

ROS2 for Robot Mapping and Navigation

Zoe Li
Zoe.Li@bshg.com

Abstract—This guide gives an overview of ROS2 status with an example of SLAM implementation. ROS2 is an upgrade after one decade since the introduction of ROS (Robotic Operating System) and is still under heavy development at this moment as June 2019. In this guide, we discuss and evaluate some of its new features by implementing SLAM in ROS2 in simulation and on a real robot. These new features are briefly introduced and some are tested in this implementation. A demo with source code is provided in the end of the paper.

I. INTRODUCTION

A. Robot Operating System Background

Robot Operating System (ROS) is a robotics middleware that was created by Willow Garage and Stanford University and now maintained by Open Source Robotics Foundation(OSRF)[2]. As an open source framework for various robotics software development, ROS provides functions such as hardware abstraction, device control, message passing, package management and libraries for different functionalities. The modulated packages of ROS allows users to focus on application development rather than spending much effort to reinvent the wheel.

B. ROS2 Design Background

ROS was originally designed for PR2 use case. PR2 robot works as a standalone robot with excellent network connectivity, also PR2 applications are mostly research based, therefore the early design concept of ROS does not need to consider real-time problems.

Nowadays ROS has gained tremendous popularity in robotics community, and the use cases has grown beyond the scope of academia and scientific research. Many robotics applications such as industrial robots, outdoor robots(for example driver-less cars), unmanned aerial vehicles(UVA) have become more and more complicated, as a result, those applications have higher demand on the robust real-time performance of the robot operating system. Although ROS1 is still a very popular development tool in the field of robotics, the limitations of the original design have become a driving force of the new ROS2 design[3].

With the growing demand of cross operating system platform and real-time functionality from the ROS community, ROS2 development was first announced at ROSCon 2014, and the first alpha version was launched in August 2015. On December 8, 2017, the highly anticipated ROS 2.0 finally released its first official version, Ardent Apalone. As of 2019, the newest version ROS 2 Dashing Diademata was released on May 30.

Compare to ROS1, ROS2 has the following support for robotics applications[3]:

- **Cross-system platform support:** ROS2 support for Linux, Windows and macOS as well as the real-time operating system(RTOS).
- **Multi-robot system support:** Improved communication system allows robust network performance for multi-robot system
- **Real-time control:** support to improve the timeliness of a robot control application and overall robot performance
- **Non-ideal networks:** Support robot applications such as UAV, underground robot or underwater robot that has bad networks connection compare to indoor robots
- **Production environments:** ROS 1 has target users as research labs, while ROS 2 is suitable for real-world applications.
- **Small embedded platforms:**ROS 2 does not rely on a master to distribute messages, and its native UDP communication system performs much better on distributed embedded system[4].

C. ROS2 Communication

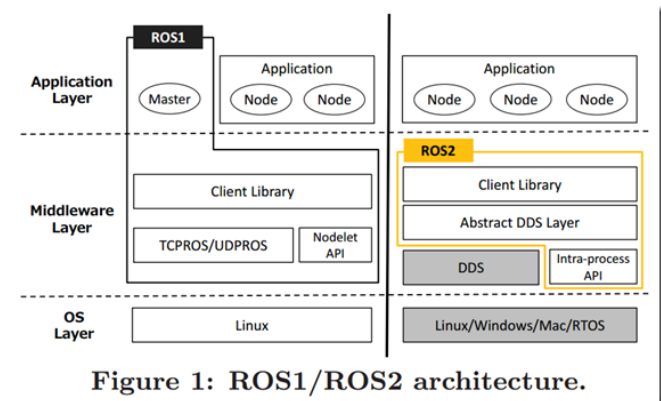


Figure 1: ROS1/ROS2 architecture.

Fig. 1: ROS1/ROS2 Architecture [1]

ROS1 uses TCP (Transmission Control Protocol) as its communication protocol. TCP is a connection oriented network, this means that TCP tracks all data sent, requiring acknowledgment for each octet (generally), therefore, ROS1 has a centralized network configuration which requires a running ROS master to take care of naming and registration services. With the help of the master, ROS nodes could find each other on the network and communicate in a peer-to-peer fashion. In ROS1 setting, all nodes will depend on

the central ROS master. When the network becomes lossy and unstable (especially if nodes are distributed on several computers), the communication will not be reliable for real-time applications or multi-robot systems.

ROS2 uses Data Distribution Service (DDS) as the communication middleware. UDP is a Data-Centric-Publish-Subscribe (DCPS) model, and this model will create global data space for individual applications to exchange information. DDS will identify a data object by its topic name and then subscribe to this topic, therefore, DDS does not have a central distributor for all information. The DDS publish-subscribe model avoids complex network programming for distributed applications. ROS2 provides an abstraction layer of DDS, so users do not need to pay attention to the underlying DDS structure. The ROS2 Middleware Interface (RMW) allows users to choose different Quality of Service (QoS). The real-time publish-subscribe (RTPS) protocol allows ROS2 nodes to automatically find each other on the network, thus there is no need for a ROS2 master. This is an important point in terms of fault tolerance.

D. ROS 2 QoS policy

II. RELATED WORK

When ROS 2 Bouncy was released, a TurtleBot 2 demo was provided to demonstrate some popular mapping and localization packages that run in ROS 2. TurtleBot 2 demo uses Google Cartographer to get maps of the environment, and use AMCL (Adaptive Monte Carlo Localization) package to localize. TurtleBot 2 demo also provided TurtleBot 2 driver and the Orbbec Astra depth camera sensor driver. At the time this demo was created, ROS 2 navigation stack was still under development, therefore, TurtleBot 2 demo uses joystick to manually operate the robot to create maps.

After ROS 2 Dashing was released, The manufacturer of TurtleBot 3 ROBOTIS[6] provided a TurtleBot 3 demo that runs ROS 2 dashing. TurtleBot 3 demo includes robot mapping and navigation, so this demo for Kobuki robot is based on the work of TurtleBot 3.

III. METHOD

The objective of this demo is to build a Kobuki SLAM and navigation demo on top of the existing TurtleBot 2 demo, and update packages so that the Kobuki robot can achieve SLAM and autonomous navigation using the latest ROS 2 Dashing Diademata release.

A. Hardware and software setup

- Hardware setup
 - Robot: Kobuki (turtlebot2)
 - Sensor: Hokuyo laser scanner
- ROS2 Packages:
 - Mapping
 - * cartographer (binary release available)
 - * cartographer-ros
 - Navigation
 - * Navigation2



Fig. 2: Image of Kobuki Robot (TODO)

- Visualization
 - * rosl_bridge (binary release available)
 - * rviz
- Controller and drivers:
 - * teleop_twist_keyboard (binary release available)
 - * urg_node: URG laser scan driver
 - * turtlebot2_drivers: provide kobuki_node to drive the Kobuki Robot
- Simulation:
 - * Gazebo9 simulation (binary release available)
- Simulation
 - Gazebo simulation environment
 - Kobuki model
 - Gazebo differential drive plug-in
 - Gazebo sensor plug-in

B. Implementation

1. Setup Build files

ROS2 uses a build system called Ament Build tool. Ament is an evolution of catkin that automatically builds a set of independent packages based on their dependency topological order. Ament will also install and configure the environment for packages, so the user does not need to follow a specific dependency order to build packages. The package.xml will provide meta information about the packages and their dependencies, and the build configuration are stored in the CMakeList.txt

2. Build packages using colcon build

ROS1 has multiple different building tools such as catkin_make, catkin_make_isolated and catkin_tools. ROS 2 provides a universal build tool called colcon. This colcon tool makes the building testing and using multiple packages easy[7].

3. Manual mapping using Cartographer

TODO Kobuki_driver SetUp

This demo uses teleop-keyboard to manually operate the Kobuki robot. The cartographer node utilizes the laser scan

data obtained from the hokuyo laser scanner and the odometry information published through the kobuki_node robot driver and create a 2D map. This demo uses teleo-keyboard to operate the Kobuki robot, and obtain a map using google cartographer. The rqt_graph of the structure is show in figure 3

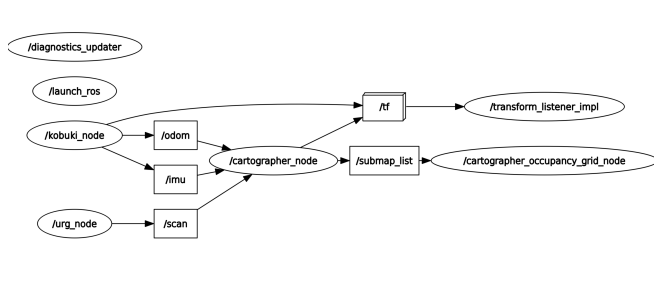


Fig. 3: ROS2 rqt_graph

4. Data visualization using ros1_bridge

ROS 2 is still under heavy development, so many packages built for ROS 1 have not yet been converted into ROS 2 compatible version. ROS 2 designers provided a solution to bridge the communication between ROS 1 and ROS 2 by ros1_bridge.

This demo uses ros1_bridge to pass data into ROS 1 to visualize the map obtained by cartographer. ros1_bridge provides a network bridge which enables the exchange of messages between ROS 1 and ROS 2. This bridge will establish connection when matching publisher-subscriber pairs are active to a topic on either side of the bridge[5]. The current version supports most of the common message types in ROS 1 and ROS 2, user can also create their own customized message pairs through a package mapping rule file that matches all messages with the same names and fields within a pair of packages.

5. Launch File Setup

ROS 2 launch system are very different compare to ROS 1, it uses python script to launch applications and describe configurations. Python launch file has very little prescribed structure, which gives uses lots of flexibility. For example, this demo uses python OS (operating system interfaces) to directly set the gazebo model path environment variable, so users do no have to manually enter this environment variable into the terminal shell.

```
os.environ['GAZEBO_MODEL_PATH'] =
gazebo_model_path
```

Also ROS 2 has a new system of setting parameters in launch files. ROS 2 stores parameters of a node in a .yaml file and pass the .yaml file as a launch argument. The following code shows a example of launching urg_node in this demo:

```
launch_ros.actions.Node(
    package="urg_node",
    node_executable="urg_node",
    output="screen",
```

```
arguments=["__params:=/PATH/urg.yaml"])
```

The following is an example of urg_node parameter file in ROS 2 setting:

```
urg_node:
  ros__parameters:
    angle_max: 3.14
    angle_min: -3.14
    ip_port: 10940
    serial_port: "/dev/ttyACM0"
    serial_baud: 115200
    laser_frame_id: "laser"
    calibrate_time: false
    default_user_latency: 0.0
    diagnostics_tolerance: 0.05
    diagnostics_window_time: 5.0
    error_limit: 4
    get_detailed_status: false
    publish_intensity: false
    publish_multiecho: false
    cluster: 1
    skip: 0
```

6. Simulation Setup

In this demo, simulation environment was modeled as the Bosch Sunnyvale office. The simulated office settings shown in figure4 has walls and desks as obstacles for the robot to perform mapping and navigation.

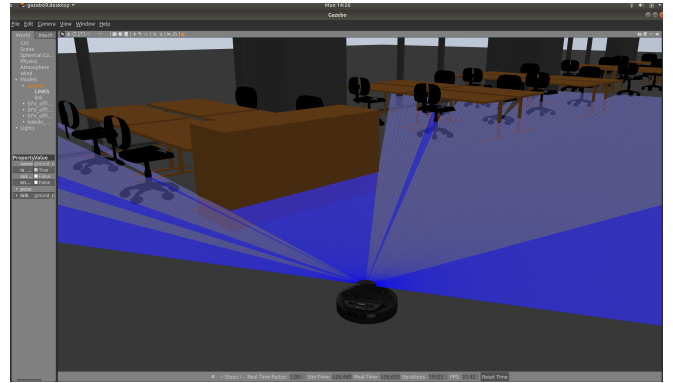


Fig. 4: Kobuki and laser scanner in Gazebo 9 Simulation

7. Navigation in simulation

The navigation 2 package is designed for ROS 2, just like the original navigation stack in ROS 1, navigation 2 enables a robot to autonomously reach a goal state. Kobuki navigation uses the navigation2 packages to navigate through the map obtained by google cartographer. Given a robot initial pose, and a goal pose shown in rviz map, navigation 2 will generate a path plan and send command to the drive the robot. The robot will reach the goal while avoiding obstacles along the path.

IV. RESULTS

A. Robot Mapping

The map shown in figure 5 was obtained by driving the robot in a room. The result draws reasonable boundaries.

However area that near large windows shows a large amount of noise due to laser reflection.

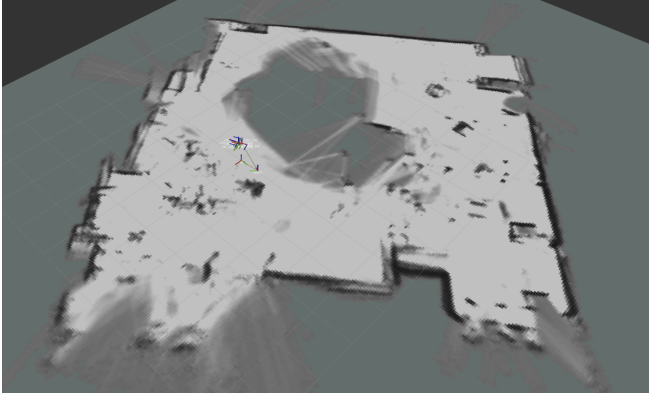
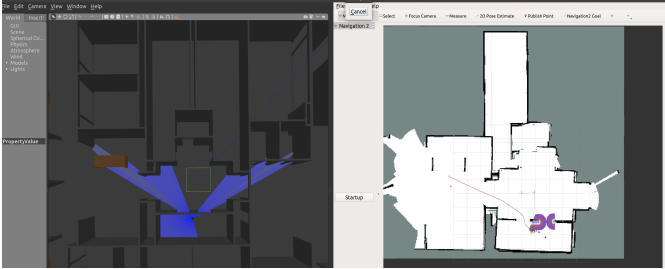


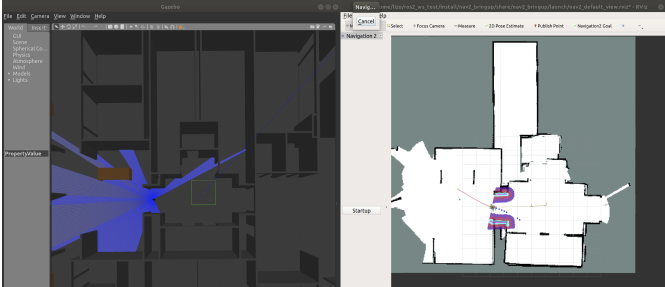
Fig. 5: map obtained by google cartographer

B. Gazebo 9 Simulation

The AMCL localization and navigation works well in simulation. However, due to lack of landmarks, the localization result is highly depend on how the use select the initial position.



(a) Starting point of navigation



(b) Midpoint of the navigation

V. CONCLUSIONS

ROS 2 is a amazing tool for Robotics application development. However, ROS 2 is still under heavy development, packages are constantly being updated, therefore it might still be too early to fully adapt ROS 2.

ROS2 has many new features that are very convenient, such as the new launch system, this allows user to set environment variables easily through launch file rather than manually type into the comment line.

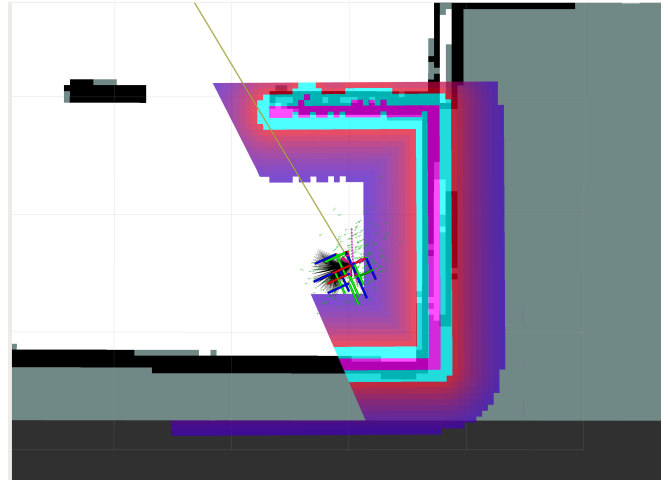


Fig. 7: Detailed navigation result

APPENDIX ACKNOWLEDGMENT

REFERENCES

- [1] Y. Maruyama, S. Kato, and T. Azumi, Exploring the performance of ROS2, Proceedings of the 13th International Conference on Embedded Software - EMSOFT 16, 2016.
- [2] Open Source Robotics Foundation(OSRF) <http://www.osrfoundation.org/>.
- [3] Gerkey, B. (2019). Why ROS 2?. [online] Design.ros2.org. Available at: https://design.ros2.org/articles/why_ros2.html [Accessed 9 Jul. 2019].
- [4] Ros.org. (2019). Morgan Quigley (OSRF): ROS 2 on "Small" Embedded Systems - ROS robotics news. [online] Available at: <http://www.ros.org/news/2016/03/morgan-quigley-osrf-ros-2-on-small-embedded-systems.html> [Accessed 15 Jul. 2019].
- [5] GitHub. (2019). ros2/ros1_bridge. [online] Available at: https://github.com/ros2/ros1_bridge [Accessed 9 Jul. 2019].
- [6] ROBOTIS. (2019). ROBOTIS. [online] Available at: <http://en.robotis.com/> [Accessed 15 Jul. 2019].
- [7] "Using colcon to build packages", Index.ros.org, 2019. [Online]. Available: <https://index.ros.org/doc/ros2/Tutorials/Colcon-Tutorial/>. [Accessed: 16- Jul- 2019].