面向安全属性的机器人运行时验证

楚 敏 邵振洲 王 瑞 李晓娟 张 芮
(首都师范大学信息工程学院 北京 100048)
(轻型工业机器人与安全验证北京市重点实验室(首都师范大学) 北京 100048)

摘 要 为了保证机器人操作系统(Robot operating system,ROS)的安全性,提出一个运行时验证框架 ROS-Monitor 来监控系统。将所有的监控信息分为节点消息和节点行为,并实现从用户自定义场景模型自动生成相应监控器的工具。实验证明了该方法的有效性。

关键词 运行时验证 机器人操作系统 安全属性 监控器 面向监控编程

中图分类号 TP3 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2022.03.051

SECURITY PROPERTY-ORIENTED ROBOT RUNTIME VERIFICATION

Chu Min Shao Zhenzhou Wang Rui Li Xiaojuan Zhang Rui
(*College of Information Engineering*, *Capital Normal University*, *Beijing 100048*, *China*)
(*Beijing Key Laboratory of Light Industrial Robot and Safety Verification*, *Capital Normal University*, *Beijing 100048*, *China*)

Abstract To ensure the security of robot operating system(ROS), this paper proposes a runtime verification framework named "ROS-Monitor" to monitor the security properties of the system. We classified all monitoring information into node message and node action. A tool was implemented to generate the corresponding monitor automatically from a user-defined scene model. The experiment shows the efficiency of this method.

Keywords Runtime verification Robot operating system Security property Monitor MOP

0 引 言

ROS 是一个用于机器人软件开发的开源操作系统[1]。ROS 为机器人应用提供了统一的接口标准,容易与其他的机器人软件平台集成。ROS 中执行某一任务的进程称为节点,机器人控制系统通常由多个节点组成。节点之间传递消息的主要方式是主题。主题通信方式通过发布者发布主题,订阅者订阅主题来进行。但是,在 ROS 的通信过程中,存在一些安全隐患。例如,节点的活动不受限制,任何节点都可以进行注册和主题发布。攻击者可以很容易地伪造节点来发布假消息,并窃听重要主题的消息。另一方面,通信数据缺乏实时监控,异常消息无法被捕获。因此,需要建立合理的安全监控机制保护节点通信过程的安全。这对于提高机器人系统的安全性和可靠性具有重要意义。

运行时验证[2]是一种轻量级的验证方法,采用形式化的方式对系统行为规范进行描述。它实时监控系统程序的运行轨迹,判断系统行为是否违反安全规范。该方法与模型检测[3]和定理证明方法[4]相比,复杂度更低。与传统的测试方法相比,运行时验证在对属性的描述以及对程序的监控方面具有灵活性和扩展性。运行时验证已经应用在 ROS 系统安全性保障中,Huang[5]提出了一个 ROSRV 框架,该框架可以监控消息数据,并可选地修改消息内容,但是并未对程序进行逻辑验证。Balsa-Comerón 等[6]认为 ROS 中的主要安全问题是明文通信,提出对通信数据使用对称加密算法。但是,密钥的管理和泄露会带来隐患。其他学者针对 ROS 通信安全提出了其他策略,如 White 等[7]通过 TLS(传输层安全加密协议)对两个通信应用程序之间的隐私和数据完整性进行了保护,Breiling 等[8]提出建立一个安全的传输信道,使 ROS 节点能够更加安全

收稿日期:2019 - 11 - 27。国家重点研发计划项目(2017YFB1301100)。**楚敏**,硕士生,主研领域:形式化验证。**邵振洲**,副教授。**王瑞**,教授。**李晓娟**,教授。**张芮**,硕士生。

的通信。

本文基于运行时验证技术,针对 ROS 操作系统在主题通信和节点活动方面存在的安全问题,提出了一个 ROS-Monitor 监控框架。该框架支持时序逻辑描述系统安全属性,根据自定义场景生成监控器来监控系统行为。若系统行为违背安全属性,则打印相关告警信息。

1　背景知识

1.1　ROS[1,9]的通信方式

主题通信是 ROS 节点进行消息通信的主要方式,图 1 展示了 ROS 主题通信的机制与过程[10]。它通过一系列 XML/RPC(一种用于远程过程调用协议)请求来启动,具体步骤为:1) 发布者通过 RPC 请求向节点管理器注册发布者信息,包括主题名 topic1 和 RPC 地址 addr1,并将这些信息存储到注册列表中。2) 订阅者向节点管理器订阅主题 topic1。3) 节点管理器根据订阅者所需的主题 topic1,在注册列表中进行查找。若匹配成功,节点管理器向订阅者传递发布者的 RPC 地址 addr1。4) 订阅者联系发布者获取主题连接请求。5) 在收到订阅者的连接请求后,发布者继续通过 RPC 请求向订阅者确认连接信息,包括发布主题的 TCP 地址 addr2。6) 订阅者接收到确认消息后,通过 TCP 请求连接到发布者地址 addr2。7) 二者成功建立 TCP 连接,订阅者开始接收发布者发出的消息数据。

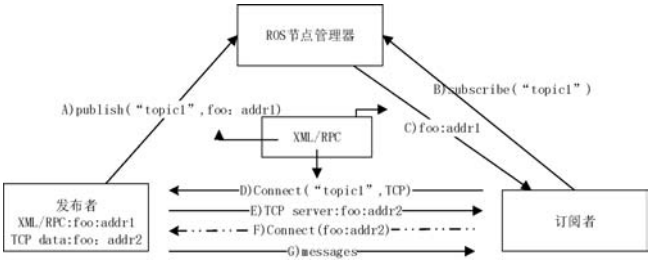

图 1　ROS 通信过程

1.2　JavaMOP[11]

运行时验证系统的结构[12-13]如图 2 所示,它由运行的程序和监控器组成。监控器通过分析程序的多个执行路径,将它们转换为相应事件,并结合给定的属性,验证系统行为是否满足属性,对运行结果作出判定。

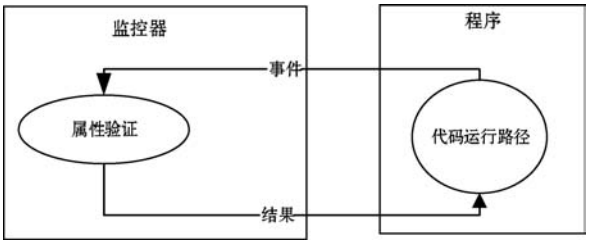

图 2　运行时验证结构

运行时验证的目的是实时监控系统行为,验证系统的正确性[2]。通常使用面向监控的编程(Monitoring-oriented Programming, MOP)[14]技术,这是一种扩展了运行时验证技术的轻量级形式化方法。MOP 将属性编织到原始系统中,并根据指定的属性自动生成监控器。监控器监视程序在运行期间的行为,当违反规范时将触发用户定义的操作。JavaMOP[10,15-16]是一个针对 Java 的监控编程开发工具,具有可插拔性和灵活性。JavaMOP 规范主要包括变量(var)、事件(event)、属性(property)和处理程序(handler)。

1) 变量。变量指的是待监控的程序变量。

2) 事件。事件是目标程序执行期间发生的操作。它对应于监控程序中被监控变量的更新,或者被监控方法的调用。事件的声明遵循 AspectJ 语义,事件被编织到 Java 程序中,被用来检查是否符合属性规约。

3) 属性。属性包括逻辑类型和逻辑语义,JavaMOP 插件支持多种逻辑类型,如 FSM(有限状态机)、LTL(线性速度逻辑)、ERE(扩展正则表达式)、CFG(上下文无关文法)等。系统安全属性可以用时态逻辑表示,本文主要以 LTL 逻辑[17-18]语句进行介绍。LTL 语法由原子命题和运算符组成,运算符包括[ ],[*], < >, < * >, o, (*), !, not, U, S, And, Xor, Or, =>(蕴含)[19]。$T = t_1, t_2, \cdots, t_i, \cdots, t_n$ 是一组原子命题,LTL 语法及时态运算符解释如下:

$\mu, v ::= t_i \mid [\,]\mu \mid [*]\mu \mid (*)\mu \mid <>\mu \mid <*>\mu \mid o\mu \mid \mu S v \mid \mu U v$

- $[\,]\mu$:μ 在所有的时间内都是成立的。
- $[*]\mu$:μ 在过去的时间内都是成立的。
- $(*)\mu$:μ 在之前的时间是成立的。
- $<>\mu$:μ 在将来的某个时间点成立。
- $<*>\mu$:μ 在过去的某个时间点成立
- $o\mu$:μ 在下一个时间点成立。
- $\mu S v$:自从 μ 成立的时间点起,v 在过去的某个时刻已经成立了。
- $\mu U v$:v 在未来的某个时间点成立,μ 在 v 成立之前已经成立了。

4) 处理程序:处理程序由任意的 Java 代码组成,用户可以自定义操作,该操作可以是记录异常或信息更正的任何代码。以@ fail、@ violation 或@ validation 开头,分别代表自动机的不确定、违背和验证三种状态。

2　ROS-Monitor 运行时验证框架

ROS-Monitor 是一个针对机器人操作系统的运行时验证框架。如图 3 所示,我们设计了一个能够监控

所有节点活动和节点消息的监控中心节点,并将所有的待监控信息存放到事件消息中枢。同时设计了一个代码生成器,可以根据自定义场景描述自动生成特定的监控器。最后通过运行监控器,验证系统的安全属性,若系统行为违反安全属性逻辑,则发出违背警告。

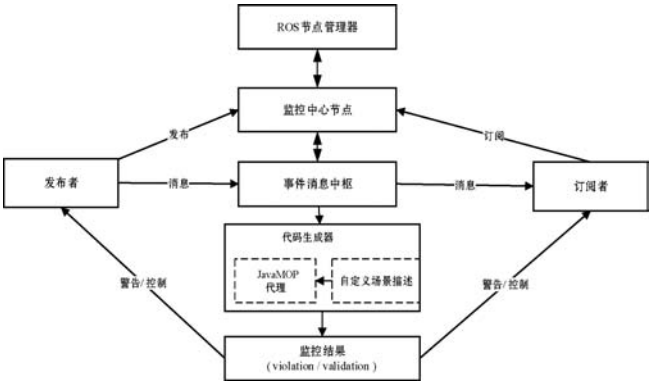

图 3 ROS-Monitor 框架

2.1 监控中心节点

ROS 主题通信机制详见 1.1 节,节点管理器默认启动在 11311 端口,系统各节点连接到该端口进行主题发布等相关操作,节点管理器记录各节点发布和订阅的信息。为了获取节点的所有信息,我们根据 ROS 的 XML/RPC 的通信协议,设计了一个支持节点注册、主题发布订阅等功能的监控中心节点,可将其理解为节点管理器的代理。本框架将节点管理器启动在 11111 端口,监控中心节点启动在 11311 端口。监控中心节点一方面与节点管理器进行通信,另一方面与系统各功能节点进行通信,因此可以获取 ROS 系统中所有待监控的信息。

监控中心节点需要监控节点活动和节点消息两种事件类型,并把所有的数据存放到事件消息中枢。其中节点活动类型是指在 ROS 系统中所有节点的活动数据,包括节点的启动、发布主题、订阅主题等事件。节点消息类型是指系统中各个节点之间传递的主题消息。

2.2 事件消息中枢

ROS 程序主要由 C++编写,而 JavaMOP 是监控 Java 程序的一种运行时验证技术。为了能够更好地利用 JavaMOP 技术,本系统设计了一个用于信息传输的中间件——事件消息中枢,它包括 C++服务端和 Java 客户端两个部分。客户端和服务端采用请求和应答的方式进行通信,服务端将待监控的信息存放到一个缓冲消息队列,客户端主动发起请求获取服务端队列中的数据。为了方便数据传输和解析,我们使用 json 字符串的方法将数据转换成字节流。json 是一种轻量级的数据交换格式,方法简单灵活,又具有表达多层次复杂内容的能力。

针对节点消息类型,主题消息的内容是一个结构体数据。例如控制小车运动的"/cmd_vel"消息对应的结构数据是"geometry_msgs/Twist",该结构体"Twist.msg"包含 3 维浮点数:"Vector3 linear(x,y,z)"和"Vector3 angular(x,y,z)"。其中线速度表示为"msg.linear.x",角速度表示为"msg.angular.z"。为了方便使用,将其转化成 json 字符串,如一个具体的消息数据为"[1,0,0][0,0,2]"则转化后的 json 字符串为:"message":{"cmd_vel":{"x":1,"y":0,"z":0},"angular":{"x":0,"y":0,"z":2}}}

针对节点活动类型,包括节点注册与注销、主题发布与订阅、消息发送与接收,具体字段和含义如表 1 所示。该类型数据同样通过 json 字符串的方法进行传输,如下所示:

节点注册:

{"node_event":{"node":"car","type":"register"}}

发布主题:

{"node_event":{"node":"car","type":"publish","topic":"cmd_vel"}}

订阅主题:

{"node_event":{"node:car","type":"subscribe","topic":"cmd_vel"}}

表 1 节点活动类型

字段	含义
Register	节点注册
Unregister	节点注销
Publish	发布主题
Subscribe	订阅主题
Send	发送消息
Receive	接收消息

2.3 自定义场景描述

一个正在运行的 ROS 系统会产生许多节点活动和节点消息数据,但是并非所有数据都需要监控,仅需要关注与系统安全相关的部分。ROS 中的待监控数据本身具有结构化,我们通过一种形式化描述的方法更加方便地表达要监控的数据。因此,本系统设计了一个代码生成工具,可以根据自定义场景描述文件自动生成特定的监控器,包括 Java 代码和 Mop 代码。

表 2 是自定义场景描述字段和格式内容说明。"Source"字段指定监控内容的来源,它包括节点活动或节点消息类型的数据,格式为"主题:主题名,或者节点:节点名"。"Var"表示要监视消息中的变量,格

式为“变量内容:消息结构体值”。“Event”字段是节点活动事件或节点消息变量监控事件,格式为触发函数的事件条件语句,它将与“Property”字段中的逻辑语句组合在一起形成属性逻辑语义。“Property”字段是一个属性表达式,格式为“逻辑类型:逻辑语义”。“Violation”为违背字段,内容为违背时的错误执行代码段,如果该属性被违反,将根据“Violation”字段后面的代码内容进行打印警告。

表 2　自定义场景描述说明

字段	含义	格式内容
Source	来源	主题:主题名,或者节点:节点名
Var	变量	变量内容:消息结构体数值
Event	事件	触发函数的事件条件语句
Property	属性	属性表达式,逻辑类型:逻辑语义
Violation	违背	违背时的错误执行代码段

2.4　监控代码生成机制

监控器的代码生成机制如图 4 所示。代码生成器能依次解析自定义场景描述的“Source”、“Var”、“Event”和“Property”字段内容,计算出这几个字段的内部依赖关系,生成若干个代码段。并将生成的代码填充到预先准备好的代码模板中,最终生成完整的 Java 代码和 Mop 代码文件。其中生成的 Java 代码包含一个 TCP 客户端,该客户端通过连接事件消息中枢获取待监控的数据。我们使用 shell 脚本在 Ubuntu 系统上编译代码,通过执行 javac 等指令把 Java 代码编译成可执行文件,通过 javamop 等指令把 Mop 文件编织成 JavaMOP 代理。

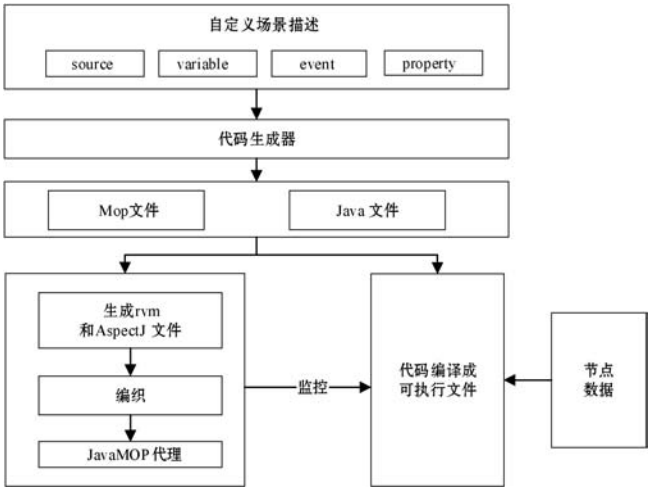

图 4　自动代码生成的主要流程

下面展示了一个生成的简单的“monitor. mop”文件,其中“RESET”是一个特殊表达式,可以将监控器重置为初始状态。

```
monitor( )
  event exceed_speed after( ):
        call( * * . exceed_speed( ) )
  ltl :[ ]! (exceed_speed)
  @ violation {System. out. println("warning from mop"); RESET;}
```

2.5　界面设计

我们在 Ubuntu14. 04 上成功地实现了 ROS-Monitor 框架。为了让系统易操作和使用,提供了一个用户界面,如图 5 所示,用户可以设置“Source”、“Var”、“Event”和“Property”字段内容。当用户点击“Generator”按钮时,工具将根据自定义场景描述,自动生成 Java 代码和 Mop 代码并编译到 JavaMOP 代理中。当用户单击“Reset”按钮时,输入内容将被清除,如果监控器正在运行,它将被强制停止监控。当用户点击“Monitor”按钮后,系统会结合场景描述内容,监控正在运行 ROS 的系统。如果系统行为违背安全属性,警告日志将被打印在“Monitor Result”文本框内。

图 5　用户界面

3　安全策略和实验

本节通过一个简单示例和一个综合实验来介绍 ROS-Monitor 框架的使用。用户可根据具体的需求,通过自定义场景描述生成特定的监控器,即可方便地监控系统行为。

3.1　简单示例

攻击者可以通过在 TCP/IP 层构造虚假请求包来伪造 ROS 节点。例如在节点注册之前进行发布或订阅主题,该操作不符合主题发布的正常运行顺序,会带来潜在威胁。因此必须 ROS 系统中节点在发布主题

之前,确保该节点已经注册。假设系统发布节点为“talker”,则自定义场景描述为:

```
Source:
    node: talker
Var: None
Event:
    node_start: talker. register
    node_publish: talker. publish
Property:
    ltl:[ ]node_publish => ( * ) node_start
violation:
    System. out. println("Suspicious Nodes");
```

此示例对“talker”节点的注册和发布进行安全监控。由于操作不涉及变量,“Var”字段为空。当触发“talker”节点注册活动时,激活“node_start”事件;触发“talker”节点的发布主题时,激活“node_publish”事件。为了保证系统始终处于安全状态,则必须保证在节点发布主题之前先注册该节点,即在激活“node_pulish”事件之前先激活“node_start”事件,LTL 语句为“[ ] node_publish => (*) node_start”。若系统行为违背该安全属性,则打印出“Suspicious Nodes”警告信息。

图 6 显示了节点发布的有限状态机的转化过程。其中“ST”指的是系统处于“start”状态。“S”指的是系统处于“safety”状态,“US”指的是系统处于“unsafety”状态。当“node_start”事件被激活时,系统从“ST”启动状态迁移到“S”安全状态。若之前未先激活“node_start”事件,而直接进行“node_publish”事件操作,系统将迁移到“US”状态并发出警告提示。

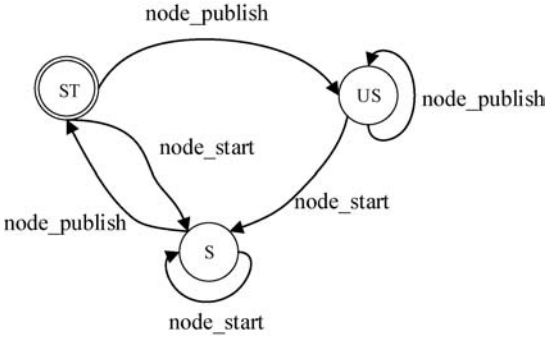

图 6 节点发布的有限状态机

3.2 综合实验

本节我们以一个综合的案例介绍 ROS-Monitor 框架。无人驾驶汽车又称为轮式移动机器人,它通过车载传感器感知周围环境,获取道路和障碍物信息来控制车辆的转向和速度,从而使车辆能够安全、可靠地在道路上行驶。

Gazebo[20]是 ROS 中的物理仿真环境,它是一款机器人的仿真软件,基于 ODE 的物理引擎,可模拟机器人以及环境中很多物理特性。我们采用 Gazebo 建立无人驾驶小车模型,车身搭载激光雷达[21]。激光雷达数据量大、检测距离远、精度高,可以获取很多的三维坐标信息。通过这些信息,我们可以检测障碍物的方位,判断它的大小、高度等。本实验基于 Gazebo 环境构建了一个虚拟的驾驶道路场景,模拟小车通过十字路口的过程。期间进行多次实验,让小车分别以直行和转弯的方式通过该路口。我们设计了小车在行驶过程中,必须保证的 4 个重要的安全属性:

1) 节点必须遵循主题的发布订阅规则,主题被订阅之前,该主题已经被发布了。即在“car”节点订阅“cmd_vel”主题之前,“car_control”节点已经发布了该主题。

2) 当“car_control”节点发送消息时,该节点必须已发布主题,并且该节点处于未关闭状态。

3) 当汽车行驶到十字路口,不能超过十字路口限定速度 16 km/h,否则发出超速警告。

4) 不允许“car”节点高速急转弯通过十字路口。角度和速度分别不能同时超过 50°和 13 km/h。

根据上述安全属性描述,我们设定自定义场景描述如下:

```
Source:
    topic: cmd_vel
    node: car_control, car
Var:
    speed: cmd_vel. linear. x
    angle:cmd_vel. angular. z
Event:
    crtl_publish: car_control. publish
    car_subsribe: car. subsribe
    crtl_close: car-control. unregister
    crtl_send: car_control. send
    turn_speed: speed > 13
    turn_angle: angle > 50
    exceed_speed: speed > 16
Property:
  ltl:[ ]car_subsribe => ( * ) crtl_publish
    @ violation: System. out. println("violate publish subscribe");
  ltl:[ ]crtl_send => ( not crtl_close S crtl_publish)
    @ violation:System. out. println("Suspicious Nodes");
  ltl:[ ]! exceed_speed
    @ violation:System. out. println("exceed speed");
  ltl:[ ]! (turn_speed and turn_angle)
    @ violation: System. out. println("Sharp turn at high speed");
```

本实验运行了 2 个小时,监控结果参见表 3。此实验共出现 22 条警告。其中,V1 属性违反次数为 0,所有的节点都遵循发布/订阅规则进行通信。对于 V2 属性,正常情况下较难发生,我们模拟了 5 次不合理的

节点启动顺序。关闭节点后,该节点发布主题消息,监控器打印了 5 次违规警告。由于小车无法识别特殊路段的限速标志,导致通过十字路口时,超过了我们设定的最大速度限制。因此汽车在 V3 属性上显示 13 条超速警告。对于 V4 属性,有 4 个急转弯警告是由于遇到突然出现的其他汽车时没有及时进行降速躲避。

表 3　实验结果

编号	属性	违反次数
V1	[ ]car_subscribe => (*) crtl_publish	0
V2	[ ]crtl_send => (not crtl_close S crtl_publish)	5
V3	[ ]! exceed_speed	13
V4	[ ](turn_speed and turn_angle)	4

通过实验分析,该框架可以全面监视汽车每个节点的运行状态,在系统没有违背安全属性的情况下没有发出告警信息,在系统违背安全属性的情况能够准确打印告警,并且告警次数符合预期。以上实验验证了 ROS-Monitor 框架的可用性,保证了系统的正常安全通信。

4　结　语

针对 ROS 在节点活动和消息通信方面存在的安全问题,本文提出 ROS-Monitor 的运行时验证框架。该工具可以将自定义描述场景生成特定的监控器,包括自动生成的 Java 代码和 Mop 代码,并支持 LTL 等时序逻辑对 ROS 进行验证。在系统行为违反安全属性时,打印相关警告信息。该框架适用于主题方式通信的 ROS 模型(支持 Indigo 版本),该工具从场景描述到监控结果的产生,实现了高度的简单化和自动化,并通过实际的案例验证了该框架的实用性。ROS 消息通信方式有主题和服务两种类型,本框架仅对主题通信进行安全验证,后续将在服务通信方面进行相关研究。

参　考　文　献

[1] 张建伟,张立伟,胡颖,等. 开源机器人操作系统——ROS[M]. 北京:科学出版社,2012.

[2] 张硕,贺飞. 运行时验证技术的研究进展[J]. 计算机科学,2014,41(S2):359 - 363.

[3] Belta C, Yordanov B, Gol E A. Formal methods for discrete-time dynamical systems[M]. Springer, 2017.

[4] Rashid A, Hasan O. Formal analysis of linear control systems using theorem proving[C]//ICFEM 2017: Formal Methods and Software Engineering, 2017:345 - 361.

[5] Huang J, Erdogan C, Zhang Y, et al. ROSRV: Runtime verification for robots[C]//Runtime Verification,2014:247 - 254.

[6] Balsa-Comerón J, Guerrero-Higueras á M, Rodríguez-Lera F J, et al. Cybersecurity in autonomous systems: Hardening ROS using encrypted communications and semantic rules[C]//ROBOT 2017: Third Iberian Robotics Conference, 2017:67 - 78.

[7] White R, Christensen H I, Quigley M. SROS: Securing ROS over the wire, in the graph, and through the kernel[EB]. arXiv:1611.07060, 2016.

[8] Breiling B, Dieber B, Schartner P. Secure communication for the robot operating system[C]//11th Annual IEEE International Systems Conference. IEEE, 2017.

[9] Hussein S, Meredith P, Roşlu G. Security-policy monitoring and enforcement with JavaMOP[C]//Workshop on Programming Languages & Analysis for Security, 2012.

[10] 张琪琦,吴叶兰,王建涛,等. 基于 ROS 的智能家居服务机器人设计[J]. 信息技术与信息化,2019(5):91 - 94.

[11] 王亚,王瑞,关永,等. RGMP-ROS 混合机器人操作系统节点间通信的形式化验证[J]. 小型微型计算机系统,2015,36(10):2379 - 2383.

[12] Meredith P, Roşu G. Runtime verification with the RV system[C]//Runtime Verification,2010:136 - 152.

[13] Leucker M, Schallhart C. A brief account of runtime verification[J]. Journal of Logic & Algebraic Programming, 2008,78(5):293 - 303.

[14] Jin D, Meredith P O, Lee C, et al. JavaMOP: Efficient parametric runtime monitoring framework[C]//2012 34th International Conference on Software Engineering(ICSE), 2012: 1427 - 1430.

[15] Chen F, D'Amorim M, Roşu G. Checking and correcting behaviors of java programs at runtime with Java-MOP[J]. Electronic Notes in Theoretical Computer Science,2006,144(4): 3 - 20.

[16] Meredith P O, Roşu G. An overview of the MOP runtime verification framework[J]. International Journal on Software Tools for Technology Transfer, 2012, 14(3):249 - 289.

[17] Gastin P, Oddoux D. Fast LTL to Buchi automata translation[C]//Proceedings of the 13th International Conference on Computer Aided Verification, 2002.

[18] Bauer A, Leucker M, Schallhart C. Runtime verification for LTL and TLTL[J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(4):1 - 68

[19] 肖美华,薛锦云. 时态逻辑形式化描述并发系统性质[J]. 海军工程大学学报,2004,16(5):10 - 13.

[20] Okoli F, Lang Y, Kermorgant O, et al. Cable-driven parallel robot simulation using Gazebo and ROS[M]//ROMANSY 22-Robot Design, Dynamics and Control. Springer,2019: 288 - 295.

[21] 韩昊旻,张崇明,陈志红,等. 基于 ROS 和激光雷达的 AGV 导航系统设计与实现[J]. 电子测量技术,2018,41(8):112 - 117.