ROBOTCRAFT©

ROSin

INGENIARIUS

Robot Operating System

6.1: Introduction to ROS

Panagiotis Karfakis
Robotics Researcher
panagiotis@ingeniarius.pt

Craft #6

30/07/2021

Objectives and Layout (ROS Day 2)

- In the previous session we have:
  - Gone through the basic ROS topics (which we will recap);
  - Implemented a very simple controller for turtlesim.
- Today we will:
  - Improve our controller;
  - Have some extra time for your doubts and exploring different functionality.
- We're still:
  - Using checkpoints;
  - Doing everything as practical as possible;
  - Going slowly;
  - Covering a lot of material.

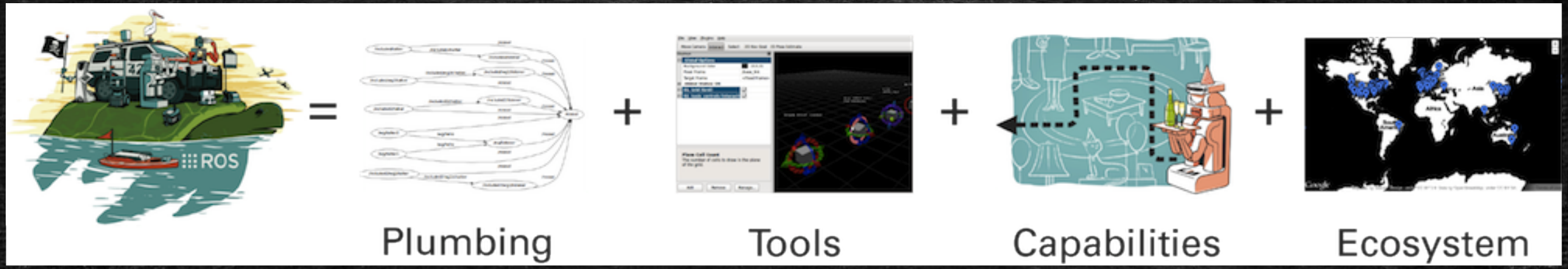
Recap: Basic ROS

Why ROS?

- What are robots?
  - Hardware + Software
- We want to have the same “intelligence” in different robots!
- We need a unified software suite that abstracts away the particularities of each robot.

ROS

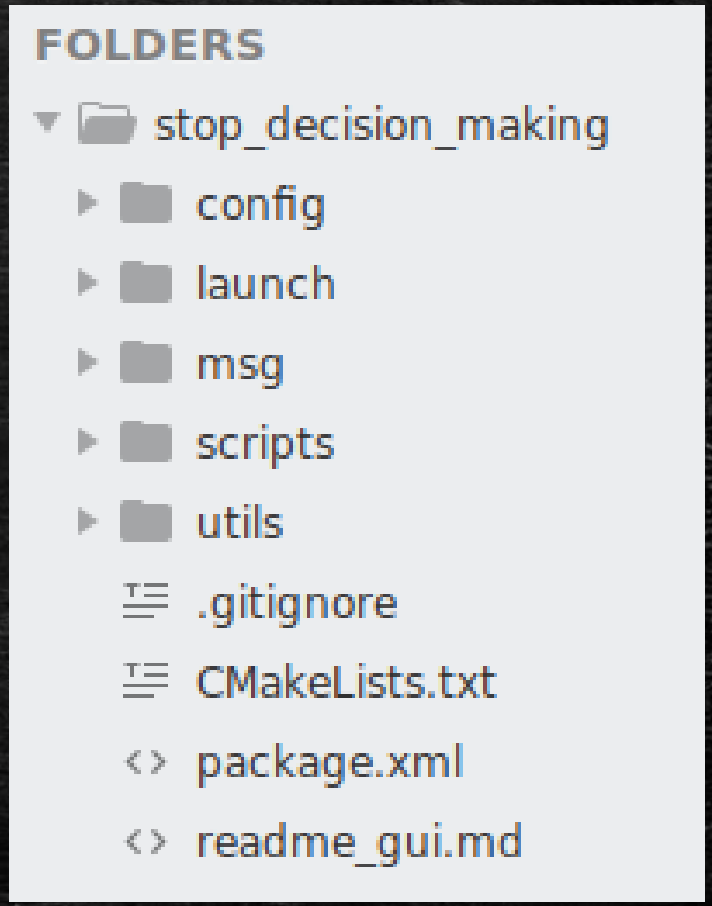
What is ROS?



- ROS is a suite of formalisms, tools and libraries (a **framework**) for implementing robot-related software;
- The ROS conventions allow us to write robot-abstract software that can be easily ported from platform to platform;
- By greatly expediting development, it became very, very popular.

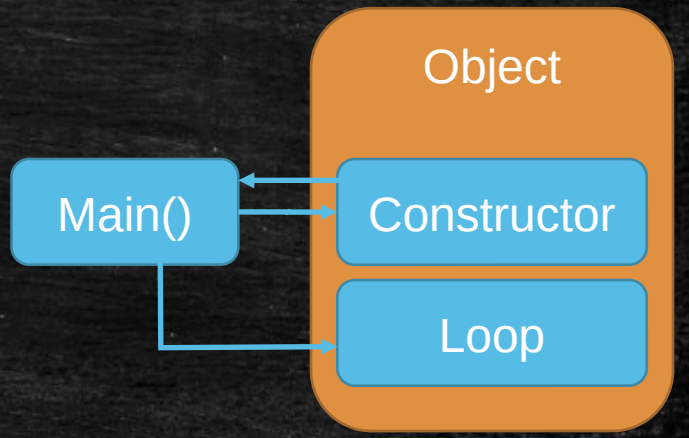
What are packages? Why do we care?

- ROS packages are folder structures that keep all your code neat, tidy, portable and distributable;
- Different sub-folders are created as you generate content;

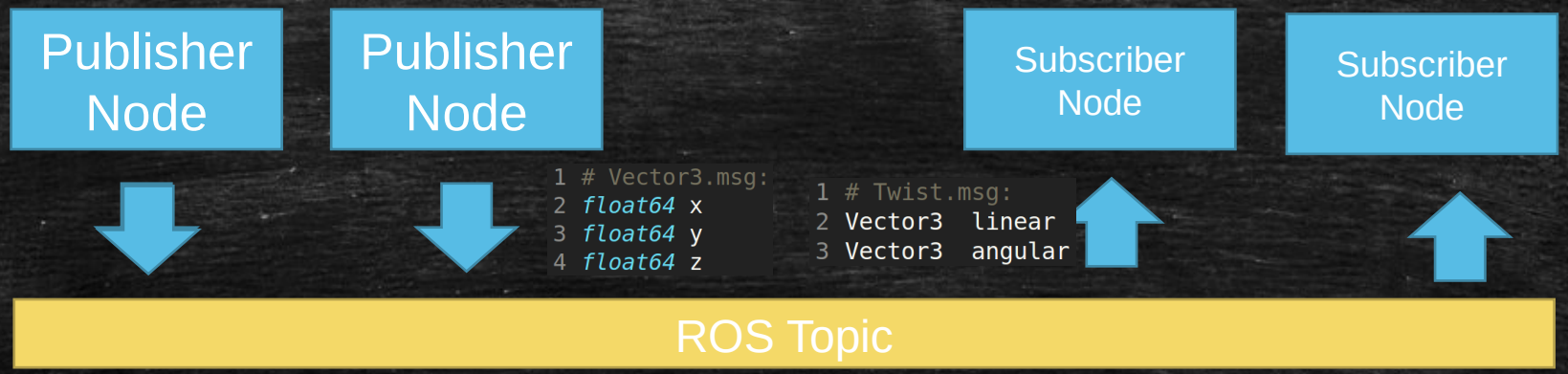


Basic Structure of a ROS Node

- Now that we have something that builds against the ROS libraries, we should write a first version of our node.
- The structure of a ROS node is very up to the programmer, but I like to structure it as follows:
  - Encapsulate all functionality in a class (this allows us to elegantly maintain state, and maybe re-use our module in the future if necessary);
  - Use the constructor (or an initialization function) to create all necessary topics, etc…
  - Use a run() function to run the node's loop.



Topics: (kind of) Like WhatsApp Groups



- Any number of publishers can publish into the topic, and any number of subscribers can subscribe;
- **Any publisher can publish messages at any time**;
- **All subscribers will receive the messages**;
- Each topic has a pre-defined **message type**.

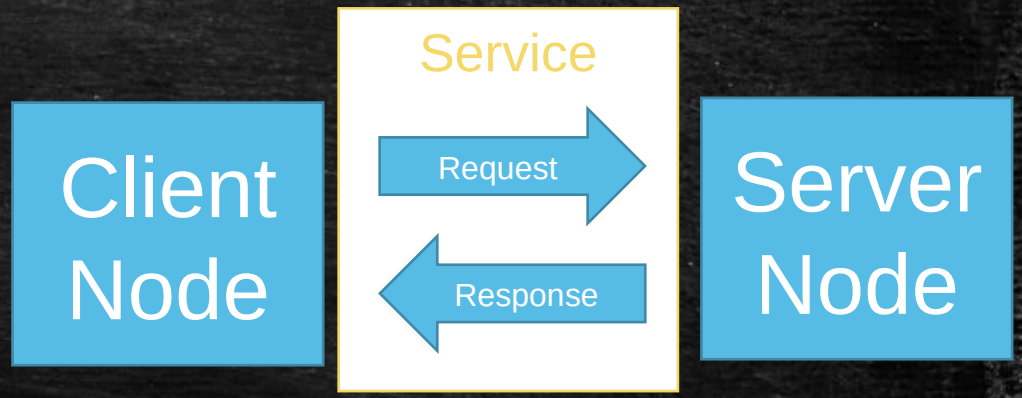
Launch Files: What are they?

```
<launch>

    <node pkg="turtlesim" type="turtlesim_node" name="turtlesim" output="screen"/>

    <!-- TODO: launch our node and remap the cmd_vel topic -->

</launch>
```

- Launch files specify a set of nodes that will run at the same time.
- Launch files can be immensely useful as your system grows.
- Each launch file can specify a number of nodes that run, and can also import other launch files.

Services: Introducing Synchronicity

- Services allow for functionality of a different node to be called synchronously.
  - What does synchronously mean?
- Client sends a request and waits for a response.
- Can be used for (so many things):
  - Decouple nodes (dependencies, etc);
  - Offload processing power;
  - Implement synchronous client-server communication protocols.
- Can be defined with essentially the same format as messages with one key difference: a request and a response need to be defined.




```
1 # Goal: a pose where we want to be
2 geometry_msgs/PoseStamped target_pose
3 ---
4 # Result is not needed since the
5 # service response will tell us
6 # if we were successful.
```

Services: Where do we (ING) use a service? Example

Services: Tasks

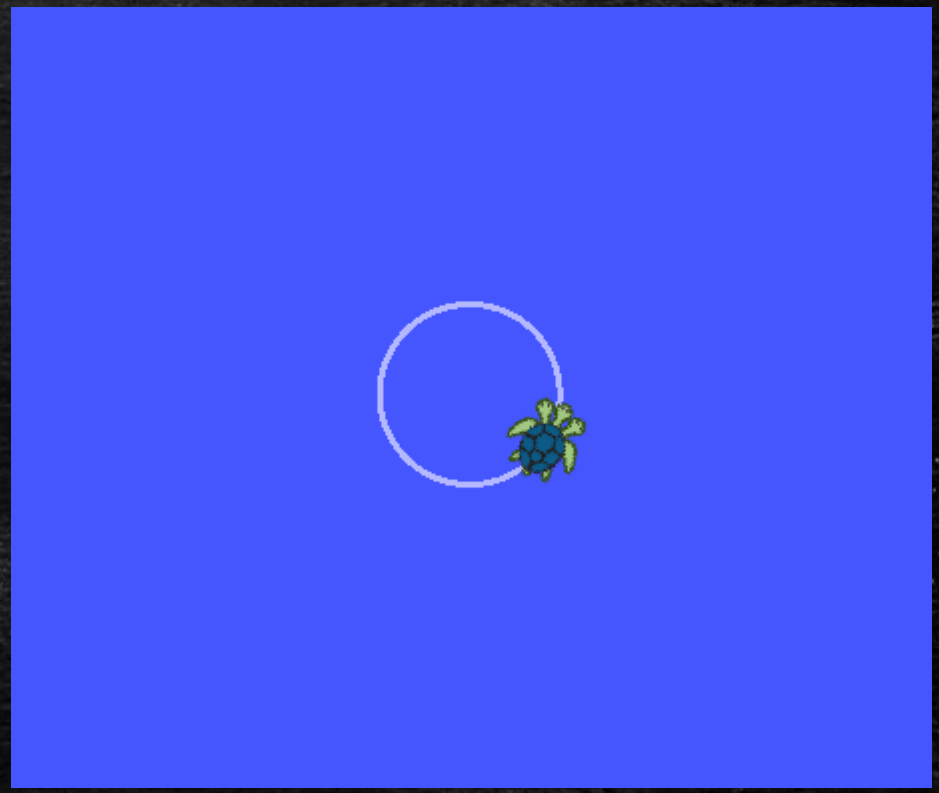
Using the ROS Documentation, create a simple Service (link) that receives a request float and return a confirmation string response.

  - P.S. do not forget to create a .srv file before.

- Within the same Service, publish a /cmd_vel to your turtle (using the float received as x or y linear velocity) in order to make it move.
  - How?
  - Create a simple Publisher (http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29%28plain%20cmake%29)
- Once it runs with rosrun, make it runnable with the test_controller.launch.

Checkpoint: Your Turtle Should Move!

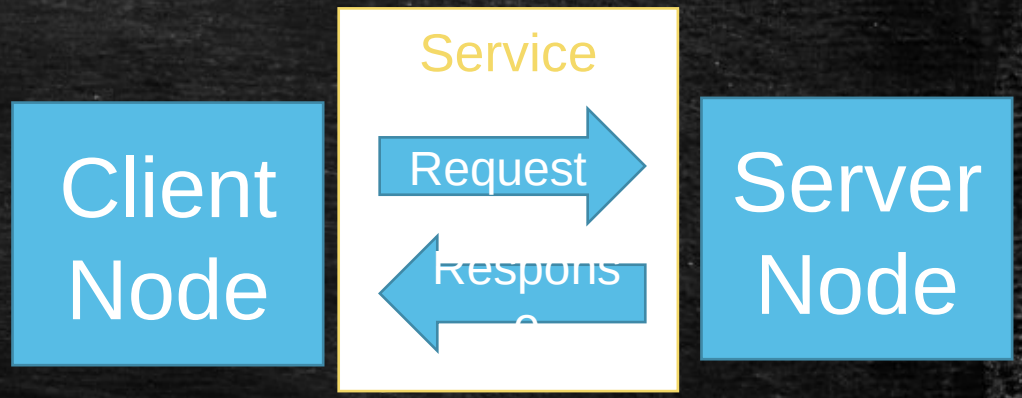
- roslaunch turtlebot_control test_controller.launch
- You should see a turtle showing up and moving in circles.
- If your turtle moves, we can move on to Day 2.



ROS Services

Services: Introducing Synchronicity

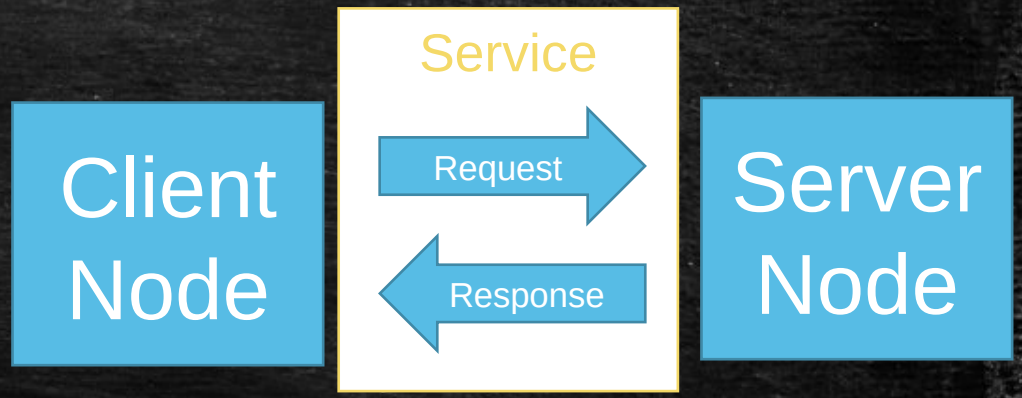
- Services allow for functionality of a different node to be called synchronously.
  - What does synchronously mean?
- Client sends a request and waits for a response.
- Can be used for (so many things):
  - Decouple nodes (dependencies, etc);
  - Offload processing power;
  - Implement synchronous client-server communication protocols.
- Can be defined with essentially the same format as messages with one key difference: a request and a response need to be defined.




```
# Goal: a pose where we want to be
geometry_msgs/PoseStamped target_pose
---
# Result is not needed since the
# service response will tell us
# if we were successful.
```

Checkpoint: Adding a Service to our Controller

- What would be an interesting service to add to our controller? Why?
- I've decided for you: we will add a service to stop and re-start the robot.
- We will need to:
  - Include the correct (Empty) service;
  - Implement a handler function that changes a flag;
  - Extend our publishing loop to take the flag into account;
  - Add a spinOnce to our execution loop.
- Let's write some more code!

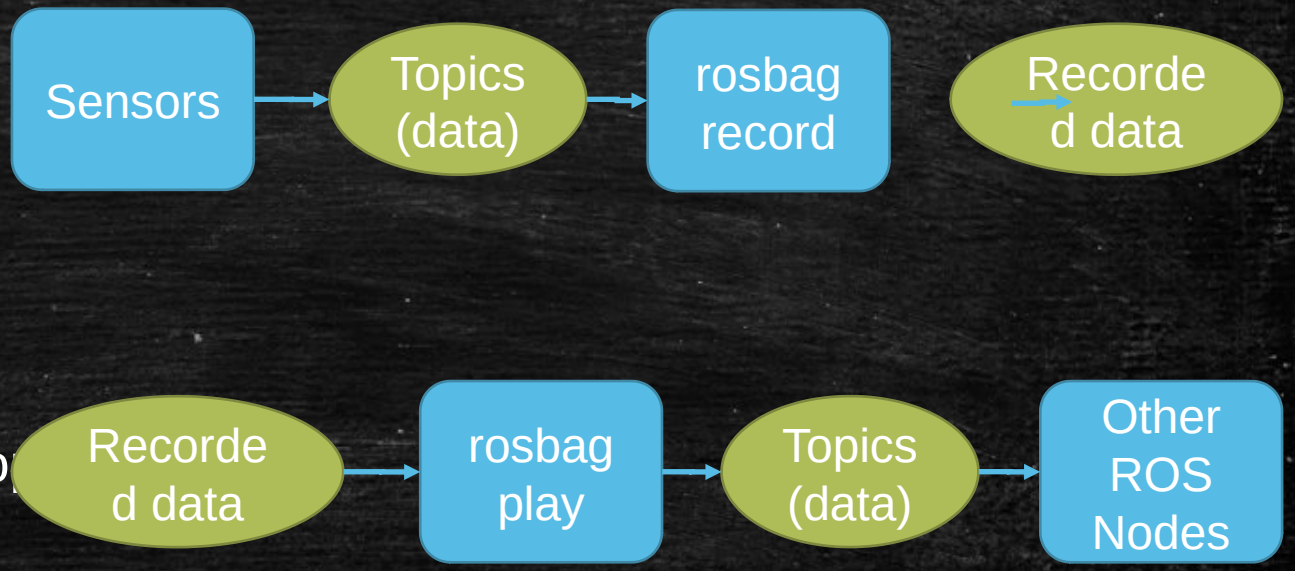



```
1 # Goal: a pose where we want to be
2 geometry_msgs/PoseStamped target_pose
3 ---
4 # Result is not needed since the
5 # service response will tell us
6 # if we were successful.
```

ROS Bags

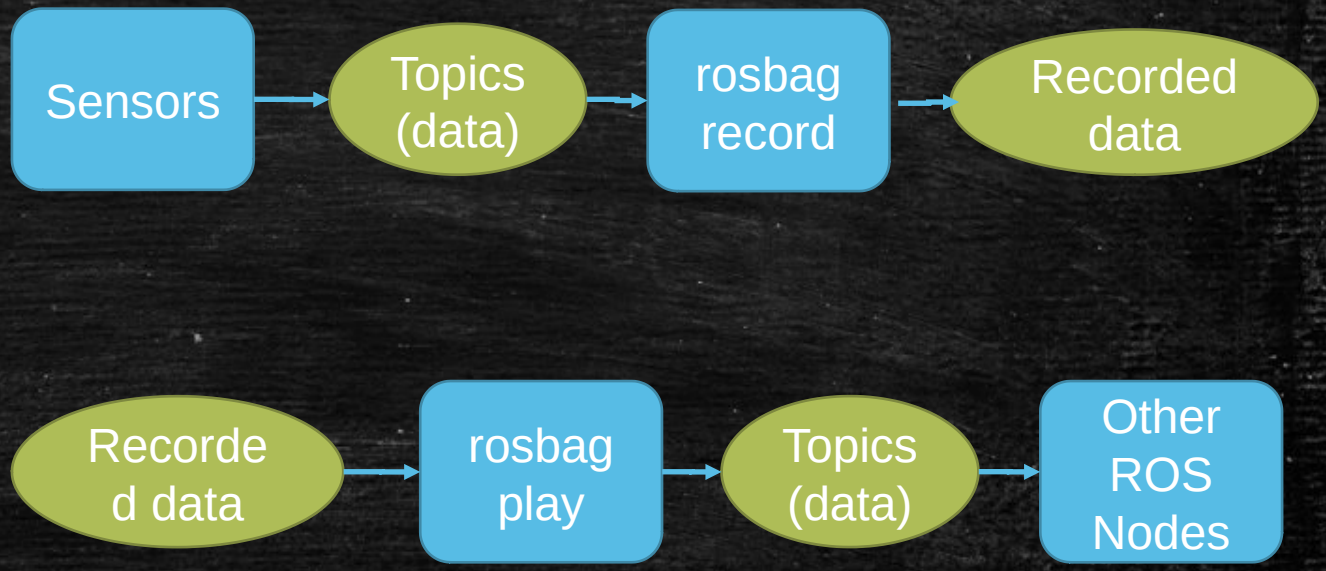
What are bags?

- Most information in ROS flows in topics;
- Bags allow us to record this information in a timestamped manner and play it back at will;
- Bags are extremely useful:
  - To record and play back data from an expensive experiment;
  - To allow us to develop a module on to well-known data;
  - …



Rosbag: Command Line Usage

- Record data: rosbag record <topics>
- Play data back: rosbag play <topics>
- There is a very large number of options and flags you should look at.
  - Reference: http://wiki.ros.org/rosbag/Commandline



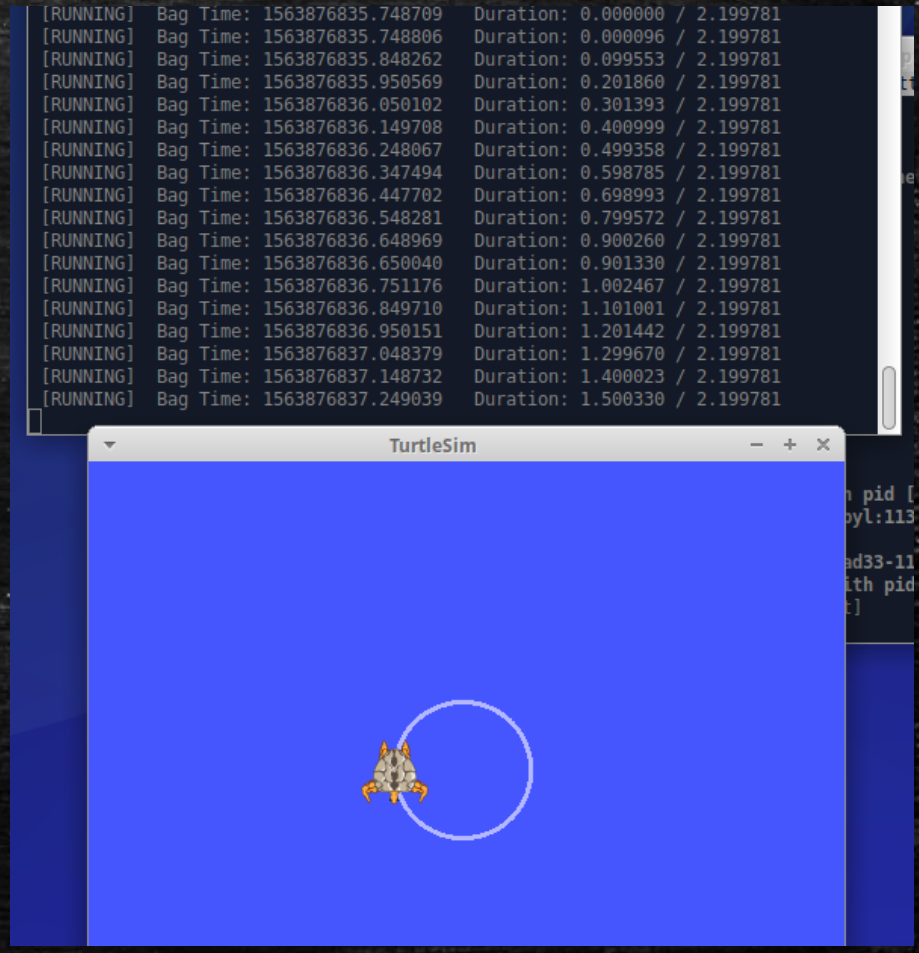
Checkpoint: Let’s Record Some Data

- Run turtlesim_node and teleop;
- rosbag record /turtle1/cmd_vel and let it run for a few seconds;
- Run rosbag info 20<tab>. What do you see?
- Run rosbag compress 20<tab>. What happens?

```
gondsm@chernobyl:~$ rosbag record /turtle1/cmd_vel
[ INFO] [1563876835.435986885]: Subscribing to /turtle1/cmd_vel
[ INFO] [1563876835.445877130]: Recording to 2019-07-23-11-13-55.bag.
^Cgondsm@chernobyl:~$ rosbag info 2019-07-23-11-13-55.bag
path:        2019-07-23-11-13-55.bag
version:     2.0
duration:    2.2s
start:       Jul 23 2019 11:13:55.75 (1563876835.75)
end:         Jul 23 2019 11:13:57.95 (1563876837.95)
size:        8.2 KB
messages:    23
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
topics:      /turtle1/cmd_vel   23 msgs    : geometry_msgs/Twist
```

Checkpoint: Let's Play it Back!

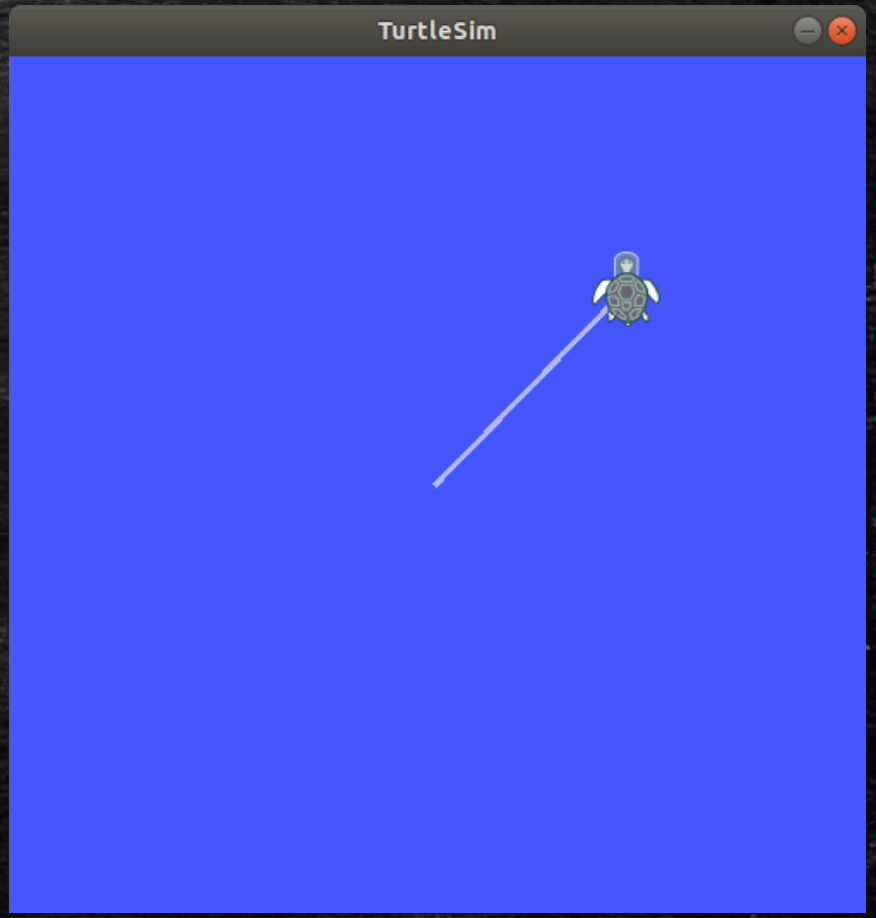
- Now run only the turtlesim node
  - roscore
  - rosrun turtlesim turtlesim_node
- And play back your bag: rosbag play 20<tab>
- What happened? Why?
  - Look at the topics with rostopic;
  - Try listing and echoing topics.



Recap: ROS Exercise

Checkpoint: Move the turtle to 8.0 8.0

- go_to.cpp
  - Pub to */cmd_vel*
  - Sub to */turtle1/pose*
- CMakeLists.txt
- Start.launch
  - turtlesim_node
  - go_to (remap: */cmd_vel to /turtle1/cmd_vel*)



Controlling Two Turtles

Creating a New Launch File

- Navigate to github.com/gondsm/turtlebot_control_blanks and copy the launch/two_turtles.launch file.
- We basically need to fill in the blanks:
  - Name the controllers with different names;
  - Remap cmd_vel;
  - Set speed params;
  - Set a new param that controls the name of the new turtle.
- Let's code!

```
<launch>

    <node pkg="turtlesim" type="turtlesim_node" name="turtl

    <node pkg="turtlebot_control" type="basic_controller" n
        <remap from="cmd_vel" to="turtle1/cmd_vel"/>
        <param name="linear_speed" value="5.0"/>
        <param name="angular_speed" value="5.0"/>
    </node>

    <node pkg="turtlebot_control" type="basic_controller" n
        <!-- TODO: remap cmd_vel topic -->
        <!-- TODO: set speed parameters -->
        <!-- TODO: set spawn_turtle_name param -->
    </node>

</launch>
```

Checkpoint: Creating a New Launch File

- Launch your new launch file.
- rosnode list should show two controller nodes and one turtlesim node.
- rostopic list should show your new remapped topics.
- How many turtles are on the screen? Why?

```
<launch>

    <node pkg="turtlesim" type="turtlesim_node" name="turtl

    <node pkg="turtlebot_control" type="basic_controller" n
        <remap from="cmd_vel" to="turtle1/cmd_vel"/>
        <param name="linear_speed" value="5.0"/>
        <param name="angular_speed" value="5.0"/>
    </node>

    <node pkg="turtlebot_control" type="basic_controller" n
        <!-- TODO: remap cmd_vel topic -->
        <!-- TODO: set speed parameters -->
        <!-- TODO: set spawn_turtle_name param -->
    </node>

</launch>
```

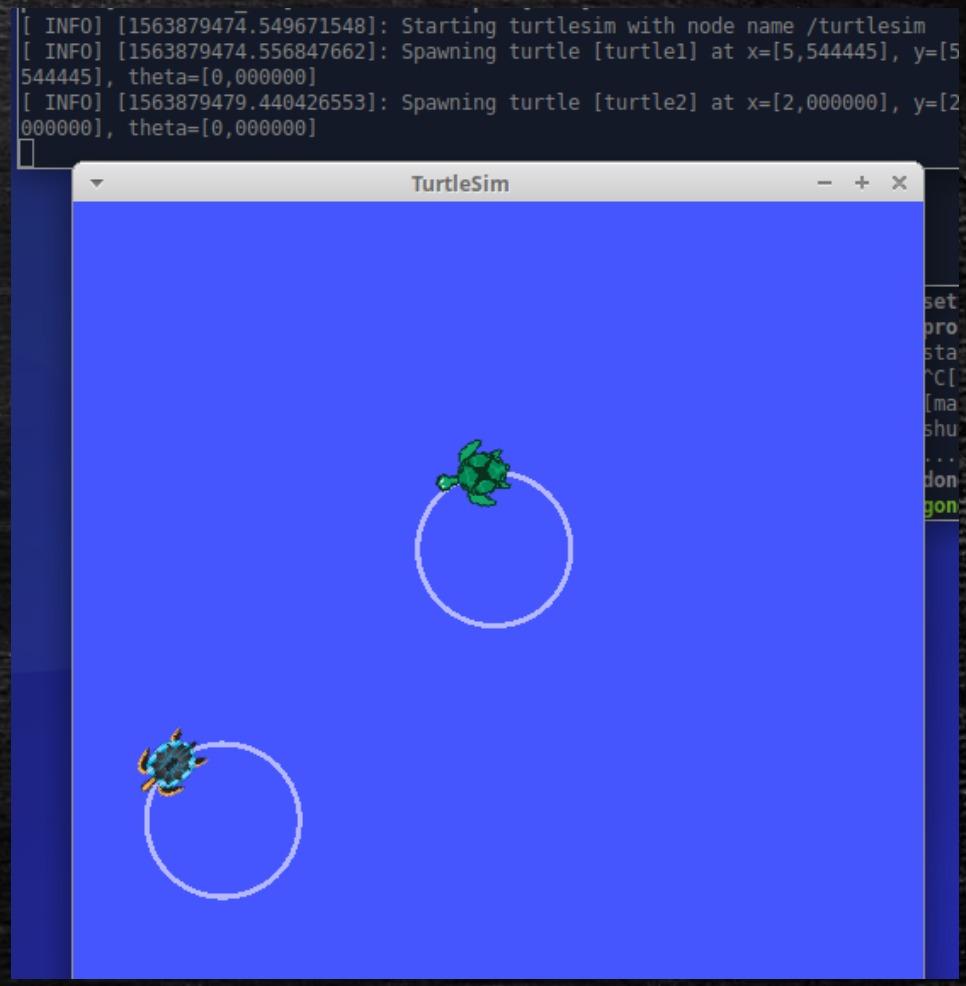
Spawn more turtles

- Create new cpp file:
  - A private method **spawn_turtle** that creates turtles by calling the respective service on turtlesim.
  - New parameter;
  - If that value exists, we need to create a new turtle (waiting a bit);
- Call that node from launch file with parameters
- Time to code!

```
// Create a service client
ros::ServiceClient client = n.serviceClient<turtlesim::Spawn>("spawn");
```

Checkpoint: Dual Turtles

- Launch the new launch file.
- After a few seconds you should see a second turtle pop up and spin around as well.
- Try messing around with the parameters:
  - Speeds for different turtles;
  - Name of the second turtle;
  - What happens?



Improvements

Turtle improvements ideas

- Spawn more that one turtle (using for loop);
- Subscribe to the color topic on the turtles and do something with it:
  - change movement direction;
  - set speed based on the color;
- Restrict movement to certain coordinates;
- Navigate towards a goal.

Conclusion

What did we do today? What now?

- In the previous session we covered basic ROS concepts;
- Today we focused on honing those skills through a number of extra exercises;
- We were very good developers and kept everything source-controlled using git;
- Next week we will focus on using a more accurate simulator and will have an actual challenge for you to solve.

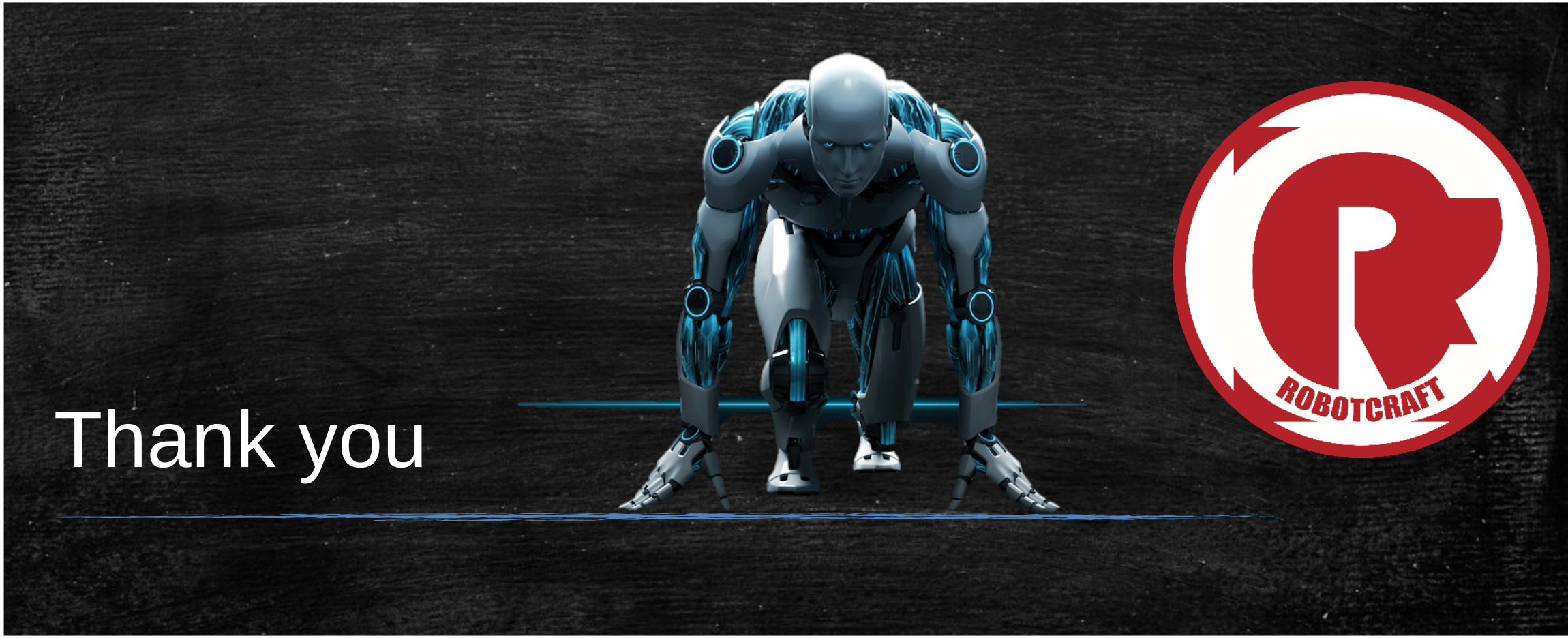
Additional Resources

- The ROS tutorials are excellent and cover most of the material:
  - http://wiki.ros.org/ROS/Tutorials
- Don’t forget about the repository with the barebones code:
  - https://github.com/gondsm/turtlebot_control_blanks
- And also my own solutions, in case you need reference:
  - https://github.com/gondsm/robotcraft_turtlebot_control

ROBOTCRAFT©

ROSin

INGENIARIUS



Panagiotis Karfakis
Robotics Researcher
panagiotis@ingeniarius.pt

Craft #6

01/08/2022