Introduction of Robot Operating System 2: ROS2

-- H2020-MSCA-ITN SAS Program - ESR2 - LIAO Yuan

In autonomous driving, the increasing complexity of systems poses great changes in maintaining dependability for lifecycle of software engineering. Among them, middleware [1] is one of the most challengeable research parts for overall systems. The middleware challenges are involved in: capabilities of runtime adaptation to applications, communication between heterogeneous systems in different levels of middleware [1], runtime safety assurance under failure circumstances, and others. In this article, I introduce a new Robot Operating System (ROS2) platform, aiming to provide an overview about its architecture, advantages, and features.

![Figure 1. ROS robots overview [2]](image)

Why ROS2

Because of the lack of support for real-time performance in ROS1, the creation of ROS2 was proposed to address the performance limitations.

ROS1, the open source software, is one of the most popular prototyping platform for developing robots (Figure 1). The initial design goal of ROS1 was to provide a development platform for the Willow Garage PR2 robot to do research [3]. Gradually, with the continuous software iteration by community around the world, ROS1 expanded its usage space to many different fields such as industrial arms, self-driving cars, aerial vehicles, and more. However, the architectural design limitations (i.e. single point of failure) and performance bottlenecks (i.e. real-time capabilities) restrict ROS-driven systems to be applicable in ever-changing autonomous fields. Specifically, ROS1 has the following limitations [3] [4]:

- **Multi-robot system**: Multi-robot system is a key issue in the field of robotics. It can solve the problems of insufficient performance and unavailability of a single robot. However, there is no standard method for building a multi-robot system in ROS1.
- **Multi-platform**: ROS1 is based on Linux and cannot be applied or has limited functions on Windows, MacOS, RTOS and other systems. This places higher requirements on robot developers and development tools, and also has great limitations.
- **Real-time**: Robots in many application scenarios have high requirements for real-time performance, especially in the industrial field. The system needs to achieve hard real-time performance indicators, but ROS1 lacks real-time design, so it is stretched in many applications.
• **Network connection:** The distributed mechanism of ROS1 requires a good network environment to ensure data integrity, and the network does not have data encryption, security protection and other functions. Any host in the network can obtain the message data issued or received by the node.

• **Production:** The stability of ROS1 is not good. The important links such as ROS1 Master and nodes will be inexplicably down in many cases, which makes many robots transition from research to market very difficult.

Although many developers and research institutions have tried to improve the above features of ROS1 problem, it is often hard to improve the overall performance. This motivated the development of ROS2, a complete refactoring of ROS that puts the successful concept onto a modernized and improved foundation [5].

![Image](image1.jpg)

Figure 2. ROS1/ROS2-based industrial applications [6],[7],[8],[9]

Compared with ROS1, the design goals of ROS2 are listed below [3]:

• **Supports multiple robot systems:** ROS2 adds support for multi-robot systems and improves the network performance of communication between multi-robots. More multi-robot systems and applications will appear in the ROS community.

• **Bridging the gap between prototypes and products:** ROS2 is not only aimed at the scientific research field, but also concerned about the transition from research to application of robots, which can allow more robots to directly carry ROS2 systems to the market.

• **Support microcontroller:** ROS2 can not only run on existing X86 and ARM systems, but also support embedded microcontrollers like MCUs (ARM-M4, M7 cores).

• **Support real-time control:** ROS2 also adds support for real-time control, which can improve the timeliness of control and the performance of the overall robot.

Multi-platform support: ROS2 not only runs on Linux systems, but also adds support for Windows, MacOS, RTOS and other systems, giving developers more choices.

**ROS/ROS2 Architecture**

ROS2 absorbs development experience based on ROS1 platform and makes summarization to provide abundant off-the-shelf library resources to support popular applications like navigation. Developers can maintain less non-robotics-specific code, but focus more on taking advantage of features in ROS1.
libraries to improve software. From an architecture perspective, ROS2 is designed into multiple layers, which is visualized in Figure 3. Compared with ROS1, the main differences are shown as below:

- ROS1 mainly supports Linux-based operating system. ROS2 provides more portability of deployment on underlying operating systems, such as Linux, Windows, Mac, and RTOS.
- ROS data transport protocol uses TCPROS/UDPROS, and communication is highly dependent on the operation of Master node. Communication in ROS2 is based on DDS (Data Distribution Service) [12] standard, enhancing fault tolerance capabilities.
- Intra-process in ROS2 provides more optimized transmission mechanism.

![Figure 3. ROS/ROS2 architecture overview [10]](image)

DDS and ROS2 API layout is shown in Figure 4. In user land section, it officially supports both C++ and Python programming languages. Correspondingly, it requires underlying layer (rclcpp or rclpy) to provide language-specific API to support applications. Below it is C-based ROS1 client library (rcl), which is used to ensure core algorithms in all language-specific client libraries. ROS1 middleware interface (rmw) aims to abstract DDS implementations and streamline Quality-of-Service (QoS) configuration.

![Figure 4. DDS and ROS2 API layout [11]](image)

For inter-node communication, ROS2 uses DDS, an industrial communication standard from OMG, to Publish-Subscribe transport. The benefit of using DDS is that there are concrete specifications for third parties to review, audit, and implement with varying degrees of interoperability [11]. QoS (Quality-of-Service) of DDS gives flexible parameters settings to control the reliability of communication. What’s more, ROS2 can work with different vendors of DDS like FastRTPS, RTI-Connext, OpenSplice, and more.
Portability among DDS vendors provides users alternatives to select the specified requirements, as well as against changes in DDS vendors.

Communication model of ROS2

The communication model of ROS1 mainly includes communication mechanisms such as topics and services. The communication model of ROS2 will be slightly complicated, and many DDS communication mechanisms are added. The DDS-based ROS2 communication model contains the following key concepts [10] (Figure 5):

- **Participant**: In DDS, each publisher or subscriber is called a participant. Corresponding to a user using DDS, a defined data type can be used to read and write the global data space.
- **Publisher**: The performer of data publishing supports the publishing of multiple data types, and can be associated with multiple data writers (DataWriter) to publish messages of one or more topics.
- **Subscriber**: The data subscription executor supports subscriptions of multiple data types. It can be associated with multiple data readers and subscribe to messages of one or more topics.
- **DataWriter**: The upper-layer application updates the object of data to the publisher. Each data writer corresponds to a specific topic, similar to a message publisher in ROS1.
- **DataReader**: The upper-layer application reads data from subscribers. Each data reader corresponds to a specific topic, similar to a message subscriber in ROS1.
- **Topic**: Similar to the concept in ROS 1, topics need to define a name and a data structure, but each topic in ROS2 is an instance that can store historical message data in that topic.
- **Quality of Service**: Abbreviated as QoS Policy, this is a new and very important concept added in ROS2. It controls all aspects of the communication mechanism with the underlying layer, mainly from the aspects of time limit, reliability, continuity, and history to meet user Data requirements for different scenarios.

![Figure 5. DDS communication model [10]](image)
Conclusion

Notwithstanding ROS2 is still in the early stage of development, but it provides a promising blueprint to the autonomous domain. More industrial-driven requirements in safety mechanisms, real-time scheduling performance, and interoperability with ROS are being realized. It will motivate thousands of developers and researchers from both industry and academia to improve the development of ROS2 for the autonomous field. In the near future, more ROS2-driven autonomous software product will be released into public.

References