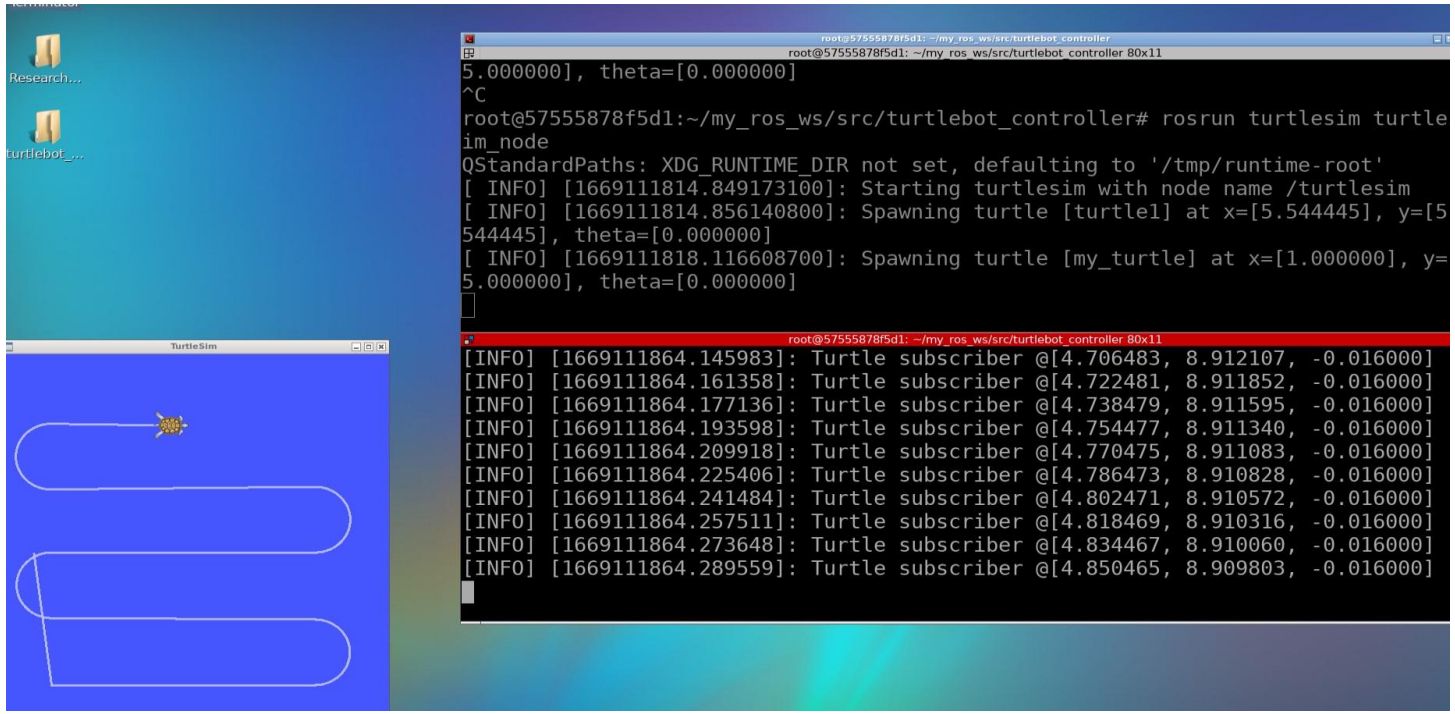
Custom Messages and Services

Carmine Tommaso Recchiuto

Exercise proposed

Here you can find solutions (in python and cpp) for the exercises proposed last week: https://github.com/CarmineD8/turtlebot_controller)



Defining Custom Messages

Msg: msg files are simple text files that describe the structure of a ROS message.

Msg files are stored in the msg directory of a package.

- A message can be composed of
  - int8, int16, int32, int64 (or uint)
  - float 32, float 64
  - string
  - Time, duration
  - Other msg files (i.e. geometry_msgs package)
  - Variable-length array and fixed-length array

Defining Custom Messages

While we can use messages that are already defined in some packages, we can also create some custom messages in our packages. This may be useful also for increasing the modularity and integration capabilities of our module. These are the steps to be followed:

- Create a folder named ***msg*** in your package
- Create a **.msg** file with the definition of the message structure:

  e.g. string first_name

  string last_name

  uint8 age

  uint32 score

Defining Custom Messages

For this example, let's keep working with the package turtlebot_controller

Inside the folder msg, we define a new .msg file (Vel.msg), with the following structure:

string name
float32 vel

Defining Custom Messages

- Modify the CMakeLists.txt in your package:

Add ***message_generation*** to the list of components

i.e.: *find_package (catkin REQUIRED COMPONENTS*
roscpp
std_msgs
message_generation*)*

Also, you need to modify the **package.xml** file as seen in the previous week for the geometry_msgs package, by adding message_generation among the dependencies.

Defining Custom Messages

- Again concerning the CMakeLists.txt file, you will need to uncomment and modify the following lines:

```
# add_message_files(
#   FILES
#   Message1.msg    ➔ here the actual name of the message #
    Message2.msg
# )
```

and

```
# generate_messages( #
    DEPENDENCIES
#   std_msgs     ➔ here the other packages used in the message
# )
```

Defining Custom Messages

If you are going to use custom messages:

- Add the header (es. #include *“turtlebot_controller/Vel.h”)* in the cpp source code, or the import (from turtlebot_controller.msg import Vel) in python.

If the code is in cpp, add the following lines in the CMakeLists.txt file:

add_dependencies(<node_name> ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

- Also add the related dependency (CMakeLists.txt and package.xml) if the messages have been built in a different package.

In the example (turtlebot_controller), we may try do define a second publisher using the custom message just defined, which publishes the string «linear» and the actual linear velocity.

Defining Custom Services

- As well as messages, service files are simple text files
- Services are composed by two parts:
  - **Request**
  - **Response**
- Example:

```
string first_name
string last_name
---
uint32 last_score
```

Defining Custom Services

- Services files should be defined in the **srv** directories of the package
- Cmakelists.txt and package.xml should be eventually modified by adding the message_generation dependency
- Let's create a new package, my_srv, with the dependencies **message_generation**, roscpp and *rospy*
- Here, we define a custom *service message (Velocity.srv)* with the following structure:

float32 min
float32 max
---
float32 x
float32 z

Defining Custom Services

- Also for custom services, you will need to uncomment the CMakeLists.txt

```
# add_service_files(
#   FILES
#   Service1.srv    ➔ here the actual name of the service
#   Service2.srv
# )
```

and

```
# generate_messages(
#   DEPENDENCIES
#   std_msgs    ➔ here the other packages used in the service
# )
```

Writing a Service Node

- We may be interested now in creating our own service node.

```
#include "ros/ros.h"
#include "my_srv/Velocity.h"
```

- Definition of the service: name of the service and callback
  - *Ros::ServiceServer service=n.advertiseService("/velocity", my_random);*

Example: service node that sends two random floats between a maximum and a minimum value

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "velocity_server");
  ros::NodeHandle n;
  ros::ServiceServer service= n.advertiseService("/velocity", my_random);
  ros::spin();
  return 0;
}
```

Writing a Service Node

The service callback will be something like this:

```
bool my_random(my_srv::Velocity::Request &req,
my_srv::Velocity::Response &res){
        res.x = req.min + (rand()/(RAND_MAX/(req.max-req.min)));
        res.z = req.min + (rand()/(RAND_MAX/(req.max-req.min)));
        return true;
}
```

Also remind to add the needed headers:

```
#include <stdlib.h>
#include <stdio.h>
```

The same thing, but in python

```
#!/usr/bin/env python3

import rospy
import random
from my_srv.srv import Velocity, VelocityResponse

def set_vel(req):
        return VelocityResponse(random.uniform(req.min, req.max),random.uniform(req.min,req.max))

def my_vel_server():
        rospy.init_node('my_vel_server')
        s = rospy.Service('velocity', Velocity, set_vel)
        print("Service ready.")
        rospy.spin()

if __name__=="__main__":
        my_vel_server()
```

Example: Turtlebot controller

```
#include "my_srv/Velocity.h"
```

In the main function, you should add

```
Ros::ServiceClient client2 =
nh.serviceClient<my_srv::Velocity>("/velocity");
```

```
ros::Rate rate(1);
my_srv::Velocity server_vel;
while (ros::ok()){
        server_vel.request.min = 0.0;
        server_vel.request.max = 5.0;
        client2.call(server_vel);
        geometry_msgs::Twist my_vel;
        my_vel.linear.x = server_vel.response.x;
        my_vel.angular.z = server_vel.response.z;
        pub.publish(my_vel);
        turtlebot_controller::Vel my_new_vel;
        my_new_vel.name = "linear";
        my_new_vel.vel = my_vel.linear.x;
        pub2.publish(my_new_vel);
        ros::spinOnce();
        rate.sleep();
        }
```

General Remarks

Using custom messages and services, remember to:

- add the necessary headers in nodes using custom messages and services
- in the CMakeLists.txt and package.xml of packages contaning nodes using custom messages and/or services, add the dependency from the package where custom messages are defined *(es. turtlebot controller)*
- In the CmakeLists.txt of packages contaning nodes using custom messages and/or services, add *(es. turtlebot controller)*:

  add_dependencies(<node_name> ${catkin_EXPORTED_TARGETS})

- If a node using custom messages and/or services is built in the same package where custom messages and services are defined, in the CMakeLists.txt of the package add *(es. my_srv):*

  add_dependencies(<node_name> ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

General Remarks

As a general remark, before submitting your code, please be sure that the packages will be built without errors.

How can you do that?

Practical hint: - remove the directory build and devel of your ros workspace.

- build again the workspace (catkin_make)

If it works without any error, everything is fine. If not, something should be fixed in the CMakeLists.txt of your packages.

Exercise

In a package, create a custom srv, which takes a float as request (x position), and replies with a float (x velocity)

In the same package, create a Service which set a velocity depending from the position (from 2.0 to 9.0), with an armonic oscillator behaviour -> vel = 0.1 + 2*sin (pi*x/7 -2*pi/7) –

In the turtlebot controller of the exercise2, let the new turtle start from x=2.0, y=1.0, and control the velocity of the robot by setting the velocity computed by the service when x is between 2.0 and 9.0