https://wiki.bu.edu/roboclass/index.php?title=ROS_with_Python

6 min read

ROS with Python - Introduction to Robotics

The module rospy constitutes the main interface between Python and ROS. The following are the most important functions and classes from this module.

Initializes a node (i.e., it registers it with the ROS Master node, and becomes listed under rosnode list). The argument name must be a string. If anonymous is set to True, a random number is appended to the name, so that it becomes unique even if the same script is called again (see talkerClass.py for instance). It will throw an error if the master is not running.

Returns True if the node is running, or False if ROS is asking to terminate the node (e.g., because the user pressed Ctrl+C). It is typically used in a while loop for nodes that periodically publish data (see talker.py for an example).

std_msgs.msg[edit]

sensor_msgs.msg[edit]

geometry_msgs.msg[edit]

Headers and stamped messages[edit]

A class with a method .sleep() that pauses execution for a time equal to 1/hz, where hz (Hertzs) is the argument passed to the constructor. This method is typically used at the end of the while loop mentioned for is_shutdown(), and it is used to delay the execution of the iteration so that, approximatively, the loops runs at a frequency of "hz" Hertzs.

Shows information/debug/warning/error information on the console. The string str is the content of the messages. This logs are also captured by ROS and can be automatically recorded. It is a good idea to use these methods instead of raw "print" commands (the latter, for instance, do not appear when the node is launched by a launch file). See the [[1][tutorial on logging]] for additional information.

Creates a publisher object. topicname must be a string with the name of the topic on which to publish. Type is the name of a Python class corresponding to a message type. See explanation below on message types for details. See also section about queuing below.

To publish a message, one firs needs to create a publisher object (only once per node), for instance, from the talker.py file in the provided repository:

```
pub = rospy.Publisher('chatter', String)
```

Every time a new message needs to be published, you need to create an instance of the message type (i.e., an object from the corresponding class), set it, and then pass it to the publish method of the publisher object. For instance (again, from the talker.py file):

```
hello_str = "Hello world."
pub.publish(hello_str).
```

Note that, in this simple example, we use one of the fundamental types (a string), which we do not to explicitly instantiate. An alternative, more general example is the following (which is an excerpt from the file zigzag_twist.py):

```
msg=Twist()
msg.linear.x=0.5
self.pub.publish(msg)
```

Note that the second example requires importing the message type Twist from the geometry_msgs package (from geometry_msgs.msg import Twist).

Creates a subscriber. topicname and Type correspond to those in rospy.Publisher(). callback is the name of a function of the form "def callback(msg):".

After creating the subscriber, there is no need to directly "call it" as you need to do for a publisher. Instead, ROS will automatically call the callback function whenever a message is available in the queue, filling the msg structure with the data from the message.

Notice that msg will be of the type "Type" specified in the call to rospy.Subscriber. For instance, in listener.py the subscriber is created as

```
    rospy.Subscriber('chatter', String, callback)
```

In this case the message if of type String, imported from std_msgs.msg. According to the [[2][documentation]], this type has a single field, called data. As such, the following callback function will log the content of the message.

```
def callback(msg):
    rospy.loginfo(' I heard %s', msg.data)
```

Note[edit]

Creating the subscribers takes some time. Therefore, there is an unavoidable delay between the call to `rospy.Subscriber` and the first callback invocation; as a result, messages that are published during this delay will be lost. This will be especially apparent for nodes that are supposed to subscribe to their own publishers.

Class to handle time in ROS. They contain two fields: a variable for seconds, and another for nanoseconds. The Time class is for "absolute" times (i.e., a time with respect to "time zero"), while the Duration class is for time intervals (the difference between two absolute times); this difference is similar to the difference between points and vectors (TODO: insert internal link).

The most important methods are

```
- .now() (only for the Time class): return a Time object with the current ROS time
- .to_sec()/.to_nsec(): return the content of the object as a floating point variab
```

Time and Duration arithmetic[edit]

The addition and subtraction operations on time and duration objects are defined to respect their semantic meaning:

```
duration + duration = duration
duration - duration = duration
time + duration = time
time - time = duration
time + time is undefined
```

Level ***: Queue sizes

Every publisher and every subscriber has a buffer with a maximum number of messages. This can be specified with a queue_size named argument to the constructors for Subscriber and Publisher objects in rospy.

TODO: explain message queueing, dropping, and possible delays. Message types

Typically, this class is imported by an instruction of the form from <package_name>.msg import (e.g., from std_msgs.msg import String). The most common standard packages from which to import message types are the following: - [[3][std_msgs]] : all the fundamental message types (integer and floating point numbers, strings, times, etc) - [[4][geometry_msgs]]: messages for describing geometric quantities (points, vectors, poses, velocities, etc.) - [[5][sensor_msgs]]: types for data coming from a sensor (IMU, images, point clouds, etc.)

TODO: fix links above.

Generated with Reader Mode