计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(7):70−79 [doi: 10.15888/j.cnki.csa.008022]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Ubuntu 下 ROS 变种 ROCOS 的系统架构①

杜晋华[1], 杜　刚[1], 张　虎[2]

[1](中国地质大学 (北京) 信息工程学院, 北京 100083)
[2](湖北大学 物理与电子科学学院, 武汉 430062)
通讯作者: 杜晋华, E-mail: dujinhua@cugb.edu.cn

摘　要: 伴随人工智能等相关技术的进步, 智能机器人技术得到极大提高. 其软件上的日渐成熟和硬件设施的不断完善对研究和开发机器人操作系统和体系结构提出新的需求. 本文针对目前广泛使用的传统机器人操作系统 ROS 进行介绍和分析, 并对比研究了新型机器人操作系统 ROCOS 在系统架构方面对传统 ROS 的改进和优化, 详细介绍了两者的组织框架, 并通过仿真实验实际测试了两者框架的合理性与使用效果, 对后续新型机器人操作系统的研发具有借鉴意义.

关键词: ROS; ROCOS; 系统架构; 机器人操作系统; Ubuntu

引用格式: 杜晋华,杜刚,张虎.Ubuntu 下 ROS 变种 ROCOS 的系统架构.计算机系统应用,2021,30(7):70–79. http://www.c-s-a.org.cn/1003-3254/8022.html

System Architecture of ROS Variant ROCOS under Ubuntu

DU Jin-Hua[1], DU Gang[1], ZHANG Hu[2]

[1](School of Information Engineering, China University of Geosciences (Beijing), Bejing 100083, China)
[2](School of Physics and Electronic Sciences, Hubei University, Wuhan 430062, China)

Abstract: With the progress in artificial intelligence and other related technologies, the intelligent robot technology has been greatly advanced. Its increasingly mature software and hardware facilities put forward new requirements for the research and development of the robot operating system and architecture. In this paper, a traditional widely-used robot operating system, ROS, is introduced and analyzed, and the improved robot operating system, ROCOS, is compared with ROS to analyze the optimized system architecture. Additionally, the organizational frameworks of the both are detailed, and the simulation experiment is conducted to test the rationality and performance of the frameworks. The presented work can serve as a reference for the subsequent research and development of new robot operating systems.

Key words: ROS; ROCOS; system architecture; Robot operating system; Ubuntu

近年来, 伴随人工智能技术和传感技术等新型技术的快速发展, 智能机器人技术获得了突飞猛进的进步. 与之对应的是机器人相关体系结构和操作系统迎来了发展的黄金时期, 出现了以 ROS 为代表的专用于机器人的操作系统[1].

鉴于 ROS 操作系统在架构的模块化与层次化上的良好表现, 国内外很多学者都基于 ROS 进行了有关研究与技术实现, 如基于 ROS 实现机器人的各子功能部件的三维建模与开发[2] 有效降低了工作复杂度、使用 ROS 作为机器人控制的嵌入式底层避免了对策略核心的重复设计与实现[3]、通过 ROS 提供的各种功能包来进行二次开发[4] 以期减少成本投入以及使用

① 基金项目: 中国地质大学 (北京) 大学生创新创业项目 (202011415049); 中国地质大学 (北京) 实验技术研究与应用项目 (354711002)
Foundation item: College Student Innovation Training Program of China University of Geosciences (Beijing) (202011415049); Experimental Technology Research and Application Project of China University of Geosciences (Beijing) (354711002)
收稿时间: 2020-10-30; 修改时间: 2020-12-02; 采用时间: 2020-12-25; csa 在线出版时间: 2021-06-30

ROS 进行相关仿真实验[5] 取代实际重复性或长周期的实地实验, 并能保证更高精度的数据获取等. 除此之外, 国外研究大多集中于具体 ROS 应用的实现, 包括但不限于基于 ROS 的仿人双臂机器人 PR2[6] 的实现、汽车自动驾驶路径规划与导航[7] 的实现、家用服务型机器人系统[8] 的实现、医疗辅助式机器人交互逻辑[9] 的实现等.

除直接基于 ROS 操作系统进行机器人方面的工作外, 也有部分开发人员考虑借鉴 ROS 的逻辑, 为专门化的机器人提供相应的操作系统, 这种操作系统在保留 ROS 的主要功能和特点的前提下, 都会在某一特定方面做出一定优化, 以期更好地控制与服务于相关机器人. 其中有代表性的是 ROCOS 操作系统, 主要实现了多台机器人的配合调度.

本文将对 ROS 和其变种 ROCOS 做简要介绍, 并详细对比分析两者的架构逻辑, 希望对有关机器人系统研究方面的人员提供一定的参考价值.

1 系统简介

1.1 机器人操作系统 ROS

ROS (Robot Operating System), 即机器人操作系统, 是用于编写机器人软件程序的一种具有高度灵活性的软件架构, 是一个适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, 底层设备控制, 常用函数的实现, 进程间消息传递, 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.

在它诞生之前, 很多专家致力于实现一个能够自我感知、自我导航和自我控制的复杂机器人设计. 过程中, 大家发现很多现有资源难以共享或者直接使用, 所以都认为机器人研究需要一个开放式的协作框架, 用以整合和提高各种资源的利用率. 所以在这样的需求下, 2007 年 ROS 自斯坦福大学诞生, 并在大量研究人员的共同努力下, 逐渐发展和完善. ROS 发展历史如图 1.

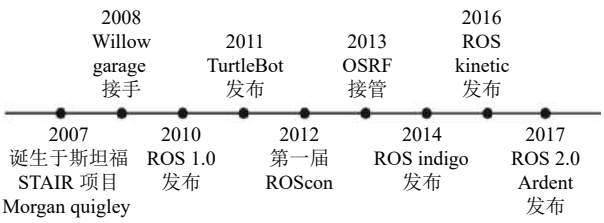


图 1 ROS 发展历史

目前, ROS 广泛应用于机器人领域, 主要实现机器人的建模、感知、导航和规划, 并能直接和机器人连接通讯, 以实现对其物理实体的控制.

1.2 机器人操作系统 ROCOS

ROCOS 是世界杯机器人操作系统 (RObotCup Operating System) 的英文缩写, 是以 ROS 操作系统为原型, 剔除不必要的功能分支, 专用于多机器人调度的软件架构.

它源自机器人世界杯 RobotCup, 一个为促进人工智能和机器人等相关领域研究的国际项目. 该项目是以足球机器人为主要研究平台, 以推进人工智能和机器人研究的国际比赛, 旨在研究如何在实时性强, 即高动态性和高对抗性的环境中, 调度多台机器人完成指定的工作任务, 和实现多机配合的智能化. 图 2 是 RobotCup 赛场实况.

图 2 RobotCup 赛场实况

ROCOS 最先由卡内基梅隆大学 (CMU) 在 1997 年发起构建, 其核心思路是在控制子系统的实现时按照 STP (Skill-Tactic-Play) 框架进行分层开发, 之后于 2012 年对其改版, 提出 SSPS (Skill-State-Play-Selector) 框架,有效提高了各机器人之间的配合度, 在响应机器指令方面也有很大提升.

与此同时, 世界上其他机器人强校如康奈尔大学等也在积极引进 ROCOS 并投入到对其优化与研发中. 我国浙江大学也在持续跟进, 其控制学院工业控制国家重点实验室于 2003 年就着手 ROCOS 的研究, 2020 年已形成比较完善的 ROCOS 版本, 并对 ROCOS 实效做了相应的测试与研究[10,11]. 表 1 说明了浙江大学现有 ROCOS 在世界机器人比赛上的优异表现.

表 1 RoboCup SSL 荣誉榜

Rank	Team	Country	1st
1	CMμs	USA	5(97,98,06,07,15)
2	ZJUNlict	China	4(13,14,18,19)
3	Cornell Big Red	USA	4(99,00,02,03)
4	Skuba	Thailand	4(09,10,11,12)

2 系统架构

2.1 ROS 架构

从硬件实现角度, ROS 系统可分为 3 个层次: OS 层、中间层和应用层. 其中 OS 层为底层, 主要使用 Linux 系统实现 (如 Ubuntu), 中间层实现 ROS 核心通信机制以及功能库, 应用层通过运行 ROS Master (管理者) 来负责系统正常运行.

具体到中间层的实现, 又可将 ROS 架构细分为计算图层、文件系统层和开源社区层, 如图 3.

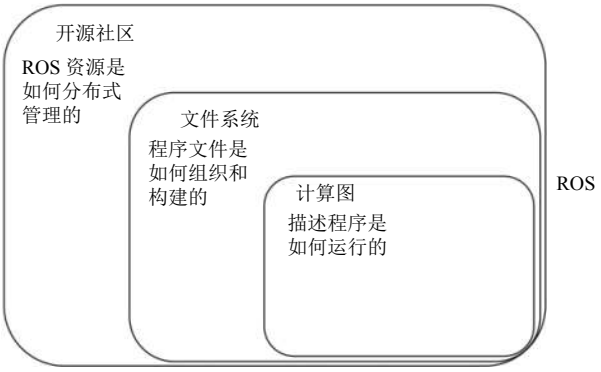


图 3 ROS 架构

2.1.1 计算图层

ROS 系统软件的功能模块以节点为单位独立运行, 通过端对端的拓扑结构连接. 所以 ROS 引进计算图来描述这种拓扑结构, 进而体现程序的运行方式.

计算图的构成方式是: ROS 创建一个连接所有进程 (节点) 的网络, 其中的任何进程 (节点) 都可以访问此网络, 并通过该网络与其他进程 (节点) 交互, 获取其他进程 (节点) 发布的信息, 并将自身数据发布到网络上. 其中, 构成计算图网络的各个节点 (node)、主题 (topic)、服务 (server) 等都有唯一的名称做标识.

1) 节点: 是主要的计算执行进程, 功能包 (在第 2.1.2 节中详细定义) 中创建的每个可执行程序在被启动加载到系统进程中后, 该进程就是一个 ROS 节点, 如图 4 中的 node1、node2等都是节点. 节点都是各自独立的可执行文件, 能够通过主题 (topic)、服务 (server) 或参数服务器 (parameter server) 与其他节点通信. ROS 通过使用节点将代码和功能解耦, 提高了系统的容错力和可维护性.

2) 消息: 节点通过消息 (message) 完成彼此的沟通. 它是 ROS 中一个进程 (节点) 发送到其他进程 (节点) 的信息. 消息类型是消息的数据结构, ROS 系统提供了很多标准类型的消息可以直接使用, 如果要使用非标准类型的消息, 需要进行自定义.

3) 主题: 每个消息都必须发布到相应的主题 (topic), 通过主题来实现在 ROS 计算图网络中的路由转发. 节点发送数据被称为该节点正在向主题发布消息. 节点可以通过订阅某个主题, 接收来自其他节点的消息. 通过主题进行消息路由不需要节点之间直接连接, 这意味着发布者节点和订阅者节点之间不需要知道彼此是否存在, 从而保证发布者节点与订阅者节点之间的解耦合.

4) 服务: 在一些特殊的场合, 节点间需要点对点的高效率通信并及时获取应答, 此时需要用服务的方式进行交互. 提供服务的节点被称为服务端, 向服务端发起请求并等待响应的节点被称为客户端, 客户端发起一次请求并得到服务端的一次响应就完成了一次服务通信过程. 如图 4 中 node1 向 node3 发起一次请求, 并得到 node3 返回给 node1 的响应.

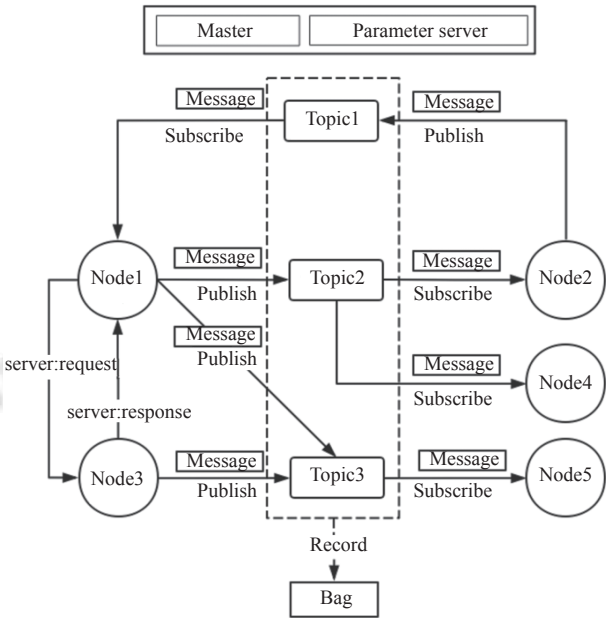


图 4 计算图示例

2.1.2 文件系统层

与其他操作系统类似, 一个 ROS 程序的不同组件需要被放在不同的文件夹下进行管理, 而这些文件夹则根据不同的功能来对文件进行组织, 它们构成了 ROS 的文件系统.

在文件系统中, 最基本的管理单元就是功能包. 功能包是 ROS 中软件组织的基本形式, 一个功能包具有

用于创建 ROS 程序的最小结构和最少内容, 它可以包含 ROS 运行的进程 (节点) 和配置文件等.

一个功能包可以按照图 5 的方式分为多个部分, 其中具体到每个子部分如下文.

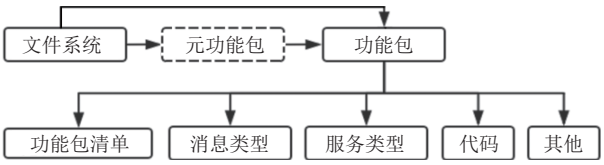


图 5 ROS 文件系统中的功能包

1) 功能包清单: 主要用于记录功能包的信息以及发行者、功能包之间的依赖关系以及编译的相关信息等. 在功能包中扮演管理者的角色.

2) 消息类型: 用于描述 ROS 采用话题通信时消息的类型, 通过该文件可了解这个功能包的相关通信接口.

3) 服务类型: 用于描述 ROS 采用服务通信时服务的类型, 通过该文件也可了解这个功能包的相关通信接口.

4) 代码: 节点是通过相应的编程语言进行编写的, 其代码称作源代码, 存放于特定文件夹中.

其中, 由功能包还可以组成更大的功能包, 被称之为元功能包, 用以实现更大、更复杂的功能.

而在元功能包之上, 还有一层文件夹, 被称作工作空间. 工作空间是一个包含功能包、可编译源文件和编译包的文件夹, 当用户想同时编译不同的功能包或保存本地开发包时必须使用这一机制. 用户可根据实际需要创建多个工作空间, 在每个工作空间中开发不同用途的功能包.

如图 6 所示, 工作空间通常由 3 种文件夹构成.

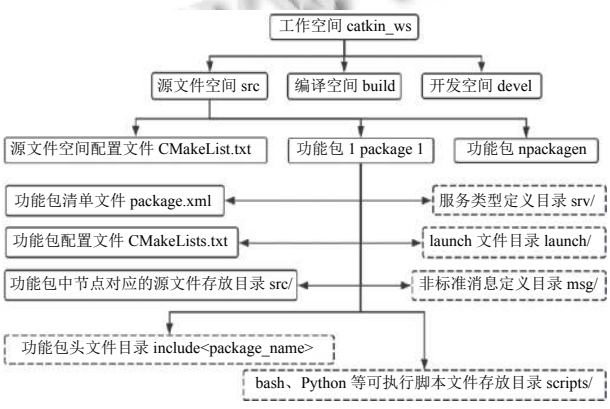


图 6 ROS 文件系统中的工作空间示例

1) src 源文件空间: 该文件夹放置各个功能包和用于这些功能包的 CMake 配置文件 CMakeLists.txt. 由于 ROS 中的源码采用 catkin 工具进行编译, 而 catkin 工具基于 CMake 技术, 所以在 src 源文件空间和各个功能包中都会有 CMakeLists.txt, 它起编译配置的作用.

2) build 编译空间: 该文件夹放置 CMake 和 catkin 编译功能包时产生的缓存、配置、中间文件等.

3) devel 开发空间: 该文件夹放置编译好的可执行程序, 这些可执行程序不需要安装就能直接运行. 一旦功能包源码编译和测试通过后, 可以将这些编译好的可执行文件直接导出与其他开发人员分享.

2.1.3 开源社区层

开源社区层是位于 ROS 架构最外面的层次 (注意这里讨论的范围还在中间层, 其实开源社区上还有应用层). 由于 ROS 的开源让很多人得以共享资源, 这个共享资源的平台称之为 ROS 的开源社区, 如图 7.

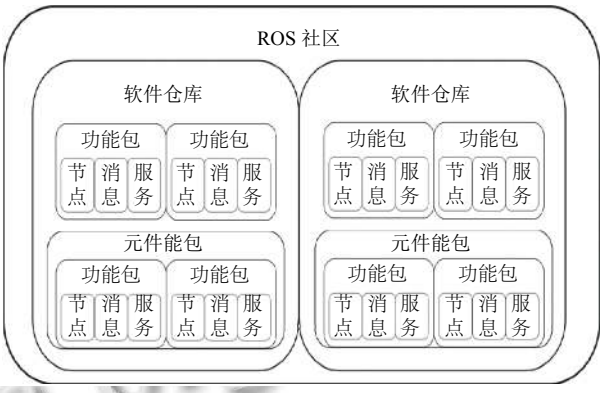


图 7 ROS 开源社区构成

ROS 的开源社区主要由网站、博客和代码资源构成. 因为 ROS 有各种功能的功能包, 但 ROS 安装时只会安装固定的一部分, 所以用户搭建自己的机器人时, 如果需要特定功能的功能包, 可以自己写功能包源码, 也可以从开源社区中寻找, 采用软件源的形式下载别人的功能包来参考使用.

具体的开源社区功能模块构成如下:

1) 发行版 (Distribution): ROS 发行版包括一系列带有版本号、可以直接安装的功能包.

2) 软件源 (Repository) : ROS 依赖于共享网络上的开源代码, 不同的组织机构可以开发或者共享自己的机器人软件.

3) 文档社区 (Wiki): 记录 ROS 信息文档的主要论坛.

4) 邮件列表 (Mailing list): 交流 ROS 更新的主要渠道, 同时也可以交流 ROS 开发的各种疑问.

5) 问答社区 (Answers): 咨询 ROS 相关问题的网站.

6) 博客 (Blog): 发布 ROS 社区中的新闻[12]、图片、视频等内容.

2.2 ROCOS 架构

因为 ROCOS 操作系统是在 ROS 系统的基础之上做出的单方向 (多机调度) 的优化. 所以从实现角度看, 其架构与 ROS 类似, 也可分为 3 个层次: OS 层、中间层和应用层. 其中 OS 层为底层, 主要使用 Linux 系统实现 (如 Ubuntu), 中间层实现 ROCOS 核心通信机制以及功能库, 应用层主要通过图形界面的模式为用户提供服务.

具体到中间层的实现, ROCOS 更加关注具体的机器人仿真与通讯细节, 可将 ROS 架构细分为后端策略层、前端控制层、文件系统层和开源社区层, 如图 8.

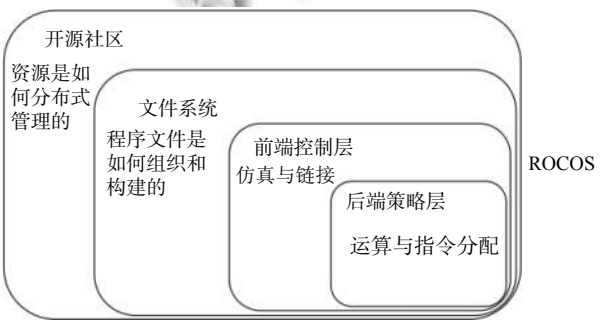


图 8 ROCOS 架构

2.2.1 后端策略层

后端策略层是 ROCOS 用于实现机器人操控逻辑、处理 I/O 信息 (如视觉信息、从机器人传回的通讯信息、传感器信息等) 和形成指令的控制中心.

在图 9 中, 机器人的正常运行有赖于多个相互关联系统的信息交互与功能配合. 具体包括: 视觉子系统、决策子系统、通讯子系统、机器人硬件子系统以及场地辅助子系统等. 其中, ROCOS 操作系统装载于策略机上, 由其后端策略层处理从其他子系统搜集到的实时信息, 并进行整合、处理与计算, 之后依靠代码逻辑做出相应的决策和命令, 通过通讯机制传输给机器人, 完成逻辑层与物理层之间的配合.

为完成相应功能, 后端策略层需要依赖外置功能包, 并使用合理的框架对功能进行解耦和处理.

涉及到的功能包与运行环境具体包括:

1) 开发环境: 为有效组织代码逻辑、合理预留前端控制层需要的接口以及确保应用层较好的可视化效果, 选择具有优良跨平台特性、面向对象的、含丰富 API、支持 OpenGL 的 QT 作为默认集成开发环境.

2) 开发语言: 在 ROCOS 中, 绝大部分底层逻辑代码采用 C++11 规范完成开发. 而考虑到多机器人的调度, 使用具有可扩展、简单高效、与平台无关等特性的 LUA 语言实现相应的功能. (具体实现中需要引入第三方软件包 tolua++)

3) 相关库: 后端策略层需要对大批量、实时性数据进行处理, 有赖于 C++里支持线性代数运算、矩阵和矢量运算、数值分析及其相关的算法的开源模版库 Eigen. 另外, 为实现后端策略层和前端控制层、前端控制层和通讯子系统的数据传输, 需要引入与平台和语言无关、可扩展且轻便高效的序列化数据结构协议 Protobuf.

图 9 ROCOS 在整个大型系统中的位置

具体而言, 后端策略层通过 C++和 LUA 两种计算机语言的有机结合, 用 C++进行各种基础算法、动作任务的编写, LUA 进行上层任务的分配, 构建出一个能使机器人从完成一个简单动作到多个机器人进行相互协调配合的框架.

框架结构如图 10 所示.

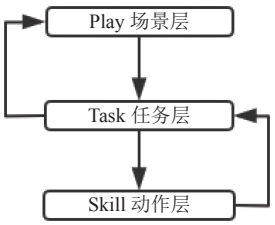


图 10 ROCOS 后端策略层分层

1) 使用 C++完成底层 skill 动作层的开发: 该层主要完成单台机器人的单个动作, 与硬件实现紧密相关, 如实现到达指定点、转体等.

具体实现框架为:

```
1. #include "src/utils/PlayerTask.h" ……
2. extern "C"_declspec(dllexport) PlayerTask
player_plan(const WorldModel* model, int robot_id);
3. enum FourQuadrant{LeftUp,……}area;
4. PlayerTask player_plan(const WorldModel*
model, int robot_id){
5. PlayerTask task;
6. ……
7. if (ball.y > 0) {
8.  if (ball.x > 0) {area = RightUp;}
9.  else {area = RightDown;}
10. }
11. else {……}
12. switch (area){
13. case LeftUp:
14.  task.orientate = (ball - goal).angle();//dir
15.  task.target_pos = point2f(half_x, -half_y);
16.  break;
17. case LeftDown:
18. ……}
19. return task;}
```

2) 使用 LUA 完成 task 任务层的开发: 该层主要调用多个 skill 函数, 并将其封装为一个完整的 task 任务, 用以将该任务分配给单台机器人执行, 如足球机器人执行到达指定点拿球的任务 (具体包含到达拿球点 skill、控球 skill、检测是否拿球 skill 等).

具体实现框架为:

```
1. gPlayTable.CreatePlay{
2. firstState = "halt",
3. switch = function()
4. return "halt"
5. end,
6. ["halt"] = {
7. ["Leader"] = task.stop(),
8. ["Special"] = task.stop(),
9. match = "[LS]"
10. },
11. name = "NormalPlayV1",
12. applicable = {exp = "a", a = true},
13. attribute = "attack",
14. timeout = 99 999
15. }
```

3) 使用 LUA 完成 play 场景层的开发: 该层主要调用多个 task 对象, 并将其封装为一个完整的 play 场景, 用以对多台机器人进行任务分配, 使之并行执行多个任务, 如多台家用机器人处于清扫场景 (扫地机器人分配地面清扫任务、擦窗机器人分配玻璃清洁任务等). 其实现框架与 task 层类似.

如果对策略层做进一步的细化, 可将其分为 6 个工作层:

1) 视觉和传感器的信息处理: 将视觉系统传过来的信息进行分析整合, 过滤重复内容. 另外, 需要对从机器人本体上传感器 (如红外传感器) 获得的工作信息进行处理, 为后续指令的下达提供依据;

2) 运动预测: 通过保留过去若干秒内的图像信息, 将位置信息转化为速度信息, 根据数学计算来预测环境中各实体 (包括机器人在内) 位置, 该工作也为机器人合理的路径规划打下基础;

3) 路径规划: 实现机器人的动态避障, 在完成移动任务的同时确保机器人不与各类静止、运动物体发生严重碰撞;

4) 多层决策协作: 实现从一个机器人完成一个动作到多台机器人协同运作的多决策层的有机配合;

5) 场景实现: 包括场景动态变换后的任务更新以及动态分配;

6) 各种相关的算法: 包括基于硬件提供功能的抽象与二次开发.

2.2.2 前端控制层

前端控制层是 ROCOS 用于仿真机器人实时场景、提供应用层控制接口以及与通信子系统连接配置等功能的集成.

图 11 是足球机器人前端控制层在应用层的图形页面, 其构成包括左部的环境仿真与右部的控制面板, 用以实现对机器人控制的可视化处理.

具体使用过程中, 需要在控制面板中设置相关参数、选择研究模式. 如果场地子系统可以提供支持, 可选择实地模式, 并在结合下文的通讯子系统后, 完成实地图像在系统上的二维仿真建模与控制; 如果场地子

系统不提供支持, 用户可以脱机进行研究, 在 ROCOS 中填充自定义的程序实现文件, 系统将解析用户的文件并在环境仿真界面进行仿真运算, 可获得与实地模式同等的研究效果.

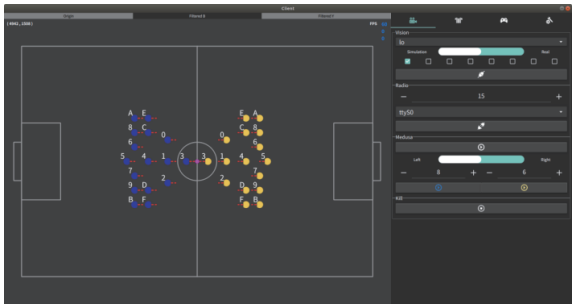


图 11 ROCOS 前端控制层在应用层的图形页面

图 12 是 ROCOS 前端控制层功能的另一方面体现, ROCOS 搭载的策略机可直接和信号发射机连接, 通过无线、蓝牙等技术实现与机器人之间的通讯. 其具体实现逻辑仿照无线电系统和无线网络系统的实现模式: 当机器人与策略机连接在同一网段, 或两者频率收发器均设置为相同的频点, 即可实现自主式非人工连接.

图 12 ROCOS 前端控制层在应用层的图形页面

为完成相应功能, 前端控制层也需要依赖外置的功能包, 并使用合理的框架对功能进行解耦和处理.

值得注意的是: 由于前端控制层的框架不是特别清晰, 也需要之后随着 ROCOS 的版本迭代逐步优化, 这里仅对其依赖的功能包做进一步的说明:

1) 仿真: 为实现机器人环境与实体的三维建模与实时仿真, 需要引入开源的图形函数库 OpenGL 进行场景建模, 也需要引入开源的动态引擎库 ODE 提供相应的支持.

2) 驱动: 需要为 ROCOS 配置相应从其他子系统获取信息和进行 I/O 管理的软件, 如与信号通讯相关的发射机驱动、加密狗驱动; 与视觉信息捕获相关的显卡驱动等.

2.2.3 文件系统层

ROCOS 的文件系统层和 ROS 没有特别大的差别, 而且在对文件系统管理的文件树结构也大致相同, 均包含基本的文件夹架构:

1) bin: 该文件夹保存可执行的程序和命令, 大多是在 ROCOS 的终端界面可直接运行的命令行.

2) include: 该文件夹用于放置头文件. 由于 ROCOS 里面的某一具体的头文件属于某一具体的功能包, 如果一个功能包需要依赖另外的功能包, 自然必须包含另外功能包的头文件, 功能包的头文件在功能包安装的时候被直接存放于该文件夹.

3) lib: 主要存放一些.so 结尾的可执行程序.

4) etc: 主要存放 ROS 和 catkin 配置文件.

5) share: 放置 ROCOS 里面已经安装的功能包, 主要包含功能包的各种信息, 包括其接口、配置信息等. 值得注意的是, 如果用户需要真正操作机器人, 并不是直接在该文件夹下进行, 需要额外建立一个文件夹, 即上文提及的工作空间, 然后在工作空间下创建自己的功能包, 进行一系列开发工作.

2.2.4 开源社区层

由于 ROCOS 相对较为小众, 所以并不像 ROS 一样提供一个广泛开源的社区, 它主要依托 Git 实现代码开源与数据共享 (Git 是目前世界上最先进的分布式版本控制系统).

本文介绍的浙江大学实现的 ROCOS, 就在应用 Git 系统的主流版本管理网站 Github 上进行了开源[13], 目前有包括笔者在内的多个开发者参与到相关代码的研发与优化中.

3 仿真与实验

为检验 ROCOS 架构的合理性及其在单一实现方向上所展示出的强针对性, 分别就 ROS 系统和 ROCOS 系统进行仿真实验.

由于本文主要针对浙江大学实现的 ROCOS 做研究, 研究者位处系统的优化人员之列, 所以不能估计本 ROCOS 的构建成本. 从浙江大学实验室获悉的详细数据是: ROCOS 的初始架构时间花费不到半年, 相较 ROS 系统直接作为成品或工具而言, 其成本偏高. 但注意使用 ROS 的用户需要花费至少 1 到 2 年的时间学习 ROS 的基本技术, ROCOS 仅需 1 个月或者更短, 所以从总时间成本投入上, ROCOS 更胜一筹.

而通常情况下, ROCOS 是为专门化领域或专门性需求所构建, 所以与 ROS 普适性构建相比, 其与机器

人的适配性会更好.

基于上述原因, 本次实验以 SSL (Small Size League) 小型足球机器人 (其二维参数如图 13) 为研究对象, 在一台策略机上运行 Ubuntu 下 ROS 系统、一台策略机上运行 Ubuntu 下 ROCOS系统. 为保证实验不受无关因素干扰, 确保两台机器有相同硬件环境配置: Intel core i5-8300, 2.3 GHz, 四核, 内存 8 GB. 并同时接入 SSL_Vision 对应显卡, 加载相应视觉软件. 相关参数如图 14.

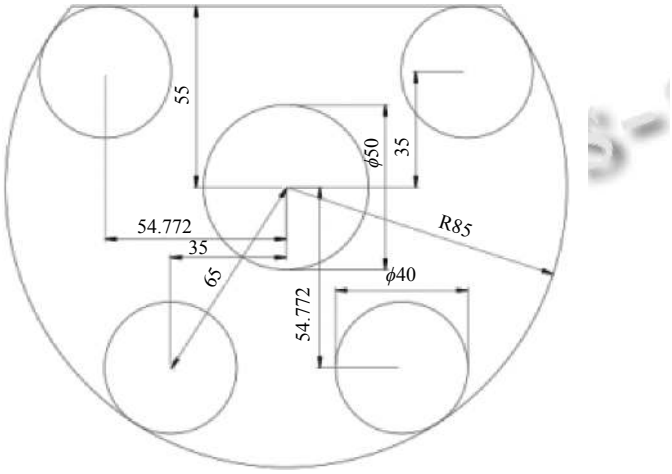


图 13 小型足球机器人二维参数

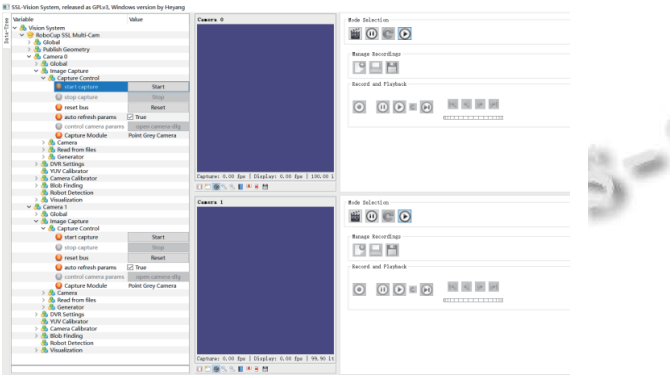


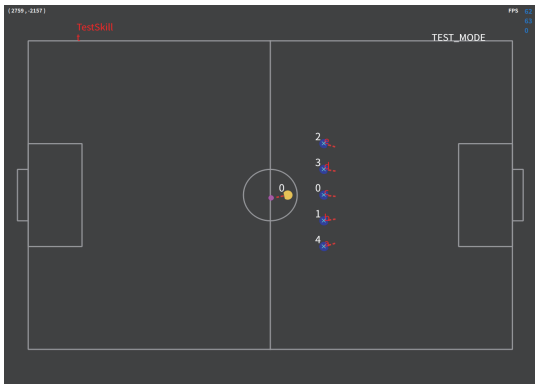
图 14 SSL_Vision 配置环境及参数

从图 14 可知, 初始条件下, 数据传输率均为 0. 在仿真实验中, 该值将指征系统的仿真效果以及稳定性.

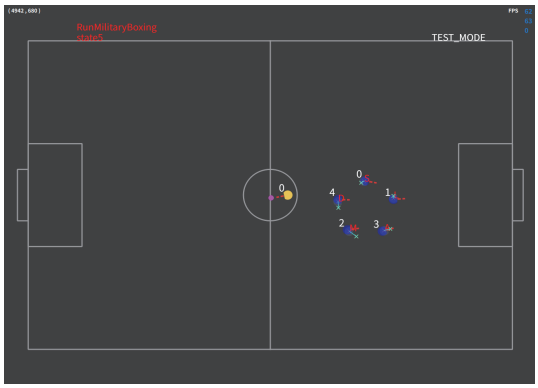
3.1 多场景适应性仿真

使用 5 台蓝色标识标记的足球机器人进行仿真模拟, 使用 LUA 脚本指定其分别实现多边形路径规划与队列路径规划, 检验数据传输率、ROCOS 计算速度与计算精度以及画面延时情况.

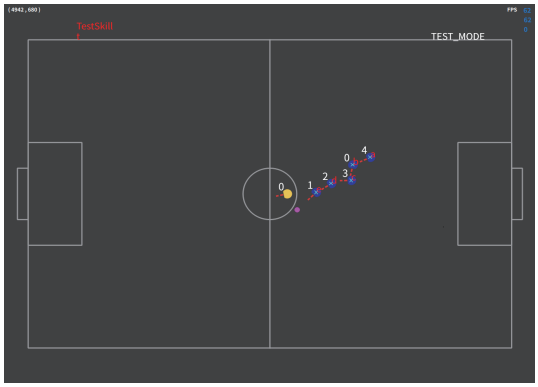
在图 15 中, 图 15(a) 是初始场景: 可以看到数据传输率稳定在 62 帧/s, 静态仿真效果良好. 图 15(b) 和图 15(c) 分别是多机路径规划场景中的多边形路径和队列路径动态仿真, 可以看到数据传输率仍旧维持在 62 帧/s, 所以系统稳定性非常好.



(a) 初始场景



(b) 多边形路径



(c) 队列路径

图 15 ROCOS 多场景适应性仿真图

使用命令行查看 CPU 使用情况可以看到 (ROCOS 没有提供独立任务管理器) 如图 16.

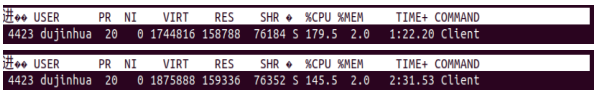


图 16 Ubuntu 下命令行任务管理器数据

初始静态仿真时刻 CPU 占有率为 179.5%, 而动态仿真时刻 CPU 占有率为 145.5% (注意这里占有率超过 100% 属系统问题, 相对占有率为准确值). 可以看出相对占有率不增反降, 主要原因在于静态仿真需要维持机器人仿真二维视图在可视时间内保持静止, 实际仍在进行计算, 所以 CPU 占用率较高.

为对比 ROCOS 的显著提升效果, 在 ROS 上做同组对比实验.

在图 17 中, 图 17(a) 是初始场景: 可以看到数据传输率稳定在 60 帧/s, 静态仿真效果与 ROCOS 持平. 图 17(b) 和图 17(c) 分别是多机路径规划场景中的多边形路径和队列路径动态仿真, 可以看到数据传输率下降较为严重, 低至 30 帧/s 左右, 所以系统稳定性不甚理想.

同样使用命令行查看 CPU 使用情况如图 18.

初始静态仿真时刻 CPU 占有率为 0.3%, 而动态仿真时刻 CPU 占有率为 70.9%. 可以看出虽然静态环境下, ROS 系统的效率极高, 但转为动态环境后 CPU 占有率提升 70.6%, 与 ROCOS 不增反降相比, 具有普适性的 ROS 系统在针对性领域效果不甚理想, 由此可以看出 ROCOS 系统所具有的强针对性: 在专一领域, 它的功能更加稳定且效果良好.

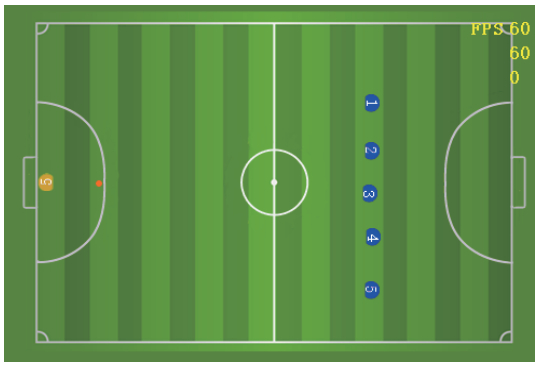
3.2 交互式仿真

在机器人领域, 给研究人员带来极大阻碍的就是在程序运行过程中, 无法实时查看相关参数. 此外, 在机器人运行过程中如果出现问题, 一般很难立即定位问题来源, 极大程度上制约了项目推进速度和效率.

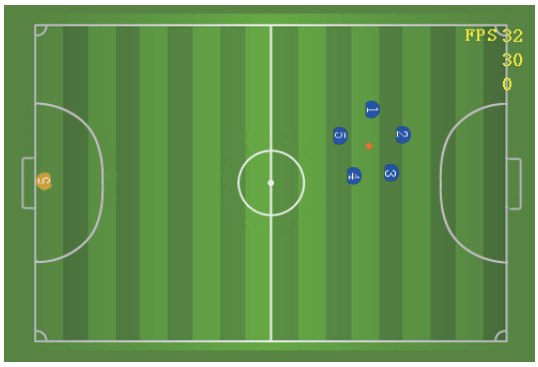
ROS 系统允许用户根据机器人实际情况, 借助系统提供的功能包自行定义交互式辅助软件. 仍以 SSL 为例. ROS 交互界面如图 19.

使用 ROS 提供的元功能包以及小型足球机器人硬件开发文档提供的相关接口, 实现上述界面, 可执行简单的机器人控制: 包括速度、自身行为、动作触发、多机选择. 但实际使用过程中, 一旦出现程序问题, 只能确定触发源为当前时刻的前一条指令, 无法确定具体原因.

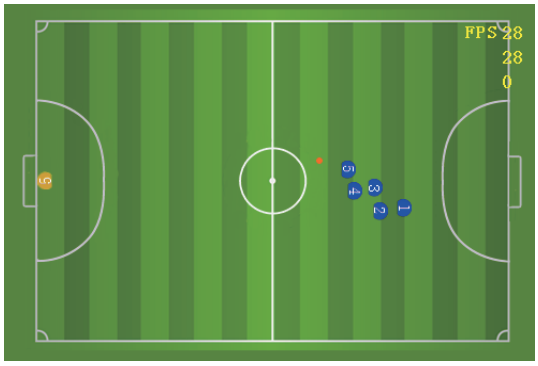
主要原因在于交互界面不灵活, 不能实现用户“所想即所得”, 即实时性动态调整可交互元素. ROCOS 可以实现上述需求.



(a) 初始场景



(b) 多边形路径



(c) 队列路径

图 17 ROS 多场景适应性仿真图

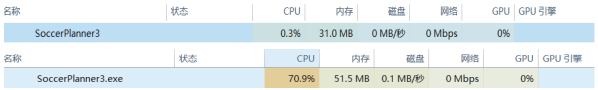


图 18 Ubuntu 下 ROS 自定义任务管理器数据

如图 20, 使用 ROCOS 搭建系统交互界面, 可在可视化窗口中有选择地或者全部显示用户需要获取的中间参数, 在控制面板中, 以列表形式提供各参数的设置

接口, 并允许用户在使用系统时动态调整列表项. 凡系统可从实体机器人硬件或者仿真后台程序中获取的参数项, 均可在控制面板动态定义.

与此同时, 在本次实验中, ROCOS 系统由于主要针对多机调度, 所以从图 20 中可以看出, 针对每一台机器人都可以获取其独立的参数. 为多机系统的设计与控制提供了极大方便.

图 19 ROS 交互界面

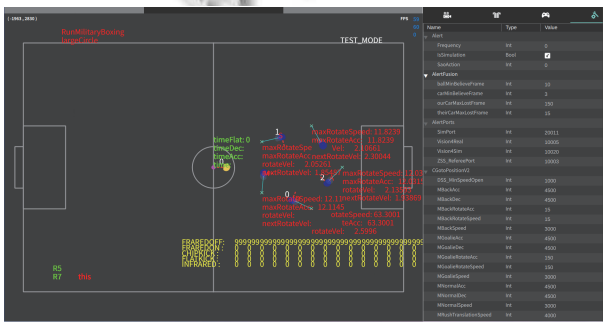


图 20 ROCOS 交互界面

通过以上对比, 可见 ROCOS 在交互方面也获得了较大程度的提升. 与 ROS 相比, 在强针对性领域, 显示出足够优势.

4 小结与展望

本文对基于 Linux 内核 (Ubuntu) 下的 ROS 操作系统和 ROCOS 操作系统分别进行了对比式的介绍和说明, 并对二者具体的系统架构进行了相对详尽的描述, 最后以对比实验的方式论证了 ROCOS 的系统特性与系统优势. 虽然目前 ROS 的适用范围仍在不断扩大, 但像 ROCOS 这样从 ROS 中发展而来的强针对性操作系统才刚刚起步. 它在保留 ROS 原有的点对点设计、多语言支持、架构精简、组件化工具包丰富以及免费且开源等特点的基础上, 更强调对某一特定场景的支持 (如本文提及的 ROCOS 就是对多机调度场景的支持), 显然它会比 ROS 更专一, 功能更稳定, 与特定场景的适配性更好.

截止目前, 由于 ROCOS 需要的成本投入远低于 ROS, 用户不需要学习 ROS 中大量复杂而多元的概念, 仅需要参照现有 ROCOS 实现模式, 去实现自己方向的 ROCOS 系统, 所以其成本更低. 基于 ROCOS 的显著优势, 随着近年来人工智能等技术的不断推向深入, 像 ROCOS 这样的操作系统在未来可能会更有市场. 虽然目前看来, 它还有很长的路要走.

参考文献

1 Martinez A, Fernandez E. ROS 机器人程序设计. 刘品杰, 译. 北京: 机械工业出版社, 2014.
2 刘昊. 基于 ROS 的移动操作机器人设计与开发 [硕士学位论文]. 上海: 上海交通大学, 2018.
3 张鹏. 基于 ROS 的全向移动机器人系统设计与实现 [硕士学位论文]. 合肥: 中国科学技术大学, 2017.
4 管晨曦. 基于 ROS 的室内巡检机器人系统设计与研究. 机电信息, 2020, (30): 128–129. [doi: 10.3969/j.issn.1671-0797.2020.30.068]
5 董学会. 基于 ROS 的移动服务机器人进门过程关键技术研究 [硕士学位论文]. 哈尔滨: 哈尔滨工业大学, 2016.
6 Cousins S. ROS on the PR2 [ROS Topics]. IEEE Robotics & Automation, 2010, 17(3): 23–25.
7 Gim S, Adouane L, Lee S, *et al*. Clothoids composition method for smooth path generation of car-like vehicle navigation. Journal of Intelligent & Robotic Systems, 2017, 88(1): 129–146.
8 Wang Y, Gou Y, Liu Q, *et al*. Research on intelligent shopping robot based on PLC. Proceedings of 2019 5th International Conference on Applied Materials and Manufacturing Technology. Singapore. 2019. 6.
9 Wang XM, Yu HL, Li KJ, *et al*. Study on force interaction system of upper limb rehabilitation robot. Proceedings of 2019 5th International Conference on Applied Materials and Manufacturing Technology. Singapore. 2019. 983–990.
10 Huang ZY, Chen LY, Li JC, *et al*. RoboCup SSL 2018 champion team paper. In: Holz D, Genter K, Saad M, *et al*., eds. RoboCup 2018: Robot World Cup XXII. Cham: Springer, 2019. 401–412.
11 Huang ZY, Chen LY, Li JC, *et al*. ZJUNlict extended team description paper for RoboCup 2019. arXiv preprint arXiv: 1905.09157, 2019.
12 http://www.ros.org/news.
13 https://github.com/Robocup-ssl-China/rocos.