**ROS DOCUMENTATION**

Rosdep - used to install system dependencies

**Introduction**

You can think of ROS as a software which you use to program robots. It is open source and pretty open ended as well. It gives you a structure to robotic programming which allows everyone to use robots and hence is not limited to robotic engineers. You can use this Ubuntu, Mac OS, Debian and Windows.

**Preliminaries**

**THE ROS GRAPH**

All the messages in ROS are transmitted to one point to another. These points are what we call nodes. Suppose we are sending a message from a talker to a listener then both of them are called nodes. IRL, you can think of the camera, sensors, motors, etc as nodes. A ROS graph is a representation of the web of these nodes and how they are connected with each other.

**ROSCORE**

But how are these nodes connected? The answer is ROSCORE. It is a master node that connects other nodes. Nodes register their information to roscore and it connects that node with the other nodes. It needs to be running all the time. It only connects the node and not transmit the message from node to node.

Every ROS node has an environmental variable in its process called ROS\_MASTER\_URI which will have the port number and the host name. We need this to connect to the network.

**Rosparam** - roscore has a parameter server which is accessed by rosparam. It stores arbitrary information such as robot structure, parameter algorithms and other information of the robots.

**CATKIN, WORKSPACE AND ROS PACKAGES**

1. catkin - build system with cMake files
2. catkin\_init\_workspace - creates the CMakeLists.txt file in the src folder where we invoked it.
3. catkin make - creates build and devel files.

* BUILD: stores libraries and executable commands
* DEVEL: setup files, other files and directories

1. catkin\_create\_pkg <pkg\_name> - this creates a ROS package named pkg\_name which is stored in the lsrc file.

* This command creates the package along with a directory with the same package name, a CMakeLists.text file, package.xml file and an SRC directory in it.

1. Colcon is an iteration of catkin
2. A package is an organizational unit for your ROS 2 code. If you want to be able to install your code or share it with others, then you’ll need it organized in a package.

**ROSRUN**

Rosrun is used to locate a package because it can be difficult to chase all the directories in the file system to locate a package. We can also pass additional arguments based on the parameters we want.

* **SYNTAX:** user@hostname rosrun <package executable> [args]
* To do this, we will first run roscore in one terminal window and then run rosrun for all the other nodes in another terminal window.

**NAMES, NAMESPACE AND REMAPPING**

**ROSLAUNCH**

1. Rosrun operates on nodes while roslaunch operates on launch files.
2. Launch files has a suffix of .launch.

* **SYNTAX:** roslaunch <package\_name> [args]

**tf: COORDINATE TRANSFORMS**

1. A way to manage the coordinate frames and transform. Tf also works with publishing, subscribing, remembering or computing of transforms. It has libraries that do so.

**Poses -** An orientation is a vector of roll, pitch and yaw. The coordinate (position, orientation) is a pose. It is 6D because of x,y,z, roll, pitch and yaw dimensions.

**Transform (tf) -** It is a node that will take carry the information of some movement of function of the robot. A tf message is called tf/tfMessage sent over /tf topic. The tf/tfMessage message contains a list of transforms that specify the parent and child frames, their relative position and orientation, and the timestamp of each transform.

**Topics**

Topic is a message!

STEPS:

* Before nodes transmit data, it will first announce or advertise the topic name and the type of messages that are going to be sent.
* The message is then published on the topic.
* The receiving node will then subscribe to that topic by making a request to roscore.

All the messages on the same topic must be of the same data type.

**PUBLISHING TO A TOPIC**

We need to add permission to execute the file so we use the command **chmod u+x topic\_publisher.py** and if we are using another package, we have to tell the ROS build system that and therefore add a dependency in our package.xml file (**<depend package = “std\_msg” />**)

How to verify that your node works?

We use the command **rostopic list** and it will give a list of topics that are running currently. In another terminal we will run the topic as **rosrun basics topic\_publisher.py.** Basics directory is added in this command because if write a node, we can connect it with the package.xml file. This way when the node is running we can verify it using the rostopic command.

We can also see the messages being published(**rostopic echo counter -n 5**) instead of just seeing if the node is running. Here the topic name is counter and the -n 5 flag tells rostopic to only print out 5 messages. We can also use **rostopic find std\_msgs/Int32.**  It will give you all the topics that are of package std\_msgs and message type Int32. We have to provide both package name and the message type for this to work.

**SUBSCRIBING TO A TOPIC**

**LATCHED TOPICS**

If we publish a topic and that topic is missed by the server, then that server will not receive the message. This means that the subscriber will not receive anything. In this case we use latched topics. It means that the last message will be automatically sent to the subscriber when it connects.

**Example -**

Pub = raspy.Publisher(‘map’, nav\_msgs/OccupancyGrids, latched = True)

**Services**

Services are similar to messages. It has both an input and an output.

string words

---

uint32 count

The first sentence is the input (it is a string), when the input ends, we add 2 dashes and then the output is the integer.

**Other ways to return value from a service:**

**Actions**

Actions are another type of ROS messages. They are used for time-oriented scenarios where we need to know what is happening to a robot while it gets to the goal position. Example of actions - goto\_position

While services are more like a “work done” message, actions give intermediate feedback about what is happening even before the goal is achieved.

Steps to creating an action -

1. Define the message format for goal, result and feedback

**Workspace and src and packages**

* The workspace is where you keep all your code, packages, files, nodes, etc. Every project should be in a different workspace (for clarity). The main directory of our workspace will contain our source subdirectory.
* This src directory will contain our install, build, devel and log files. These files are created in your src folder once your build the workspace using **colcon build** command. Colcon is an iteration of ROS build tools like catkin\_init\_workspace, catkin\_make, commands.
* The packages contain your nodes. All packages will be in the in the src file. Your .py or .c files are in these packages. To create a package you will use the **pkg create**

**Subsystem**

Subsystems in robotics can be divided into 3 main categories: actuation (how the robot’s motors, wheels, arms move), sensing (sensor hardware like camera and laser sensors) and computing (connects the actuators and sensor subsystems and perform some computation that will make the robot perform useful tasks).