

# A Gentle Introduction to ROS

Chapter: Recording and replaying messages

Jason M. O’Kane

Jason M. O’Kane  
University of South Carolina  
Department of Computer Science and Engineering  
315 Main Street  
Columbia, SC 29208

<http://www.cse.sc.edu/~jokane>

©2014, Jason Matthew O’Kane. All rights reserved.

This is version 2.1.6 (ab984b3), generated on April 24, 2018.

Typeset by the author using  $\text{\LaTeX}$  and `memoir.cls`.

ISBN 978-14-92143-23-9

# Chapter 9

---

## Recording and replaying messages

*In which we use bag files to record and replay messages.*

One of the primary features of a well-designed ROS system is that parts of the system that *consume* information should not care about the mechanism used to *produce* that information. This architecture is easy to see in the publish-subscribe model of communication that ROS primarily uses. A good subscriber node should work any time the messages it needs are being published, regardless of which other node or nodes is publishing them.

This chapter describes a tool called `rosvbag` that is a concrete example of this kind of flexibility. With `rosvbag`, we can **record** the messages published on one or more topics to a file, and then later **replay** those messages. Taken together, these two capabilities form a powerful way to test some kinds of robot software: We can run the robot itself only a few times, recording the topics we care about, and then replay the messages on those topics many times, experimenting with the software that processes those data.

### 9.1 Recording and replaying bag files

The term **bag file** refers to a specially formatted file that stores timestamped ROS messages. The `rosvbag` command can be used both to record and to replay bag files.<sup>1</sup><sup>2</sup>

**Recording bag files** To create a bag file, use the `rosvbag` command:

```
rosvbag record -O filename.bag topic-names
```

---

<sup>1</sup><http://wiki.ros.org/rosvbag>

<sup>2</sup><http://wiki.ros.org/rosvbag/CommandLine>

If you don't give a file name, `rosbag` will choose one for you based on the current date and time. In addition, there are a few other options for `rosbag record` that might be useful.

- ☞ Instead of listing specific topics, you can use `rosbag record -a` to record messages on every topic that is currently being published.



*Recording every topic is no problem for the kinds of small-scale systems that appear in this book. However, it can be a surprisingly bad idea on many real robot systems. For example, most robots equipped with cameras have nodes that publish multiple topics containing images that have undergone varying amounts of processing and varying levels of compression. Recording all of these topics can create staggeringly huge bag files very quickly. Think twice before using `-a`, or at least keep an eye on the size of the bag file.*

- ☞ You can enable compression in the bag file using `rosbag record -j`. This has the usual tradeoffs of compression: Generally smaller file sizes in exchange for slightly more computation to read and write. In the author's opinion, compression generally seems to be a good idea for bag files.

When you've finished recording, use `Ctrl-C` to stop `rosbag`.

**Replaying bag files** To replay a bag file, use a command like this:

```
rosbag play filename.bag
```

The messages stored in the bag file are then replayed, in the same order and with the same time intervals between them as when they were originally published.

**Inspecting bag files** The `rosbag info` command can provide a number of interesting snippets of information about a bag:

```
rosbag info filename.bag
```

An an example, here's the output for a bag that the author recorded while writing the next section:

```
path: square.bag
version: 2.0
duration: 1:08s (68s)
```

```

start: Jan 06 2014 00:05:34.66 (1388984734.66)
end: Jan 06 2014 00:06:42.99 (1388984802.99)
size: 770.8 KB
messages: 8518
compression: none [1/1 chunks]
types: geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
      turtlesim/Pose [863b248d5016ca62ea2e895ae5265cf9]
topics: /turtle1/cmd_vel 4249 msgs : geometry_msgs/Twist
      /turtle1/pose 4269 msgs : turtlesim/Pose

```

In particular, the duration, message count, and topic lists are likely to be interesting.

## 9.2 Example: A bag of squares

Let's work through an example to get a feel for how bag files work.

**Drawing squares** First, start roscore and the usual `turtlesim_node`. From the `turtlesim` package, start a `draw_square` node:

```
roslaunch turtlesim draw_square
```

This node resets the simulation (by calling its `reset` service) and publishes velocity commands that drive the turtle in a close approximation of a repeating square pattern. (You could also use any of the nodes we've written to publish velocity commands. The prefabricated `draw_square` program is a good choice because unlike, say, `pubvel`, it's easy to see the structure of the motions the turtle makes.)

**Recording a bag of squares** While the turtle is drawing squares, run this command to record both the velocity commands and turtle pose messages:

```
roslaunch turtlesim draw_square --record square.bag
```

The initial output will let you know that `roslaunch` is subscribing to `/turtle1/cmd_vel` and to `/turtle1/pose`, and that it is recording to `square.bag`. At this point, the graph (as shown by `rqt_graph`) would look something like Figure 9.1. The new and interesting part is that `roslaunch` has created a new node, called `/record_...`, that subscribes to `/turtle1/cmd_vel`. The graph shows that `roslaunch` records messages by subscribing to the topics you ask for, just like any other node, using the same mechanisms that we learned in Chapter 3.

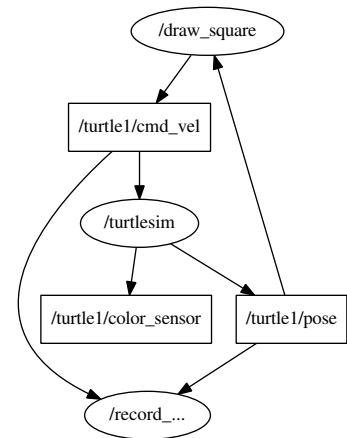


Figure 9.1: The graph of nodes and topics while rosbag record is running.




*Nodes created by rosbag use anonymous names, which we discussed in Section 5.4. In this chapter, we've replaced the trailing numbers from those names with ellipses (...) for brevity. Note that the use of anonymous names means that we can run multiple rosbag record instances at once, if we choose to.*

**Replaying the bag of squares** After this system has run for a while—a minute or two should be plenty—kill rosbag to stop the recording and kill `draw_square` to stop the turtle's drawing. Next, let's replay the bag. After ensuring that roscore and turtlesim are still running, use this command:

```
rosbag play square.bag
```

Notice that the turtle will resume moving. This happens because rosbag has created a node named `play_...` that is now publishing on `/turtle1/cmd_vel`, as shown in Figure 9.2. As we would expect, the messages that it publishes are the same ones that `draw_square` originally published.

Figure 9.3 illustrates drawings that might result from this sequence of operations. Depending on how carefully you've thought about what rosbag does, these drawings might be a bit surprising.

 The squares drawn during `rosbag play` might not be in the same place as squares drawn during `rosbag record`. Why not? Because rosbag only replicates a sequence of messages. It does not replicate the initial conditions. The second batch of squares,

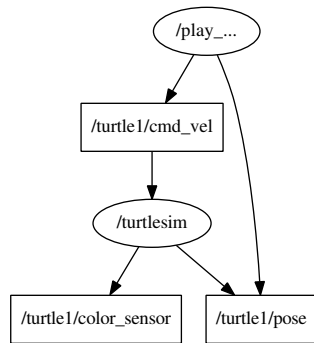


Figure 9.2: The graph of nodes and topics while rosbag play is running.



Figure 9.3: [left] A turtlesim turtle responding to movement commands from draw\_square. Those movement commands are also recorded by rosbag. [right] By replaying the bag, we can send the same sequence of messages to the turtle.

drawn during rosbag play, began from wherever the turtle happened to be at the time we executed that command.

- 👉 The original draw\_square and rosbag play can send the turtle to different places, even though the bag contains the pose data from the /turtle1/pose topic. Why? Quite simply, because in this example, no one (other than rosbag record) subscribes to /turtle1/pose. There's a difference between someone (in this case, rosbag play) publishing data about where the turtle is, and the turtle actually being there. The pose data from the bag file is ignored.

In fact, when both turtlesim\_node and rosbag play are running, the messages on /turtle1/pose can be downright contradictory. Listing 9.1 shows an example of four messages published on this topic in rapid succession, within less than a second. Notice the abrupt changes in the  $y$  coordinate. It is fortunate that no nodes are subscribed to this topic, because any such node would likely have trouble making sense of its messages.


```
1 x: 5.93630695343
2 y: 4.66894054413
3 theta: 5.85922956467
4 linear_velocity: 0.0
5 angular_velocity: 0.40000000596
6 ---
7 x: 5.56227588654
8 y: 7.4833817482
9 theta: 4.17920017242
10 linear_velocity: 0.0
11 angular_velocity: 0.40000000596
12 ---
13 x: 5.93630695343
14 y: 4.66894054413
15 theta: 5.865629673
16 linear_velocity: 0.0
17 angular_velocity: 0.40000000596
18 ---
19 x: 5.56227588654
20 y: 7.4833817482
21 theta: 4.18560028076
22 linear_velocity: 0.0
23 angular_velocity: 0.40000000596
24 ---
```

---

Listing 9.1: Four successive messages published on `/turtle1/pose` in short period time, with conflicting reports about the location of the turtle. Notice the large difference in the `y` coordinates. The conflict occurs because both `turtlesim` and `rosbag play` are publishing on this topic.



*The lesson is to avoid (or, at a minimum, to be very careful with) systems in which both `rosbag` and “real” nodes are publishing on the same topic.*

 Figure 9.3 also illustrates that service calls (see Chapter 8) are not recorded in bag files. If they were, then the bag might include some record of when `draw_square` called `/reset` before beginning to send messages, and the turtle would have returned to its starting point.



## 9.3 Bags in launch files

In addition to the `rosvag` command that we have seen already, ROS also provides executables named `record` and `play` that are members of the `rosvag` package. These programs have the same functions and accept the same command line parameters as `rosvag record` and `rosvag play`, respectively.

This means, for one thing, that it is possible—but needlessly verbose—to record or replay bag files using `rosvun`, like this:

```
rosvun rosvag record -O filename.bag topic-names
rosvun rosvag play filename.bag
```

More importantly, these `record` and `play` executables make it easy to include bags as part of our launch files, by including the appropriate node elements. For example, a `record` node might look like this:

```
<node
  pkg="rosvag"
  name="record"
  type="record"
  args="-O filename.bag topic-names"
/>
```

Likewise, a `play` node might look like this:

```
<node
  pkg="rosvag"
  name="play"
  type="play"
  args="filename.bag"
/>
```

Aside from the need to pass `args` for their command lines, these nodes don't need any unusual treatment from `rosvlaunch`.



*At this point, you might be surprised to see the chapter ending without any discussion of how to use bag files from C++ programs. In fact, there does exist an API for reading and writing bag files.<sup>3</sup> However, that API is really only needed for specialized applications. For simple recording and playback operations, the `rosvag` command line interface is quite sufficient.*

## 9.4 Looking forward

This concludes our guided tour of the essential elements of ROS. The next chapter wraps things up by briefly mentioning a few other topics that show up frequently in real ROS systems.

---

<sup>3</sup><http://wiki.ros.org/rosbag/CodeAPI>