Robot Operating System[ROS]

INTRODUCTION

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. ROS has various open source tools which help build many projects. ROS was started in 2007. There are 2 main projects of ROS : ROS 1 and ROS 2. The location on our system where we are developing with ROS 2 is called as a workspace.ROS 2 allows the combining of workspaces using the shell environment. ROS 2 also allows the installation of several distributions like dashing and eloquent on the same computer and switching between them.

CLI TOOLS

The ROS command line interface(CLI) is a set of programs for starting, inspecting, controlling, and monitoring a ROS robot.

There are many tools which help us understand ROS 2 in a more efficient and simpler way such as :

Turtlesim is a lightweight simulator for learning ROS 2. It illustrates what ROS 2 does at the most basic level to give an idea of what one will do with a real robot.

The ROS 2 tool is used to start a node, set a parameter, listen to a topic , etc. It basically gives an idea about how a user manages and interacts with the ROS system.

rqt is a GUI tool for ROS 2 which provides a user-friendly way to manipulate ROS 2 elements. We use rqt to call the /spawn service which is used to create another turtle in the turtlesim window.

The /set\_pen service allows the customization of each line assosciated with each turtle.

turtle\_teleop\_key is used to switch between the 2 different turtles and customize the lines for them individually.

# ROS 2 GRAPH

The ROS graph is a network of ROS 2 elements processing data together at the same time. It is a way of visualizing and understanding the relationships and interactions between different ROS nodes and their communications.

# NODES

Each node in ROS is responsible for a single modular process for eg : controlling the wheel motors. Each node can send and receive data from other nodes via topics, services, actions, or parameters. In ROS 2 a single executable program can contain multiple nodes.

Commands:

ros2 node list : It will show the names of all running nodes.

Remapping : It allows to reassign default node properties, like node name, topic names, service names, etc to custom values.

ros2 node info : It returns a list of subscribers, publishers, services, and actions. i.e. the ROS graph connections that interact with that node.

# TOPICS

Topics act as a medium for nodes to exchange messages. Topics can have various types of communication like : point to point, one-to-many, many-to-one or many-to-many. A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.

The rqt\_graph helps visualize the changing nodes and topics, as well as the connections between them.

Nodes send data over topics using messages. Publishers and subscribers must send and receive the same type of message to communicate.

Commands:

ros2 interface show : It helps to learn the details and also the structure of data that the message expects.

ros2 topic echo : It helps to see the data being published on a topic.

ros2 topic pub : To publish data to a topic directly.

ros2 topic hz : To view the rate at which data is published.

To summarize, topics mainly have one way communication pattern based on publisher-subscriber model where node publishes information that can be consumed by one or more subscribers. Topics allow nodes to subscribe to data streams and get continual updates.

# SERVICES

Services are based on a call-and-response model. Services only provide data when they are specifically called by a client. There can be many service clients using the same service but only one service server for a service. Nearly every node in ROS 2 has these infrastructure services that parameters are built of.

There are some turtlesim-specific services like : /clear, /kill, /reset, /spawn, /turtle1/set\_pen, /turtle1/teleport\_absolute, and /turtle1/teleport\_relative

Services have types that describe how the request and response data of a service is structured***.***

service types have two parts: one message for the request and another for the response.

ros2 service type is the feature that helps us to find out the type of service.

To find all the services of a specific type we use ros2 service find.

To summarize, a service is a request/response pattern where a client makes a request to a node providing the service and the service processes the request and generates a response.

# Parameters

Parameters are like node settings/node configuration values. A node can store parameters as integers, floats, booleans, strings, and lists. One can “get” and “set” parameter values from the command line.

Commands:

ros2 param : displays the type and the current value of the parameter.

ros2 param : allows to change the value of parameters during runtime.

ros2 param dump : displays all the current parameter values of a node.

ros2 param load : allows the loading of parameters from a file to a currently running node.

# Actions

Actions are interpreted as a communication type in ros2. It consists of 3 types : a goal,feedback

and a result. Actions have the same functionality as services with only one difference that

actions can be cancelled. Another difference is that service provides a single response whereas

actions also provide a steady feedback.

Actions use a client-server model, similar to the publisher-subscriber model. . An “action client”

node sends a goal to an “action server” node that acknowledges the goal and returns a stream

of feedback and a result.

Commands:

ros2 node info : To see the list of actions a node provides.

ros2 action list : To identify all the actions in the ROS graph.

ros2 action list -t : helps us to find the type of actions.

ros2 action info : To introspect the action in more detail.

ros2 interface show : To know the structure of the action we use.

ros2 action send\_goal : To send an action goal from the command line(for eg:

rotating the turtle).

In the rover, we use actions for navigation. An action goal could tell the rover to travel to a

position. While the rover navigates to the position, it can send updates along the way (i.e.

feedback), and then a final result message once it’s reached its destination.

So basically actions are like services that allow you to execute long running tasks, provide regular

feedback, and are cancellable.

#rqt\_console

Rqt\_console helps to see the log messages in an organized manner , filter them and save them.

The console window is divided into three parts :

1. Display area: where the log message are displayed.
2. Exclude messages area : where we can filter the messages by excluding security levels or by adding other exclusion features.
3. Highlight messages area: where we can highlight the messages and also add more filters.

In the messages displayed on the console, there are warning levels also known as logger levels which show the severity of the warning or the error.

Logger levels :

The order of loger levels classified on the basis of severity are :

Fatal :  indicates that the system is going to terminate.

Error : indicates significant issues that won’t necessarily damage the system, but are preventing it from functioning properly.

Warn : indicates unexpected activity or non-ideal results

Info :  indicates event and status updates that serve as a visual verification that the system is running as expected.

Debug : details of the entire step-by-step process of the system execution.

We can also set the logger levels(default : info). After setting the default logger level, the console will only show warnings which is either equal to the set value or more than that,

Usually, debug messages are hidden because of their low severity.

# Launching nodes

Instead of opening a new terminal for each ros2 node to be run, launch files feature helps to start and configure multiple ROS 2 nodes and their associated executables all at once. The command for the same is ros2 launch.This tool helps us to run multiple nodes using a single command which makes it easier to code in complex situations.

# Recording and playing back data

The data passed on any number of topics and services is accumulated and then saved on a database. This data can be replayed as many number of times as we want for various tests or even for sharing. The tool for recording this data is ros2 bag. There are 2 types of data : Topic data and Service data.

* TOPIC DATA

Commands :

ros2 topic list : To see the list of system’s topics.

ros2 bag record <topic\_name> : To record the data published to a topic.

ros2 bag info <bag\_file\_name> : To see details about the recording.

* SERVICE DATA

The data shared between service client and server is known as service data. To record the service data we must enable Service Introspection mode on the node.

Commands:

ros2 service list : To see the list of system’s services.

ros2 service echo -- : To check if Service Introspection is enabled on the client and service.

ros2 bag record --service <service\_names> : To record specific services.

ros2 bag record --all-services : To record all services.

ros2 bag info <bag\_file\_name> : To see details about the recording.

ros2 bag play --publish-service-requests <bag\_file\_name> : To play the service data.

CLIENT LIBRARIES

# COLCON

COLCON is the shortform of compile and configure. It is a tool that helps us build , test and install ROS2 packages. A ROS workspace is basically a directory with a specific structure. There is a src subdirectory and inside that the source code of ROS packages are present. COLCON creates other directories parallel to src directory such as : build,install and log. The build directory is used for storing of intermediate files, the install directory is the directory where each package may be installled to and the log directory contains various logging information.

# WORKSPACE

A workspace is a directory which contains ros2 packages. It is basically a location on my computer where I am developing with ros2.

OVERLAY : Creating an overlay is basically creating another workspace so that we can modify the existing packages without changing the original workspace.

Overlays are of a great help when working on a small number of packages because putting everything in the same workspace and then rebuilding the same workspace on every iteration is not feasible.

# PACKAGE

A package serves as a fundamental unit to organize the code. It helps to understand and share the work more efficiently. We can create a package in ros2 using CMake or Python.

CMake consists of :

* CMakeLists.txt : file that describes how to build the code within the package
* include/<package\_name> : directory containing the public headers for the package
* package.xml : file containing meta information about the package
* src : directory containing the source code for the package

Python consists of :

* package.xml : file containing meta information about the package
* resource/<package\_name> : marker file for the package
* setup.cfg : is required when a package has executables, so ros2 run can find them
* setup.py : containing instructions for how to install the package
* <package\_name> : - a directory with the same name as the package, used by ROS 2 tools to find the package, also contains \_\_init\_\_.py

We can place numerous packages within a single workspace, with each package in its own separate folder. It’s also possible to mix different build types like CMake and Python within the same workspace. However, nesting packages inside one another is not permitted.

Commands :

ros2 pkg create --build-type ament\_python --license Apache-2.0 <package\_name> :  For creating a new package in ROS 2.

colcon build --merge-install : COLCON allows building multiple packages in the same workspace at once.

ros2 run my\_package my\_node : To run the executable created in the package.

We can further examine package contents and customize packages.

# PUBLISHER AND SUBSCRIBER

A publisher is responsible for sending messages. It "publishes" data on a particular topic. A subscriber receives messages from a topic. It "subscribes" to data being published on a particular topic. The communication channel through which publishers and subscribers exchange data are called topics. In ros2 we can write codes on publisher and subscriber using C++ or Python or even by using packages like minimal\_publisher and minimal\_subscriber.

# SERVICE AND CLIENT

Nodes communicate via services. the client node is the one that initiates a request, and the service node is the one that processes the request and provides the response. We can write service-client in various ways like using C++,Python or by using packages like minimal\_client and minimal\_service.

# CUSTOM INTERFACES

Creating custom interfaces allows us to define our own messages and service types. Interfaces can only be defined in CMake packages alongwith python / C++ nodes.

Commands:

Set : To neatly list all the interfaces.

Firstly, we have to define an interface that is by creating either a msg file , srv file or action file. This file determines the data structure. Then we use C++/Python to code using ros2 tools for some specific task. After writing the code, we implement nodes for publishing and subscribing data or call services or even action goals. Lastly, we launch the nodes and check whether the are communicating correctly with the right interface.

# PARAMETERS

Parameters are helpful in configuring nodes and adapting to their behviour without altering the code. A node can be developed with a custom parameter that can be configured either via a launch file or by command line. To enable this functionality, we can add the necessary dependencies, define the executables, and configure a launch file in the package settings. After building the package, we can run the node and see the parameter in action.

# ros2doctor

The tool ros2doctor helps users identify and troubleshoot issues in the ROS 2 setup. ros2doctor is part of the ros2cli package.

UserWarning : Indicates that something is configured in a way that’s not ideal.

If we use --report alongwith ros2doctor, we can analyse the warnings and error in detail.

# PLUG-IN

Pluginlib : It is C++ library for loading and unloading plugins from within a ROS package.

Plugins are dynamically loadable classes that are loaded from a runtime library i.e by the concept of a shared object. Plugin eliminates the need of explicitly linking the application against the library containing the classes. Instead, pluginlib can open a library containing all the exported classes which makes it easier to code.

Plugin instances can be created using various commands and plugin loader has a way to find that library and to know what to reference within that library by creating an XML file. Finally, exporting the plugin is made available using CMakeLists.txt ( ros1 exporting was done by : package.xml )