Solutions for Services Quiz

Solution for Services Quiz

- Upgrade the Python file called **bb8_move_custom_service_server.py**. Modify the code you used in **Exercise 3.3**, which contained a Service Server that accepted a custom Service message to activate the circle movement (with a defined duration). This new service will be called **/move_bb8_in_square_custom**. This new service will have to use service messages of the **BB8CustomServiceMessage** type, which is defined here:

The first thing to do is create the BB8CustomServiceMessage.srv, creating a **srv** folder inside the package **services_quiz**.

Service Message: BB8CustomServiceMessage.srv

In [ ]:

```
float64 side       # The distance of each side of the square
int32 repetitions    # The number of times BB-8 has to execute the square moveme
---
bool success         # Did it achieve it?
```

END **Service Message: BB8CustomServiceMessage.srv**

We have to also modify the CMakelists.txt and the package.xml

CMakelists.txt

In [0]:

```
cmake_minimum_required(VERSION 2.8.3)
project(services_quiz)


## Here go all the packages needed to COMPILE the messages of topic, services ar
## It's only getting its paths, and not really importing them to be used in the
## It's only for further functions in CMakeLists.txt to be able to find those pa
## In package.xml you have to state them as build
find_package(catkin REQUIRED COMPONENTS
  std_msgs
  message_generation
)

## Generate services in the 'srv' folder
## In this function will be all the action messages of this package ( in the act
## You can state that it gets all the actions inside the action directory: DIREC
## Or just the action messages stated explicitly: FILES my_custom_action.action
## In your case, you only need to do one of two things, as you wish.
add_service_files(
  FILES
  BB8CustomServiceMessage.srv
)

## Here is where the packages needed for the action messages compilation are imp
generate_messages(
  DEPENDENCIES
  std_msgs
)

## State here all the packages that will be needed by someone that executes some
## All the packages stated here must be in the package.xml as exec_depend
catkin_package(
  CATKIN_DEPENDS rospy
)


include_directories(
  ${catkin_INCLUDE_DIRS}
)
```

package.xml

In [ ]:

```
<?xml version="1.0"?>
<package format="2">
  <name>services_quiz</name>
  <version>0.0.0</version>
  <description>The services_quiz package</description>

  <maintainer email="user@todo.todo">user</maintainer>

  <license>TODO</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <exec_depend>rospy</exec_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>std_msgs</exec_depend>
  <build_export_depend>message_runtime</build_export_depend>
  <exec_depend>message_runtime</exec_depend>

  <export>
  </export>
</package>
```

Once you have it, just compile and source in **ALL the webshells** that you are going to use so that ROS can find the new Messages.

In [ ]:

```
roscd;cd ..
catkin_make
source devel/setup.bash
```

And check that it finds the messages:

In [ ]:

```
rossrv list | grep BB8CustomServiceMessage
```

You should get:

In [ ]:

```
services_quiz/BB8CustomServiceMessage
```

- In the previous exercises, you were triggering a circle movement when calling to your service. In this new service, the movement triggered will have to be a square, like in the image below:

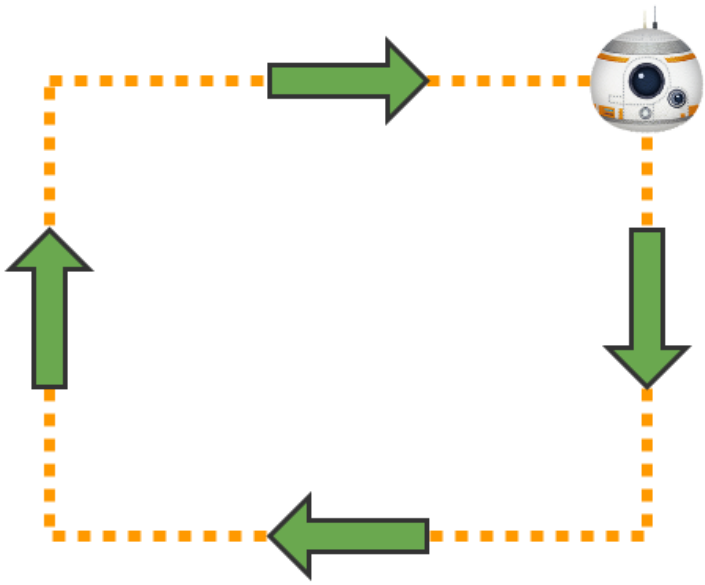


Fig.3.1 - BB-8 Square Movement Diagram

- Use the data passed to this new **/move_bb8_in_square_custom** to change the way BB-8 moves. Depending on the **side** value, the service must move the BB-8 robot in a square movement based on the **side** given.
  Also, the BB-8 must repeat the shape as many times as indicated in the **repetitions** variable of the message.
  Finally, it must return **True** if everything went OK in the **success** variable.

Now its time to create the bb8_move_custom_service_server.py, using the new service messages **BB8CustomServiceMessage**.

Python File: bb8_move_custom_service_server.py

In [ ]:

```
#! /usr/bin/env python

import rospy
from std_srvs.srv import Empty, EmptyResponse # you import the service message p
from services_quiz.srv import BB8CustomServiceMessage, BB8CustomServiceMessageRe
from geometry_msgs.msg import Twist
import time

def my_callback(request):

    rospy.loginfo("The Service move_bb8_in_square_custom has been called")

    radius = request.side
    for i in range(request.repetitions):
        rospy.loginfo("Moving forward...")
        move_circle.linear.x = 0.2
        move_circle.angular.z = 0
        my_pub.publish(move_circle)
        time.sleep(radius)
        rospy.loginfo("Rotating...")
        move_circle.linear.x = 0
        move_circle.angular.z = 0.2
        my_pub.publish(move_circle)
        time.sleep(4)
        rospy.loginfo("Moving forward...")
        move_circle.linear.x = 0.2
        move_circle.angular.z = 0
        my_pub.publish(move_circle)
        time.sleep(radius)
        rospy.loginfo("Rotating...")
        move_circle.linear.x = 0
        move_circle.angular.z = 0.2
        my_pub.publish(move_circle)
        time.sleep(4)
        rospy.loginfo("Moving forward...")
        move_circle.linear.x = 0.2
        move_circle.angular.z = 0
        my_pub.publish(move_circle)
        time.sleep(radius)
        rospy.loginfo("Rotating...")
        move_circle.linear.x = 0
        move_circle.angular.z = 0.2
        my_pub.publish(move_circle)
        time.sleep(4)
        rospy.loginfo("Moving forward...")
        move_circle.linear.x = 0.2
        move_circle.angular.z = 0
```

```
        my_pub.publish(move_circle)
        time.sleep(radius)
        rospy.loginfo("Rotating...")
        move_circle.linear.x = 0
        move_circle.angular.z = 0.2
        my_pub.publish(move_circle)
        time.sleep(4)
        rospy.loginfo("Stopping...")
        move_circle.linear.x = 0
        move_circle.angular.z = 0
        my_pub.publish(move_circle)
        time.sleep(2)

    rospy.loginfo("Finished service move_bb8_in_square_custom")
    response = BB8CustomServiceMessageResponse()
    response.success = True
    return response # the service Response class, in this case EmptyResponse

rospy.init_node('service_move_bb8_in_square_server')
my_service = rospy.Service('/move_bb8_in_square_custom', BB8CustomServiceMessage
my_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
move_circle = Twist()
rospy.loginfo("Service /move_bb8_in_square_custom Ready")
rospy.spin() # mantain the service open.
```

END **Python File: bb8_move_custom_service_server.py**

In [ ]:

```
rospy.loginfo("Moving forward...")
move_circle.linear.x = 0.2
move_circle.angular.z = 0
my_pub.publish(move_circle)
time.sleep(radius)
```

Note that we are using the **side** variable of the message as the value for the time that the robot will be moving forward. This means, then, that the bigger this **side** variable is, the bigger the square made by the BB-8 robot will be.

- Create a new launch called **start_bb8_move_custom_service_server.launch** that launches the server that was launched in the python file **bb8_move_custom_service_server.py**.
- Test that when calling this new service **/move_bb8_in_square_custom**, BB8 moves accordingly.

This **start_bb8_move_custom_service_server.launch** is the same as the one made in Exercise 3.3, just changing the package name and node name.

Launch File: start_bb8_move_custom_service_server.launch

In [ ]:

```
<launch>
    <!-- Start Service Server for move_bb8_in_square service -->
    <node pkg="services_quiz" type="bb8_move_custom_service_server.py" name="ser
    </node>
</launch>
```

END **Launch File: start_bb8_move_custom_service_server.launch**

- Create a new service client that calls the service **/move_bb8_in_square_custom** and makes BB-8 move in a small square **twice** and in a big square **once**. It could be called **bb8_move_custom_service_client.py** and the launch that starts it **call_bb8_move_in_square_custom_service_server.launch**.

We first create the bb8_move_custom_service_client.py that will execute a call to perform the two small squares and one big square.

Python File: bb8_move_custom_service_client.py

In [ ]:

```
#! /usr/bin/env python
import rospkg
import rospy
from services_quiz.srv import BB8CustomServiceMessage, BB8CustomServiceMessageRe


rospy.init_node('service_move_bb8_in_square_custom_client') # Initialise a ROS r
rospy.wait_for_service('/move_bb8_in_square_custom') # Wait for the service clie
move_bb8_in_square_service_client = rospy.ServiceProxy('/move_bb8_in_square_cust
move_bb8_in_square_request_object = BB8CustomServiceMessageRequest() # Create ar


"""
# BB8CustomServiceMessage
float64 side       # The distance of each side of the square
int32 repetitions    # The number of times BB-8 has to execute the square moveme
---
bool success         # Did it achieve it?
"""
move_bb8_in_square_request_object.side = 2
move_bb8_in_square_request_object.repetitions = 2

rospy.loginfo("Start Two Small Squares...")
result = move_bb8_in_square_service_client(move_bb8_in_square_request_object) #
rospy.loginfo(str(result)) # Print the result given by the service called

move_bb8_in_square_request_object.side = 4
move_bb8_in_square_request_object.repetitions = 1

rospy.loginfo("Start One Big Square...")
result = move_bb8_in_square_service_client(move_bb8_in_square_request_object) #
rospy.loginfo(str(result))

rospy.loginfo("END of Service call...")
```

END **Python File: bb8_move_custom_service_client.py**

And now we have to create a launch that launches this python node, called call_bb8_move_in_square_custom_service_server.launch:

Launch File: call_bb8_move_in_square_custom_service_server.launch

In [ ]:

```
<launch>
    <!-- Start Service Server for move_bb8_in_square service -->
    <node pkg="services_quiz" type="bb8_move_custom_service_client.py" name="ser
    </node>
</launch>
```

END **Launch File: call_bb8_move_in_square_custom_service_server.launch**

Finally, you just have to test the entire pipeline, so you have to launch the server launch in one webshell and the client in another. It should make the robot move as desired.

Execute in WebShell #1

In [ ]:

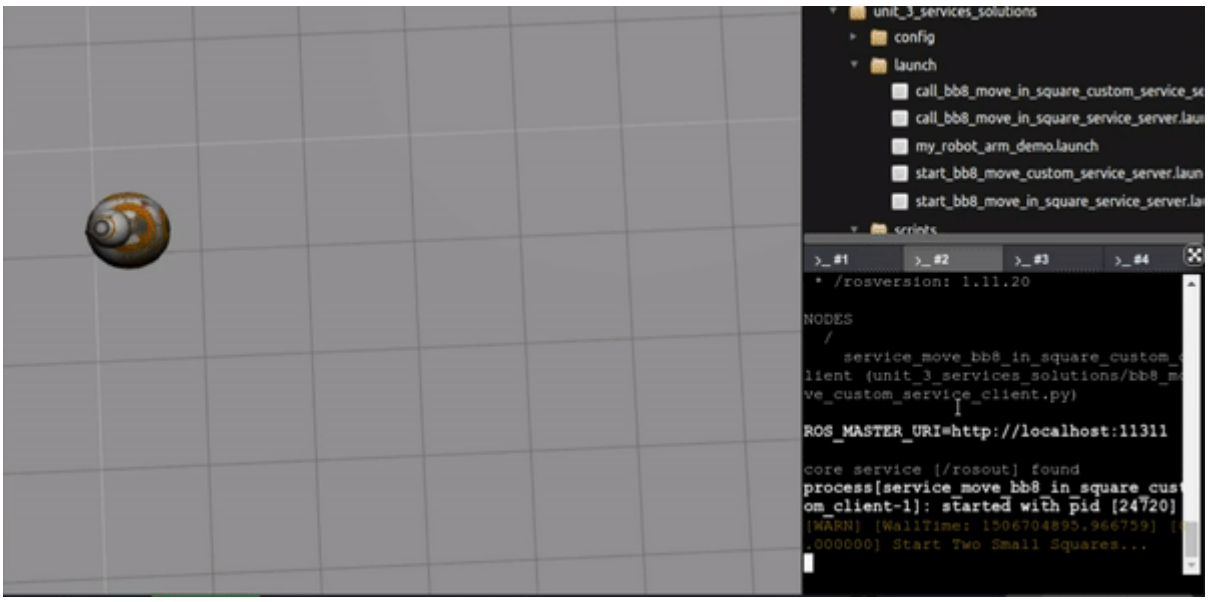
```
roslaunch services_quiz start_bb8_move_custom_service_server.launch
```

Execute in WebShell #2

In [ ]:

```
roslaunch services_quiz call_bb8_move_in_square_custom_service_server.launch
```

You should get something similar to this, but slower, because this has been accelerated for practical reasons:



In [ ]: