

← 上一页 | 下一章 →

6 机器人使用场景案例

本章节呈现了经典的机械臂使用案例，以展示产品在富有代表性的场景中的应用。这包括了水星Mercury X1轮式人形机器人在不同领域的典型应用，突显了产品的多功能性和适用性。通过这些案例，用户可以深入了解机械臂在实际应用中的灵活性和效能，为他们在特定场景中的应用提供参考。

- 二维码识别与抓取
  在水星Mercury X1轮式人形机器人机械臂末端装上2D相机及夹爪实现视觉识别二维码抓取，二维码采用stag码，通过stag.detectMarkers函数获取stag码相机系的坐标，在通过矩阵变换获取stag码基于底座的坐标，调用坐标控制函数进行抓取。

- 空间移动案例
  可使用键盘对底盘进行遥控操作，也可可以调用mercury_ros_api库对底座进行移动控制及导航。

- 移动抓取案例
  此案例将前两个案例结合起来，使用导航将水星Mercury X1轮式人形机器人移动至指定地点，然后通过视觉识别stag，实现抓取。

- 键盘打字案例
  此案例通过识别左上角与右上角的stag码，计算出全部键位的二维坐标、、三维坐标、缩放比例和旋转角度，并将键位在画面中显示出来，通过输入指定按键实现点击。

视觉识别功能使用前的准备

1.物料及开机准备

1.1物料准备

视觉识别跟踪需要准备的物料包括： 1) Mercury系列水星机器 2) 视觉传感器及stag码配件 3) 夹爪及相关配件

1.2开机准备

将电源适配器连接上机械臂，也可通过内置电源供电，并将机器人移至空旷区域防止碰撞。

底座后部电源接口位置

电源开关位置

急停开关位置

在按下急停开关后机器人将立即停止运动，复位时需将急停开关顺时针旋转，急停开关将自动弹起，之后可上电恢复控制。

点击电源开关，并确保急停处于复位状态，开始使用机械臂。

2.Mercury整机调试

2.1整机上电

机械臂支持多种方式上电，本文仅介绍使用python指令控制上电，在Terminal终端中输入下述指令使机器人上电：

```
python
from pymycobot import Mercury
mr = Mercury("/dev/right_arm")  #connect to Mercury right arm
ml = Mercury("/dev/left_Arm")  #connect to Mercury left arm
ml.power_on()
mr.power_on()
```

发送指令前，确保急停开关处于复位状态，当听到**关节抱闸声**则代表上电成功！若未听到抱闸声，使用以下指令反复上下电：

首先发送掉电指令

```
ml.power_off()
mr.power_off()
```

等待数秒后发送上电指令

```
ml.power_on()
mr.power_on()
```

若仍未听到抱闸声，联系工程师进行处理。

2.2机械臂零点校准

机械臂使用前，需确保零位正确，可以通过下述方法校准零位：

1）使用放松指令释放关节电机（**注意！放松后需要扶住关节防止机械臂下坠损坏！**）

```
ml.release_all_servos()
mr.release_all_servos()
```

2）拖动机械臂回到零位

可以通过以下细节确定零点的位置

3）发送指令锁闭电机

```
ml.focus_all_servos()
mr.focus_all_servos()
```

4）检查零点是否正确

输入获取关节角度指令查询当前位置

```
ml.get_angles()
mr.get_angles()
```

如果返回的关节角度逼近[0, 0, 0, 0, 90, 0]，则视为零点正确

5）零点校准

如果在零位读取到的关节角度与[0, 0, 0, 0, 90, 0]相差很大，则需要校准关节零位

```
for i in range(1,7):
  ml.set_servo_calibration(i)
  mr.set_servo_calibration(i)
```

校准完毕后读取关节信息，返回为[0, 0, 0, 0, 90, 0]则表示校准成功

```
ml.get_angles()
mr.get_angles()
```

2.3机械臂运动

可以通过以下方式验证机械臂运动功能是否正常：

1)确保零点正确后，使用回零指令即可控制机械臂回到零位:

```
ml.over_limit_return_zero()
mr.over_limit_return_zero()
```

2)发送关节运动指令，验证机械臂运动是否正常

```
ml.send_angle(1,20,10)
mr.send_angle(1,20,10)
```

此时应可观测到机械臂1关节发生旋转，待运动结束后使用指令读取关节信息

```
ml.get_angles()
mr.get_angles()
```

若返回值中1关节为20°，则视为运动正常

2.4异常处理

```
若2.3中机械臂运动异常，使用以下方法观察异常信息
```

1)获取电机异常信息

```
ml.get_robot_status()
mr.get_robot_status()
```

2）获取运动异常信息

```
ml.get_error_information()
mr.get_error_information()
```

将上述接口的返回信息交付工程师处理

3.相机和夹爪的安装方法

相机和夹爪的安装方式与视觉识别的手眼矩阵相对应，若安装方式改变则需重新进行手眼标定！

首先将机械臂移动回零点：

```
ml.over_limit_return_zero()
mr.over_limit_return_zero()
```

对照下图进行夹爪和相机的安装

4.相机参数配置

4.1 查找相机编号

由于Mercury水星存在多个视觉传感器，在进行相机参数配置前，首先需要找到左右臂相机对应的编号：

1)首先进行相机初始化，在Camera_ID处依次填入0~6数字

```
obj = camera_detect(Camera_ID = 0, mark_size, mtx, dist, direction)
```

```
obj = camera_open_loop()
```

开启相机，观测相机画面，标记该相机对应的编号,如果相机无法开启，修改Camera_ID并重复该操作

2)修改Camera_ID，按照步骤1依次记录相机编号

```
obj = camera_detect(Camera_ID = 1, mark_size, mtx, dist, direction)
```

```
obj = camera_open_loop()
```

3)将左右臂相机对应的编号修改到py脚本的下述位置即可

```
ml_obj = camera_detect(Camera_ID = left, mark_size, mtx, dist, direction)
```

```
mr_obj = camera_detect(Camera_ID = right, mark_size, mtx, dist, direction)
```

4.2 相机内参标定

5.夹爪功能调试

使用以下接口进行夹爪零点校准：

1）夹爪初始化

```
ml.set_gripper_mode(0)
mr.set_gripper_mode(0)
```

2）放松夹爪

```
ml.set_gripper_enabled(0)
mr.set_gripper_enabled(0)
```

手动张开夹爪至最大

3）设置夹爪零位

```
ml.set_gripper_calibration()
mr.set_gripper_calibration()
```

4）控制夹爪闭合

```
ml.set_gripper_value(0,100)
mr.set_gripper_value(0,100)
```

此时应能观察到夹爪快速闭合

夹爪异常排查：

1) 夹爪无法运动——重复1~4步骤，**若仍然无法运动，联系我方工程师** 2) 步骤4未完全合拢——步骤2中，不要张至最大，然后设置零点

6.整机综测

执行下述py脚本，应观察到以下现象：

1) 机械臂回到零位 2) 夹爪打开后闭合 3) 机械臂运动到如下位置 4) 夹爪开合

若现象与上述情况不符，重复测试运动测试、夹爪测试章节

```
from pymycobot import *
import numpy as np
import time

mr = Mercury("/dev/right_arm")  #connect to Mercury right arm
ml = Mercury("/dev/left_Arm")  #connect to Mercury left arm

ml.over_limit_return_zero()
mr.over_limit_return_zero()
time.sleep(10)

ml.set_gripper_mode(0)
mr.set_gripper_mode(0)
time.sleep(1)
ml.set_gripper_value(100,10)
mr.set_gripper_value(100,10)
time.sleep(3)
ml.set_gripper_value(0,10)
mr.set_gripper_value(0,10)
time.sleep(3)

ml.send_angles([45, 20, -45, 45, 120, 45], 10)
mr.send_angles([-45, 20, 45, 45, 120, -45], 10)
time.sleep(5)
la = ml.get_angles()
ra = mr.get_angles()
print("left", la)
print("right", ra)

ml.set_gripper_value(100,10)
mr.set_gripper_value(100,10)
time.sleep(3)
ml.set_gripper_value(0,10)
mr.set_gripper_value(0,10)
time.sleep(3)
```

← 上一章 | 下一页 →

二维码识别与抓取

案例视频

使用camera_detect视觉识别包代码

```
当我们完成了1~6章的视觉识别使用前的准备，通过Mercury的camera_detect库调用几个简单的接口即可实
```

接口介绍:

`obj = camera_detect(Camera_ID, mark_size, mtx, dist, direction)`

- **参数:**
  - `camera_ID`：相机ID
  - `mark_size`：stag码边长mm
  - `direction`：0：左臂 1：右臂
- **功能:** 相机和机械臂初始化

`obj.camera_open_loop()`

- **功能:** 显示相机画面

`obj.stag_identify_loop()`

- **功能:** 显示相机画面,识别并打印stag码的**相机坐标**

`coords = obj.stag_robot_identify(direction)`

- **参数:**
  - `direction`：0：左臂 1：右臂
  - `return`：stag码的机械臂坐标
- **功能:** 计算stag码的**机械臂坐标**

`obj.vision_trace(catch_mode, direction)`

- **参数：**
  - `catch_mode`：0:水平抓取，1:竖直抓取
  - `direction`：0：左臂 1：右臂
- **功能：** 显示相机画面,识别并打印stag码的**相机坐标**

使用案例

```
import numpy as np
from pymycobot import Mercury
# 导入视觉识别stag包
from camera_detect import camera_detect

if __name__ == "__main__":
    camera_params = np.load("camera_params.npz")  # 相机配置文件
    mtx, dist = camera_params["mtx"], camera_params["dist"]
    ml = Mercury("/dev/left_arm")  # 设置左臂端口
    mr = Mercury("/dev/right_arm")  # 设置右臂端口
    arm = 0         # 0左臂，1右臂
    catch_mode = 0  # 0水平抓取，1竖直抓取
    # 摄像头id需要根据实际更改
    left_camera_id = 3
    right_camera_id = 6
    stag_size = 32
    # 新建左右臂视觉识别对象
    ml_obj = camera_detect(left_camera_id, stag_size, mtx, dist, arm)
    mr_obj = camera_detect(right_camera_id, stag_size, mtx, dist, arm)
    # 左右臂移动至水平观察位，进行抓取
    mr_obj.vision_trace(catch_mode, mr)
    ml_obj.vision_trace(catch_mode, ml)
```

不使用camera_detect视觉识别包代码

```
from uvc_camera import UVCCamera
from marker_utils import *
import stag

# 夹爪工具长度
Tool_LEN = 175
# 相机中心到法兰距离
Camera_LEN = 78
np.set_printoptions(suppress=True, formatter={'float_kind': '{:.2f}'.format})
# 相机配置文件
camera_params = np.load("src/camera_params.npz")
mtx, dist = camera_params["mtx"], camera_params["dist"]
# 二维码大小
MARKER_SIZE = 32
# 设置左臂端口
ml = Mercury("/dev/left_arm")


# 将旋转矩阵转为欧拉角
def CvtRotationMatrixToEulerAngle(pdtRotationMatrix):
    pdtEulerAngle = np.zeros(3)
    pdtEulerAngle[2] = np.arctan2(pdtRotationMatrix[1, 0], pdtRotationMatrix[0, 0])
    fCosRoll = np.cos(pdtEulerAngle[2])
    fSinRoll = np.sin(pdtEulerAngle[2])
    pdtEulerAngle[1] = np.arctan2(-pdtRotationMatrix[2, 0],
                                  (fCosRoll * pdtRotationMatrix[0, 0]) + (fSinRoll * p
    pdtEulerAngle[0] = np.arctan2((fSinRoll * pdtRotationMatrix[0, 2]) - (fCosRoll * p
                                  (-fSinRoll * pdtRotationMatrix[0, 1]) + (fCosRoll *
    return pdtEulerAngle


# 将欧拉角转为旋转矩阵
def CvtEulerAngleToRotationMatrix(ptrEulerAngle):
    ptrSinAngle = np.sin(ptrEulerAngle)
    ptrCosAngle = np.cos(ptrEulerAngle)
    ptrRotationMatrix = np.zeros((3, 3))
    ptrRotationMatrix[0, 0] = ptrCosAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[0, 1] = ptrCosAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] - ptrSi
    ptrRotationMatrix[0, 2] = ptrCosAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] + ptrSi
    ptrRotationMatrix[1, 0] = ptrSinAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[1, 1] = ptrSinAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] + ptrCo
    ptrRotationMatrix[1, 2] = ptrSinAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] - ptrCo
    ptrRotationMatrix[2, 0] = -ptrSinAngle[1]
    ptrRotationMatrix[2, 1] = ptrCosAngle[1] * ptrSinAngle[0]
    ptrRotationMatrix[2, 2] = ptrCosAngle[1] * ptrCosAngle[0]
    return ptrRotationMatrix
```

```python
# 坐标转换为齐次变换矩阵，（x,y,z,rx,ry,rz）单位rad
def Transformation_matrix(coord):
    position_robot = coord[:3]
    pose_robot = coord[3:]
    # 将欧拉角转为旋转矩阵
    RBT = CvtEulerAngleToRotationMatrix(pose_robot)
    PBT = np.array([[position_robot[0]],
                    [position_robot[1]],
                    [position_robot[2]]])
    temp = np.concatenate((RBT, PBT), axis=1)
    array_1x4 = np.array([[0, 0, 0, 1]])
    # 将两个数组按行拼接起来
    matrix = np.concatenate((temp, array_1x4), axis=0)
    return matrix


def Eyes_in_hand_left(coord, camera):
    # 相机坐标
    Position_Camera = np.transpose(camera[:3])
    # 机械臂坐标矩阵
    Matrix_BT = Transformation_matrix(coord)
    # 手眼矩阵
    Matrix_TC = np.array([[0, -1, 0, Camera_LEN],
                          [1, 0, 0, 0],
                          [0, 0, 1, -Tool_LEN],
                          [0, 0, 0, 1]])
    # 物体坐标（相机系）
    Position_Camera = np.append(Position_Camera, 1)
    # 物体坐标（基坐标系）
    Position_B = Matrix_BT @ Matrix_TC @ Position_Camera
    return Position_B


# 等待机械臂运行结束
def waitl():
    time.sleep(0.2)
    while (ml.is_moving()):
        time.sleep(0.03)


# 获取物体坐标(相机系)
def calc_markers_base_position(corners: NDArray, ids: T.List, marker_size: int, mtx: N
    if len(corners) == 0:
        return []
    # 通过二维码角点获取物体旋转向量和平移向量
    rvecs, tvecs = solve_marker_pnp(corners, marker_size, mtx, dist)
    for i, tvec, rvec in zip(ids, tvecs, rvecs):
        tvec = tvec.squeeze().tolist()
```

```
        rvec = rvec.squeeze().tolist()
        rotvector = np.array([[rvec[0], rvec[1], rvec[2]]])
        # 将旋转向量转为旋转矩阵
        Rotation = cv2.Rodrigues(rotvector)[0]
        # 将旋转矩阵转为欧拉角
        Euler = CvtRotationMatrixToEulerAngle(Rotation)
        # 物体坐标(相机系)
        target_coords = np.array([tvec[0], tvec[1], tvec[2], Euler[0], Euler[1], Euler
    return target_coords


if __name__ == "__main__":
    # 设置摄像头id
    camera = UVCCamera(5, mtx, dist)
    # 打开摄像头
    camera.capture()
    # 设置左臂观察点
    origin_anglesL = [-44.24, 15.56, 0.0, -102.59, 65.28, 52.06, 23.49]
    # 设置夹爪运动模式
    ml.set_gripper_mode(0)
    # 设置工具坐标系
    ml.set_tool_reference([0, 0, Tool_LEN, 0, 0, 0])
    # 将末端坐标系设置为工具
    ml.set_end_type(1)
    # 设置移动速度
    sp = 40
    # 移动到观测点
    ml.send_angles(origin_anglesL, sp)
    # 等待机械臂运动结束
    waitl()
    # 刷新相机界面
    camera.update_frame()
    # 获取当前帧
    frame = camera.color_frame()
    # 获取画面中二维码的角度和id
    (corners, ids, rejected_corners) = stag.detectMarkers(frame, 11)
    # 获取物的坐标(相机系)
    marker_pos_pack = calc_markers_base_position(corners, ids, MARKER_SIZE, mtx, dist)
    # 获取机械臂当前坐标
    cur_coords = np.array(ml.get_base_coords())
    # 将角度值转为弧度值
    cur_bcl = cur_coords.copy()
    cur_bcl[-3:] *= (np.pi / 180)
    # 通过矩阵变化将物体坐标(相机系)转成(基坐标系)
    fact_bcl = Eyes_in_hand_left(cur_bcl, marker_pos_pack)
    target_coords = cur_coords.copy()
    target_coords[0] = fact_bcl[0]
    target_coords[1] = fact_bcl[1]
```

```
    target_coords[2] = fact_bcl[2] + 50
    # 机械臂移动到二维码上方
    ml.send_base_coords(target_coords, 30)
    # 等待机械臂运动结束
    waitl()
    # 打开夹爪
    ml.set_gripper_value(100, 100)
    # 机械臂沿z轴向下移动
    ml.send_base_coord(3, fact_bcl[2], 10)
    # 等待机械臂运动结束
    waitl()
    # 闭合夹爪
    ml.set_gripper_value(20, 100)

    # 等待夹爪闭合
    time.sleep(2)

    # 抬起夹爪
    ml.send_base_coord(3, fact_bcl[2] + 50, 10)
```

← 上一页 | 下一页 →

空间移动

注意底盘电量高于21V才可移动

案例视频

键盘控制底座移动

在一个全新终端执行

```
roslaunch turn_on_mercury_robot turn_on_mercury_robot.launch
```

在另一个全新终端执行

```
roslaunch mercury_x1_teleop keyboard_teleop.launch
```

此时根据控制台提示点击指定键盘按键，可控制底盘移动。

通过api控制底盘移动和导航

在一个全新终端执行

```
roslaunch turn_on_mercury_robot navigation.launch
```

```
from mercury_ros_api import MapNavigation

if __name__ == "__main__":
    # 导航到目标点，此点位需自行建图取点
    map_navigation = MapNavigation()
    flag_feed_goalReached = map_navigation.moveToGoal(1.8811798181533813, 1.2514267325
    # 底盘向前移动
    map_navigation.goStraight(0.25, 5)
    # 底盘向后移动
    map_navigation.goBack(0.25, 5)
    # 底盘向左移动
    map_navigation.turnLeft(0.25, 5)
    # 底盘向右移动
    map_navigation.turnRight(0.25, 5)
```

← 上一页 | 下一页 →

移动和抓取

案例视频

使用camera_detect视觉识别包代码

```
import numpy as np
from pymycobot import Mercury
# 导入视觉识别stag包
from camera_detect import camera_detect

if __name__ == "__main__":
    # 导航到目标点，此点位需自行建图取点
    map_navigation = MapNavigation()
    flag_feed_goalReached = MapNavigation.moveToGoal(1.8811798181533813, 1.2514267325
    # 判断是否到底目标点
    if flag_feed_goalReached:
        camera_params = np.load("camera_params.npz")  # 相机配置文件
        mtx, dist = camera_params["mtx"], camera_params["dist"]
        ml = Mercury("/dev/left_arm")  # 设置左臂端口
        mr = Mercury("/dev/right_arm")  # 设置右臂端口
        arm = 0         # 0左臂，1右臂
        catch_mode = 0  # 0水平抓取，1竖直抓取
        # 摄像头id需要根据实际更改
        left_camera_id = 3
        right_camera_id = 6
        stag_size = 32
        # 新建左右臂视觉识别对象
        ml_obj = camera_detect(left_camera_id, stag_size, mtx, dist, arm)
        mr_obj = camera_detect(right_camera_id, stag_size, mtx, dist, arm)
        # 左右臂移动至水平观察位，进行抓取
        mr_obj.vision_trace(catch_mode, mr)
        ml_obj.vision_trace(catch_mode, ml)
```

不使用camera_detect视觉识别包代码

```
from uvc_camera import UVCCamera
from arm_control import *
from marker_utils import *
import stag
from mercury_ros_api import MapNavigation

# 夹爪工具长度
Tool_LEN = 175
# 相机中心到法兰距离
Camera_LEN = 78
np.set_printoptions(suppress=True, formatter={'float_kind': '{:.2f}'.format})
# 相机配置文件
camera_params = np.load("src/camera_params.npz")
mtx, dist = camera_params["mtx"], camera_params["dist"]
# 二维码大小
MARKER_SIZE = 32
# 设置左臂端口
ml = Mercury("/dev/left_arm")


# 将旋转矩阵转为欧拉角
def CvtRotationMatrixToEulerAngle(pdtRotationMatrix):
    pdtEulerAngle = np.zeros(3)
    pdtEulerAngle[2] = np.arctan2(pdtRotationMatrix[1, 0], pdtRotationMatrix[0, 0])
    fCosRoll = np.cos(pdtEulerAngle[2])
    fSinRoll = np.sin(pdtEulerAngle[2])
    pdtEulerAngle[1] = np.arctan2(-pdtRotationMatrix[2, 0],
                                  (fCosRoll * pdtRotationMatrix[0, 0]) + (fSinRoll * p
    pdtEulerAngle[0] = np.arctan2((fSinRoll * pdtRotationMatrix[0, 2]) - (fCosRoll * p
                                  (-fSinRoll * pdtRotationMatrix[0, 1]) + (fCosRoll *
    return pdtEulerAngle


# 将欧拉角转为旋转矩阵
def CvtEulerAngleToRotationMatrix(ptrEulerAngle):
    ptrSinAngle = np.sin(ptrEulerAngle)
    ptrCosAngle = np.cos(ptrEulerAngle)
    ptrRotationMatrix = np.zeros((3, 3))
    ptrRotationMatrix[0, 0] = ptrCosAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[0, 1] = ptrCosAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] - ptrSi
    ptrRotationMatrix[0, 2] = ptrCosAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] + ptrSi
    ptrRotationMatrix[1, 0] = ptrSinAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[1, 1] = ptrSinAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] + ptrCo
    ptrRotationMatrix[1, 2] = ptrSinAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] - ptrCo
    ptrRotationMatrix[2, 0] = -ptrSinAngle[1]
    ptrRotationMatrix[2, 1] = ptrCosAngle[1] * ptrSinAngle[0]
    ptrRotationMatrix[2, 2] = ptrCosAngle[1] * ptrCosAngle[0]
    return ptrRotationMatrix
```

```python
# 坐标转换为齐次变换矩阵，（x,y,z,rx,ry,rz）单位rad
def Transformation_matrix(coord):
    position_robot = coord[:3]
    pose_robot = coord[3:]
    # 将欧拉角转为旋转矩阵
    RBT = CvtEulerAngleToRotationMatrix(pose_robot)
    PBT = np.array([[position_robot[0]],
                    [position_robot[1]],
                    [position_robot[2]]])
    temp = np.concatenate((RBT, PBT), axis=1)
    array_1x4 = np.array([[0, 0, 0, 1]])
    # 将两个数组按行拼接起来
    matrix = np.concatenate((temp, array_1x4), axis=0)
    return matrix


def Eyes_in_hand_left(coord, camera):
    # 相机坐标
    Position_Camera = np.transpose(camera[:3])
    # 机械臂坐标矩阵
    Matrix_BT = Transformation_matrix(coord)
    # 手眼矩阵
    Matrix_TC = np.array([[0, -1, 0, Camera_LEN],
                          [1, 0, 0, 0],
                          [0, 0, 1, -Tool_LEN],
                          [0, 0, 0, 1]])
    # 物体坐标（相机系）
    Position_Camera = np.append(Position_Camera, 1)
    # 物体坐标（基坐标系）
    Position_B = Matrix_BT @ Matrix_TC @ Position_Camera
    return Position_B


# 等待机械臂运行结束
def waitl():
    time.sleep(0.2)
    while (ml.is_moving()):
        time.sleep(0.03)


# 获取物体坐标(相机系)
def calc_markers_base_position(corners: NDArray, ids: T.List, marker_size: int, mtx: N
    if len(corners) == 0:
        return []
    # 通过二维码角点获取物体旋转向量和平移向量
    rvecs, tvecs = solve_marker_pnp(corners, marker_size, mtx, dist)
```

```
    for i, tvec, rvec in zip(ids, tvecs, rvecs):
        tvec = tvec.squeeze().tolist()
        rvec = rvec.squeeze().tolist()
        rotvector = np.array([[rvec[0], rvec[1], rvec[2]]])
        # 将旋转向量转为旋转矩阵
        Rotation = cv2.Rodrigues(rotvector)[0]
        # 将旋转矩阵转为欧拉角
        Euler = CvtRotationMatrixToEulerAngle(Rotation)
        # 物体坐标(相机系)
        target_coords = np.array([tvec[0], tvec[1], tvec[2], Euler[0], Euler[1], Euler
    return target_coords


if __name__ == "__main__":
     # 导航到目标点，此点位需自行建图取点
     map_navigation = MapNavigation()
     flag_feed_goalReached = MapNavigation.moveToGoal(1.8811798181533813, 1.2514267325
     # 判断是否到底目标点
     if flag_feed_goalReached:
        # 设置摄像头id
        camera = UVCCamera(5, mtx, dist)
        # 打开摄像头
        camera.capture()
        # 设置左臂观察点
        origin_anglesL = [-44.24, 15.56, 0.0, -102.59, 65.28, 52.06, 23.49]
        # 设置夹爪运动模式
        ml.set_gripper_mode(0)
        # 设置工具坐标系
        ml.set_tool_reference([0, 0, Tool_LEN, 0, 0, 0])
        # 将末端坐标系设置为工具
        ml.set_end_type(1)
        # 设置移动速度
        sp = 40
        # 移动到观测点
        ml.send_angles(origin_anglesL, sp)
        # 等待机械臂运动结束
        waitl()
        # 刷新相机界面
        camera.update_frame()
        # 获取当前帧
        frame = camera.color_frame()
        # 获取画面中二维码的角度和id
        (corners, ids, rejected_corners) = stag.detectMarkers(frame, 11)
        # 获取物的坐标(相机系)
        marker_pos_pack = calc_markers_base_position(corners, ids, MARKER_SIZE, mtx, d
        # 获取机械臂当前坐标
        cur_coords = np.array(ml.get_base_coords())
        # 将角度值转为弧度值
```

```
        cur_bcl = cur_coords.copy()
        cur_bcl[-3:] *= (np.pi / 180)
        # 通过矩阵变化将物体坐标(相机系)转成(基坐标系)
        fact_bcl = Eyes_in_hand_left(cur_bcl, marker_pos_pack)
        target_coords = cur_coords.copy()
        target_coords[0] = fact_bcl[0]
        target_coords[1] = fact_bcl[1]
        target_coords[2] = fact_bcl[2] + 50
        # 机械臂移动到二维码上方
        ml.send_base_coords(target_coords, 30)
        # 等待机械臂运动结束
        waitl()
        # 打开夹爪
        ml.set_gripper_value(100, 100)
        # 机械臂沿z轴向下移动
        ml.send_base_coord(3, fact_bcl[2], 10)
        # 等待机械臂运动结束
        waitl()
        # 闭合夹爪
        ml.set_gripper_value(20, 100)

        # 等待夹爪闭合
        time.sleep(2)

        # 抬起夹爪
        ml.send_base_coord(3, fact_bcl[2] + 50, 10)
```

← 上一页 | 下一章 →

键盘打字

案例视频

案例代码

```python
import json
import threading

from uvc_camera import UVCCamera
import cv2
import time
import numpy as np
from transformation import *
from pymycobot import Mercury
import typing as T
import cv2.aruco as aruco
from arm_control import *
from marker_utils import *
import stag
from matplotlib.path import Path


# 将旋转矩阵转为欧拉角
def CvtRotationMatrixToEulerAngle(pdtRotationMatrix):
    pdtEulerAngle = np.zeros(3)
    pdtEulerAngle[2] = np.arctan2(pdtRotationMatrix[1, 0], pdtRotationMatrix[0, 0])
    fCosRoll = np.cos(pdtEulerAngle[2])
    fSinRoll = np.sin(pdtEulerAngle[2])
    pdtEulerAngle[1] = np.arctan2(-pdtRotationMatrix[2, 0],
                                  (fCosRoll * pdtRotationMatrix[0, 0]) + (fSinRoll * p
    pdtEulerAngle[0] = np.arctan2((fSinRoll * pdtRotationMatrix[0, 2]) - (fCosRoll * p
                                  (-fSinRoll * pdtRotationMatrix[0, 1]) + (fCosRoll *
    return pdtEulerAngle


# 获取物体坐标(相机系)
def calc_markers_base_position(corners: NDArray, ids: T.List, marker_size: int, mtx: N
    if len(corners) == 0:
        return []
    rvecs, tvecs = solve_marker_pnp(corners, marker_size, mtx, dist)
    res = []
    for i, tvec, rvec in zip(ids, tvecs, rvecs):
        tvec = tvec.squeeze().tolist()
        rvec = rvec.squeeze().tolist()
        rotvector = np.array([[rvec[0], rvec[1], rvec[2]]])
        Rotation = cv2.Rodrigues(rotvector)[0]
        Euler = CvtRotationMatrixToEulerAngle(Rotation)
        cam_coords = tvec + rvec
        target_coords = cam_coords
    return target_coords


# 将欧拉角转为旋转矩阵
```

```
def CvtEulerAngleToRotationMatrix(ptrEulerAngle):
    ptrSinAngle = np.sin(ptrEulerAngle)
    ptrCosAngle = np.cos(ptrEulerAngle)
    ptrRotationMatrix = np.zeros((3, 3))
    ptrRotationMatrix[0, 0] = ptrCosAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[0, 1] = ptrCosAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] - ptrSi
    ptrRotationMatrix[0, 2] = ptrCosAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] + ptrSi
    ptrRotationMatrix[1, 0] = ptrSinAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[1, 1] = ptrSinAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] + ptrCo
    ptrRotationMatrix[1, 2] = ptrSinAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] - ptrCo
    ptrRotationMatrix[2, 0] = -ptrSinAngle[1]
    ptrRotationMatrix[2, 1] = ptrCosAngle[1] * ptrSinAngle[0]
    ptrRotationMatrix[2, 2] = ptrCosAngle[1] * ptrCosAngle[0]
    return ptrRotationMatrix


# 坐标转换为齐次变换矩阵，（x,y,z,rx,ry,rz）单位rad
def Transformation_matrix(coord):
    position_robot = coord[:3]
    pose_robot = coord[3:]
    RBT = CvtEulerAngleToRotationMatrix(pose_robot)
    PBT = np.array([[position_robot[0]],
                    [position_robot[1]],
                    [position_robot[2]]])
    temp = np.concatenate((RBT, PBT), axis=1)
    array_1x4 = np.array([[0, 0, 0, 1]])
    # 将两个数组按行拼接起来
    matrix = np.concatenate((temp, array_1x4), axis=0)
    return matrix


def Eyes_in_hand(coord, camera, arm):
    # 相机坐标
    Position_Camera = np.transpose(camera[:3])
    # 机械臂坐标矩阵
    Matrix_BT = Transformation_matrix(coord)
    # 手眼矩阵
    if arm == "left":
        Matrix_TC = np.array([[0, -1, 0, Camera_LEN],
                              [1, 0, 0, -3],
                              [0, 0, 1, -Tool_LEN],
                              [0, 0, 0, 1]])
    else:
        Matrix_TC = np.array([[0, 1, 0, Camera_LEN],
                              [-1, 0, 0, 28],
                              [0, 0, 1, -Tool_LEN],
                              [0, 0, 0, 1]])
    # 物体坐标（相机系）
```

```
    Position_Camera = np.append(Position_Camera, 1)
    Position_B = Matrix_BT @ Matrix_TC @ Position_Camera
    return Position_B


# 等待双臂运行结束
def wait():
    time.sleep(0.2)
    while (ml.is_moving() or mr.is_moving()):
        time.sleep(0.03)


# 等待左臂运行结束
def waitL():
    time.sleep(0.2)
    while (ml.is_moving()):
        time.sleep(0.03)


# 等待右臂运行结束
def waitR():
    time.sleep(0.2)
    while (mr.is_moving()):
        time.sleep(0.03)


# 画出键盘框，读取各字母坐标
def draw(frame, arm):
    global per_right_corners, per_left_corners
    # 将图片灰度
    imGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 获取图片中二维码的角点
    (corners, ids, rejected_corners) = stag.detectMarkers(imGray, 11)
    # 通过角点获取，二维码相对于相机的位移、旋转向量
    marker_pos_pack = calc_markers_base_position(corners, ids, marker_size, mtx, dist)
    if arm == "left":
        # 获取当前机械臂末端坐标
        stag_cur_coords = np.array(ml.get_base_coords())
        stag_cur_bcl = stag_cur_coords.copy()
        stag_cur_bcl[-3:] *= (np.pi / 180)
        # 通过手眼矩阵获取二维码相对于机械臂基座的坐标
        stag_fact_bcl = Eyes_in_hand(stag_cur_bcl, marker_pos_pack, "left")
        stag_coord = stag_cur_coords.copy()
        stag_coord[0] = stag_fact_bcl[0]
        stag_coord[1] = stag_fact_bcl[1]
        stag_coord[2] = stag_fact_bcl[2]
        # 存入二维码的三维坐标
        keyboard_coords[","] = stag_coord
```

```
    else:
        # 获取当前机械臂末端坐标
        stag_cur_coords = np.array(mr.get_base_coords())
        stag_cur_bcl = stag_cur_coords.copy()
        stag_cur_bcl[-3:] *= (np.pi / 180)
        # 通过手眼矩阵获取二维码相对于机械臂基座的坐标
        stag_fact_bcl = Eyes_in_hand(stag_cur_bcl, marker_pos_pack, "right")
        stag_coord = stag_cur_coords.copy()
        stag_coord[0] = stag_fact_bcl[0]
        stag_coord[1] = stag_fact_bcl[1]
        stag_coord[2] = stag_fact_bcl[2]
        # 存入二维码的三维坐标
        keyboard_coords["."] = stag_coord
    # 通过角点获取，二维码相对于相机的位移、旋转向量
    rvecs, tvecs = solve_marker_pnp(corners, marker_size, mtx, dist)
    # 画出坐标系
    cv2.drawFrameAxes(frame, mtx, dist, rvecs, tvecs, 50)
    draw_img = frame

    x1 = corners[0][0][0][0]
    y1 = corners[0][0][0][1]
    x2 = corners[0][0][1][0]
    y2 = corners[0][0][1][1]

    x = x1 - x2
    y = y1 - y2
    # 根据两个交点之间连线的角度获取偏转角
    r = np.arctan(y / x)
    r = abs(r)
    # 获取两个角点之间的距离
    size = abs(x / np.cos(r))
    # 左臂摄像头两个按键x轴之间的距离
    left_x_dis = size * 1.3
    # 左臂摄像头两个按键y轴之间的距离
    left_y_dis = size * 1.35
    # 右臂摄像头两个按键x轴之间的距离
    right_x_dis = size * 1.3
    # 右臂摄像头两个按键y轴之间的距离
    right_y_dis = size * 1.35
    # 按键框的半径
    rad = int(size / 2)
    # 左臂摄像头x轴与第一个字母的偏移距离
    left_add_x = [size * 1.25]
    # 左臂摄像头y轴各行与第一个字母的偏移距离
    left_add_y = [-size * 2, -size * 2.3, -size * 3]
    # 右臂摄像头x轴与第一个字母的偏移距离
    right_add_x = [size * 1.3]
    # 右臂摄像头y轴各行与第一个字母的偏移距离
```

```
right_add_y = [size * 4.1, size * 2.1, size * 1]
# 获取按键框的中心点
tray_frame = Path(corners[0][0])
tray_frame_center_plot = tray_frame.vertices.mean(axis=0)

# 画出按键框
def draw_keyboard(arm):
    if arm == "left":
        # 摄像头x轴与第一个字母的偏移距离
        add_x = left_add_x
        # 左臂摄像头y轴各行与第一个字母的偏移距离
        add_y = left_add_y
        # 左臂按键布局
        keyboard_txt = left_keyboard_txt
        # 左臂摄像头两个按键x轴之间的距离
        x_dis = left_x_dis
        # 左臂摄像头两个按键y轴之间的距离
        y_dis = left_y_dis
    else:
        # 摄像头x轴与第一个字母的偏移距离
        add_x = right_add_x
        # 摄像头y轴各行与第一个字母的偏移距离
        add_y = right_add_y
        # 右臂按键布局
        keyboard_txt = right_keyboard_txt
        # 右臂摄像头两个按键x轴之间的距离
        x_dis = right_x_dis
        # 右臂摄像头两个按键y轴之间的距离
        y_dis = right_y_dis

    # 循环绘制每行每列的按键框
    for j in range(len(keyboard_txt)):
        for i in range(len(keyboard_txt[j])):
            # 获取对应的按键字母
            txt = keyboard_txt[j][i]
            # 获取对应按键的二维坐标
            if arm == "left":
                x_key = int(tray_frame_center_plot[0] + add_x[0] + x_dis * j)
                y_key = int(tray_frame_center_plot[1] + add_y[j] - y_dis * i)
            else:
                x_key = int(tray_frame_center_plot[0] - x_dis * j)
                y_key = int(tray_frame_center_plot[1] + add_y[j] + y_dis * i)
            if y1 < y2:
                rr = -r
            else:
                rr = r
            row = draw_img.shape[0]
```

```
x_key = x_key
y_key = row - y_key
x_center = tray_frame_center_plot[0]
y_center = row - tray_frame_center_plot[1]
# 获取图像中一个点绕另一个点旋转后的位置
x_end = (x_key - x_center) * np.cos(rr) - (y_key - y_center) * np.sin(
y_end = (x_key - x_center) * np.sin(rr) + (y_key - y_center) * np.cos(
x_end = int(x_end)
y_end = int(row - y_end)
# 计算出这个按键的角点
key_corners = (np.array(
    [[(x_end - size / 2, y_end - size / 2),
      (x_end + size / 2, y_end - size / 2),
      (x_end + size / 2, y_end + size / 2),
      (x_end - size / 2, y_end + size / 2)]],
    dtype=np.float32,
),)
# 获取按键的坐标(相机系)
marker_pos_pack = calc_markers_base_position(key_corners, ids, marker_
if arm == "left":
    # 获取机械臂当前坐标
    left_cur_coords = np.array(ml.get_base_coords())
    left_cur_bcl = left_cur_coords.copy()
    left_cur_bcl[-3:] *= (np.pi / 180)
    # 通过矩阵变化将按键坐标(相机系)转成(基坐标系)
    left_fact_bcl = Eyes_in_hand(left_cur_bcl, marker_pos_pack, "left"
    left_end_coord = left_cur_coords.copy()
    left_end_coord[0] = left_fact_bcl[0]
    left_end_coord[1] = left_fact_bcl[1]
    left_end_coord[2] = stag_coord[2]
    # 将按键位置保存入字典中，方便后续点按
    if txt in left_control:
        keyboard_coords[txt] = left_end_coord
else:
    # 获取机械臂当前坐标
    right_cur_coords = np.array(mr.get_base_coords())
    right_cur_bcl = right_cur_coords.copy()
    right_cur_bcl[-3:] *= (np.pi / 180)
    # 通过矩阵变化将按键坐标(相机系)转成(基坐标系)
    right_fact_bcl = Eyes_in_hand(right_cur_bcl, marker_pos_pack, "rig
    right_end_coord = right_cur_coords.copy()
    right_end_coord[0] = right_fact_bcl[0]
    right_end_coord[1] = right_fact_bcl[1]
    right_end_coord[2] = stag_coord[2]
    # 将按键位置保存入字典中，方便后续点按
    if txt in right_control:
        keyboard_coords[txt] = right_end_coord
# 画出按键的框
```

```
                cv2.circle(draw_img, (x_end, y_end), rad, (0, 0, 255), 2)

    draw_keyboard(arm)
    return draw_img


# 左右壁微调量
l_x = 3
l_y = 3
r_x = - 5
r_y = -10


def keyboard():
    while True:
        key_code = cv2.waitKey(1) & 0xFF
        if key_code != 255:
            key_code = chr(key_code)
            if key_code in left_control:
                # 读取对应按键的三维坐标
                coords = keyboard_coords[key_code].copy()
                coords[0] = coords[0] + l_x
                coords[1] = coords[1] + l_y
                coords[2] = 315
                # 控制点按按键
                ml.send_base_coords(coords, sp)
                waitL()
                ml.send_base_coord(3, 305, sp)
                waitL()
                ml.send_base_coords(coords, sp)
                waitL()
                ml.send_angles(ml_pos, sp)
                waitL()
                print(coords)
            elif key_code in right_control:
                # 读取对应按键的三维坐标
                coords = keyboard_coords[key_code].copy()
                coords[0] = coords[0] + r_x
                coords[1] = coords[1] + r_y
                coords[2] = 330
                # 控制点按按键
                mr.send_base_coords(coords, sp)
                waitR()
                mr.send_base_coord(3, 320, sp)
                waitR()
                mr.send_base_coords(coords, sp)
                waitR()
                mr.send_angles(mr_pos, sp)
```

```
                waitR()
                print(coords)
            print("==============================================")


def write_string(txt):
    per_key = None
    # 循环点击按键
    for key_code in txt:
        cv2.waitKey(1)
        if key_code in left_control:
            # 读取上一个按键是哪个臂，如果是同个臂则连续点击，不是则另一个臂回初始点
            if per_key not in left_control:
                # 回到初始点
                mr.send_angles(mr_pos, sp)
                waitR()
            per_key = key_code
            # 读取对应按键的三维坐标
            coords = keyboard_coords[key_code].copy()
            coords[0] = coords[0] + l_x
            coords[1] = coords[1] + l_y
            coords[2] = 315
            # 控制机械臂点按
            ml.send_base_coords(coords, sp)
            waitL()
            ml.send_base_coord(3, 305, sp)
            waitL()
            ml.send_base_coords(coords, sp)
            waitL()
            print(coords)
        elif key_code in right_control:
            # 读取上一个按键是哪个臂，如果是同个臂则连续点击，不是则另一个臂回初始点
            if per_key not in right_control:
                # 回到初始点
                ml.send_angles(ml_pos, sp)
                waitL()
            per_key = key_code
            # 读取对应按键的三维坐标
            coords = keyboard_coords[key_code].copy()
            coords[0] = coords[0] + r_x
            coords[1] = coords[1] + r_y
            coords[2] = 330
            # 控制机械臂点按
            mr.send_base_coords(coords, sp)
            waitR()
            mr.send_base_coord(3, 320, sp)
            waitR()
            mr.send_base_coords(coords, sp)
```

```
            waitR()
            print(coords)
        print("==============================================")


# 打开相机读取画面
def camera():
    per_frame_left = None
    per_frame_right = None
    while True:
        left_camera.update_frame()
        left_frame = left_camera.color_frame()
        try:
            left_frame = draw(left_frame, "left")
            per_frame_left = left_frame
        except Exception as e:
            pass
        right_camera.update_frame()
        right_frame = right_camera.color_frame()
        try:
            right_frame = draw(right_frame, "right")
            per_frame_right = right_frame
        except Exception as e:
            pass
        # 旋转图像
        left_turn_img = cv2.rotate(left_frame, cv2.ROTATE_90_CLOCKWISE)
        right_turn_img = cv2.rotate(right_frame, cv2.ROTATE_90_CLOCKWISE)
        # 拼接图像
        imgs = np.hstack([left_turn_img, right_turn_img])
        # 展示图片
        cv2.imshow("calibed", imgs)
        key_code = cv2.waitKey(1) & 0xFF
        # 当按下q键时，截取图片进行操作
        if chr(key_code) == "q":
            left_turn_img = cv2.rotate(per_frame_left, cv2.ROTATE_90_CLOCKWISE)
            right_turn_img = cv2.rotate(per_frame_right, cv2.ROTATE_90_CLOCKWISE)
            imgs = np.hstack([left_turn_img, right_turn_img])
            cv2.imshow("calibed", imgs)
            break
    # 控制双臂点击键盘helloworld
    write_string("helloworld")


if __name__ == "__main__":
    # 设置左右壁端口
    mr = Mercury("/dev/ttyACM0")
    ml = Mercury("/dev/ttyTHS0")
    # 储存左右臂读取二维码的角点值
```

```
per_left_corners = []
per_right_corners = []
sp = 60    # 机械臂运动速度
Tool_LEN = 175  # 夹爪与与法兰的距离
Camera_LEN = 78  # 相机与法兰的距离
# 设置左右臂初始点角度
ml_pos = [-31.92, -7.36, 0.0, -110.82, 81.75, 59.04, 44.25]
mr_pos = [27.12, 28.34, 0.0, -108.77, -69.42, 71.92, -9.52]
# 左臂键盘布局
left_keyboard_txt = [
    ["q", "w", "e", "r", "t", "y", "u", "i", "o", "p"],
    ["a", "s", "d", "f", "g", "h", "j", "k", "l"],
    ["z", "x", "c", "v", "b", "n", "m"]
]
# 右臂键盘布局
right_keyboard_txt = [
    ["m", "n", "b", "v", "c", "x", "z"],
    ["l", "k", "j", "h", "g", "f", "d", "s", "a"],
    ["p", "o", "i", "u", "y", "t", "r", "e", "w", "q"],
]
# 左臂点按的字母
left_control = ["q", "w", "e", "r", "t",
                "a", "s", "d", "f", "g",
                "z", "x", "c", "v", ","]
# 左臂点按的字母
right_control = ["y", "u", "i", "o", "p",
                 "h", "j", "k", "l",
                 "b", "n", "m", "."]
# 储存全部键盘按键的三维坐标
keyboard_coords = {}

np.set_printoptions(suppress=True, formatter={'float_kind': '{:.2f}'.format})
# 相机配置文件
camera_params = np.load("src/camera_params.npz")
mtx, dist = camera_params["mtx"], camera_params["dist"]
# 二维码大小
marker_size = 15
# 设置左右臂相机id
left_camera = UVCCamera(3, mtx, dist)
left_camera.capture()
right_camera = UVCCamera(2, mtx, dist)
right_camera.capture()
# 左右壁移动到初始点
mr.send_angles(mr_pos, sp)
ml.send_angles(ml_pos, sp)
# 等待运动结束
wait()
time.sleep(2)
```

```
    camera()
    # threading.Thread(target=camera).start()
    # while True:
    #     s = input("键盘字母: ")
    #     if s in left_control:
    #         coords = keyboard_coords[s]
    #         print(coords)
    #
    #     elif s in right_control:
    #         coords = keyboard_coords[s]
    #         print(coords)
```

← 上一页 | 下一章 →

大象机器人

1. 公司介绍

大象机器人(Elephant Robotics)立足于中国·深圳，是一家专注于机器人研发设计及自动化解决方案的高新科技企业。

我们致力于为机器人教育及科研机构、商业场景、工业生产提供高柔性的协作机器人、简单易学的操作系统以及智能的自动化解决方案。其产品质量及智慧方案备受韩国、日本、美国、德国、意大利、希腊等数家来自世界 500 强名企工厂的一致认可与好评。

大象机器人秉持“Enjoy Robots World”的愿景，倡导人与机器人的协同工作，让机器人成为人类工作生活的好帮手，帮助人们从简单、重复、枯燥的工作中解放出来，充分发挥人机协同优势，进而提高工作效率，帮助人类缔造美好新生活。

未来，大象机器人希望通过新一代尖端科技推动机器人产业发展，携手与客户伙伴们共同开启自动化智能化新时代。

2. 发展历程

- 2016.08 -----大象机器人有限公司正式成立
- 2016.08 -----进入 HAX 孵化器，获得 SOSV 种子轮投资
- 2016.08 ----- 开始研发 Elephant S 工业协作机器人
- 2017.01 -----获评 “CES 中国最具创新企业 Top10”
- 2017.04 -----出席汉诺威工业博览会及韩国自动化展览会
- 2017.07 -----两位创始人入选福布斯亚洲评选的“30 位 30 岁以下商业精英”
- 2017.10 -----第五代单臂工业协作机器人 Elephant S 问世
- 2018.04 -----获得”云天使基金”天使轮投资
- 2018.06 -----首次公开亮相 2018 年汉诺威世界工业博览会
- 2018.06 -----获得长江商学院“智造创业 MBA 奖”
- 2018.06 -----获得清华经管“创业加速器 X-elerator 奖”
- 2018.11 -----获得亚洲智能硬件大赛深圳赛区第二名
- 2018.11 -----获得高工金球奖“最具投资企业奖”
- 2019.03 -----获得高工金球奖“领军人物奖”
- 2019.04 -----2019 年 3 月 Catbot 获“工业机器人创新奖”
- 2019.09 -----出席华为欧洲生态大会(HCE)，正式成为华为生态伙伴一员
- 2019.11 -----大象机器人携手哈工大出席 IROS 国际智能机器人与系统大会
- 2019.12 -----大象机器人-华南理工大学“智能机器人联合开发实验室”正式揭牌
- 2019.12 -----荣获高工 2019 年度“创新技术奖”
- 2019.12 -----荣获高工 2019 年度“十大快速成长企业”
- 2019.12 -----荣获深圳装备工业-工业机器人细分领域-“新锐企业奖”
- 2019.12 -----世界首款仿生机器猫 MarsCat 问世
- 2020.05 -----创始人获得 2019 年度深圳市机器人新锐人物奖
- 2020.10 -----全球最轻最小的六轴协作机器人 myCobot 问世
- 2021.03 -----面向科研的最小协作机器人 myCobotPro 320 问世
- 2021.05 -----火星仿生猫 MarsCat 获得新华财经、中国日报、南京日报、哈尔滨日报等多家媒体的竞相报道
- 2021.07 -----发布最小的复合机器人底盘 – 小象移动机器人 myAGV
- 2021.09 -----全球首款全包裹式的四轴机械臂-小象码垛机械臂 myPalletizer 问世
- 2022.01 -----大象机器人获36氪、极客公园关于轻量化消费级机器人行业系列报道
- 2022.02 -----MarsCat、myCobot亮相春晚央视频直播，参与深圳卫视过年特备节目
- 2022.05 -----最紧凑的小六轴机械臂mechArm问世，聚焦人工智能机器人教育
- 2022.06 -----联合Unity引擎，基于myCobot 机器人，推出人工智能机器人实践入门书籍(国际课程)
- 2022.07 -----发布人工智能时代的仿真陪伴机器猫米塔猫metaCat
- 2022.07 -----发布史上最小双臂协作机器人mybuddy
- 2022.08 -----获得“十大非工业技术创新奖”
- 2022.08 -----创始人获“2022深圳市机器人新锐人物奖”
- 2022.11 -----科大讯飞AI开发者大赛real time enganement（实时互动）赛道亚军
- 2022.12 -----2022世界声博会1024科博展最佳机器人奖
- 2023 -----捐赠获上海普陀区展翼儿童培智服务中心、深圳市福田区童伴时光特殊儿童康复中心、任达养老产业集团证书
- 2023.01 -----央视新闻-华强北创业者专访《向春而行｜深夜华强北》
- 2023.01 -----桌面级高精度机械臂ultraArm面市

- 2023.02 -----微软官方使用大象myCobot 280机械臂开展ChatGPT for Robotics实验并取得成功
- 2023.03 -----获深圳市专精特新企业认证
- 2023.06 -----获登深圳市首批《智能机器人应用示范案例》
- 2023.07 -----获得深圳商报【协作机器人，为陪伴而生 华强北走出一头“大象”】专题报道
- 2023.07 -----第十五届中国深圳创新创业大赛暨华秋第九届中国硬件创新创客大赛华南为赛区第四名
- 2023.08 -----世界上最小的七自由度协作机械臂myArm面市
- 2023.09 -----入选英伟达“ 2023英伟达NVIDIA INCEPTION SHOWCASE ”
- 2023.09 -----获“ 深圳市委组织部感谢信 ”
- 2023.10 -----第五届中国芯大赛总决赛二等奖
- 2023.10 -----第十二届中国创新创业大赛总决赛高端制造优秀企业
- 2023.11 -----获中国国际贸易促进委员会 《出口商品品牌证明书》
- 2023.11 -----获第九届中国硬创赛全国三强
- 2023.11 -----举办大象机器人&OpenCV AI机器人实战训练营
- 2023.12 -----树莓派官方杂志MagPi 137期发布大象机器人myCobot 280桌面六轴协作机械臂测评文章
- 2023.12 -----获得leaderobot2023年度机器人行业科研贡献奖&最具投资价值企业排行榜TOP50
- 2023.12 -----获得首届全国先进计算技术创新大赛总决赛二等奖
- 2023.12 -----万元级人形机器人水星Mercury系列面市

3. 相关链接

购买链接

- 淘宝：https://shop504055678.taobao.com
- Shopify： https://shop.elephantrobotics.com/
- 速卖通：https://elephantrobotics.aliexpress.com/store/1101941423

  其他信息

  - 官网： https://www.elephantrobotics.com
  - 视频
- Bilibili： https://space.bilibili.com/2126215657
- Youtube： https://www.youtube.com/c/Elephantrobotics
  - Facebook： https://www.facebook.com/mycobotcreator/
  - Linkedin： https://www.linkedin.com/company/18319865
  - X (Twitter)： https://twitter.com/CobotMy
  - Discord： https://discord.gg/2MAherp7nt
  - Hackster： https://www.hackster.io/elephant-robotics

4. 联系我们

> 我们的工作时间为中国工作日，北京时间上午 10 点至下午 6 点。

- 如果您有任何其他问题，请通过以下方式联系我们。
  E-mail：

```
support@elephantrobotics.com
```

- 如果您有购买意向或任何参数问题，请发送电子邮件至此邮箱。 [E-mail](sales@elephantrobotics.com) :

```
sales@elephantrobotics.com
```

- 如果列出的问题无法帮助您解决，而您又有更多的售后问题，请向本邮箱发送电子邮件。
  E-mail :

```
support@elephantrobotics.com
```

我们将在 1-2 个工作日内给予答复;

微信:
我们只为通过微信购买 myArm M&C 系列产品的用户提供一对一服务。

致谢

我们在这里表达我们对所有参与开发、测试和完善 mercury 系列产品的人员的深深感激。每一个细节的打磨和每一个功能的创新都离不开背后团队的辛勤工作和献身精神。

特别感谢：

研发团队：感谢你们的创新思维和无数个日夜的努力，将复杂的技术转化为用户友好的产品。
质量保证与测试团队：你们对每一个细节的严格把控保证了我们产品的可靠性和用户的最终体验。
客户支持团队：感谢你们为我们的用户提供专业的支持，帮助他们解决使用过程中的每一个问题。
合作伙伴与供应商：你们的支持和服务对产品的成功至关重要。感谢你们的高质量原材料和部件，以及时刻准备支持的态度。
投资者和顾问：没有你们的信任和资金支持，我们无法将这些创新带到市场。你们的洞察和指导也一直是我们前进的动力。

用户感谢：

我们特别感谢选择和信任 mercury 系列产品的每一位用户。你们的反馈和建议是我们不断前进和改进的动力。我们承诺将继续倾听你们的声音，不断优化我们的产品和服务。

未来展望：

我们期待与所有的利益相关者一起，继续在机器人技术的道路上探索和进步。让我们携手创造更多的可能，为世界带来更大的便利和创新。

← 上一章