

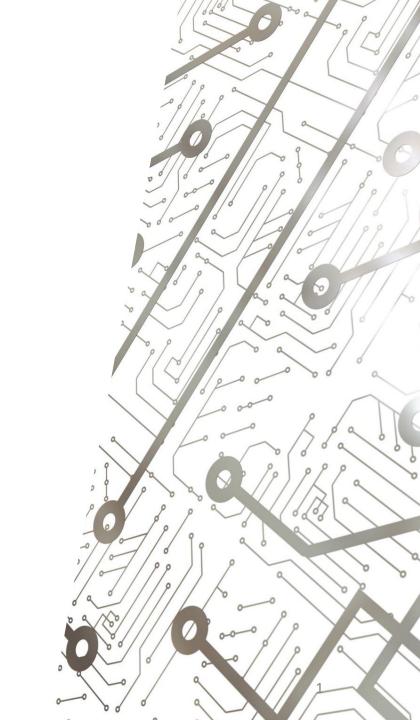




Software Lecture 4:



Jacques Cloete



Contents

In this lecture, we shall cover:

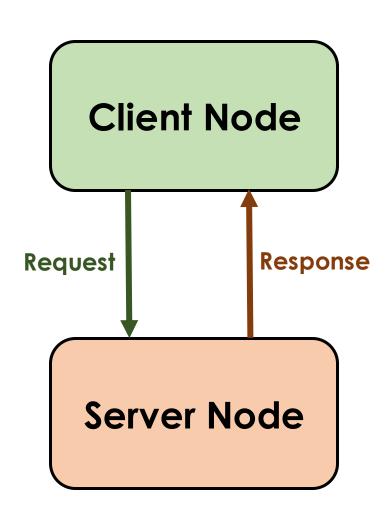
- Overview for ROS Services
- Defining a new ROS Service
- Creating our first ROS Service Client/Server pair

Before We Begin

- Again, I strongly suggest bookmarking the following link: https://github.com/OxRAMSociety/RobotArm
- All example code can be found in:
 Tutorials/Software Tutorials (2022)/Example Scripts
 Of course, code for this lecture will be in the Lecture 4 folder, and so on for future lectures
- I will provide code in these presentations, but refer to the Example Scripts to check your own!

ROS Services – Overview

- Request/response system
- Client node sends a request and awaits a response
- Server node listens to the request, processes it and sends a response back to that client when complete
- The request and response are both ROS messages



Creating a Service Client/Server Pair

 We will revisit our Square Number Factory idea, but now implement it using a ROS Service!

- ROS services must be defined in a unique 'srv' file
- Navigate to tutorial_scripts and create a folder named srv
 - This folder will contain all our custom ROS service definitions
- Inside that folder, create a file ExampleService.srv and open it

- We first define all data contained within the request message
 - In this case, we have defined a 32-bit integer referred to as 'input number'
 - We can have multiple entries if we want!
- We then define all data contained within the response message
- We separate these two groups with 3 hyphens (i.e. ---)

```
1  # Request
2  int32 input_number
3  ---
4  # Response
5  int32 output_number
```

Reproduce this!

- For the package to recognise our new ROS service, we must make sure they are generated when the package is built
- Navigate to tutorial_scripts and open the CMakeLists.txt file
- Go to line ~10 and uncomment/edit the find_package section as follows:

```
find_package(catkin REQUIRED COMPONENTS
rospy
std_msgs
message_generation
)
```

 Go to line ~60 and uncomment/edit the add_service_files section as follows:

```
57 ## Generate services in the 'srv' folder
58 add_service_files(
59 FILES
60 ExampleService.srv
61 )
```

 Go to line ~70 and uncomment/edit the generate_messages section as follows:

```
## Generate added messages and services with any dependencies listed here
generate_messages(

DEPENDENCIES
std_msgs
)
```

 Go to line ~100 and uncomment/edit the catkin_package section as follows:

```
103    catkin_package(
104     # INCLUDE_DIRS include
105     LIBRARIES tutorial_scripts
106     CATKIN_DEPENDS rospy std_msgs
107     # DEPENDS system_lib
108     DEPENDS message_runtime
109 )
```

 Now open the package.xml file, go to line ~58 and add the following:

Finally, save all files and rebuild the workspace

Creating a Server Node

 In tutorial_scripts' scripts folder, create the file Example_Service_Server.py:

```
#! /usr/bin/env python3
    import rospy
    from tutorial scripts.srv import ExampleService, ExampleServiceResponse # Import message types for our Example ROS service
    # Call requests are passed into this function
    def handleRequest(req):
        x = req.input number # Extract number to be squared
        rospy.loginfo("Order Received: %d",x)
        x \text{ squared} = x*x
                                # Square the number
        rospy.loginfo("Processed Order: %d \n ---",x squared)
        return ExampleServiceResponse(x squared) # Return square number to client
    if name == ' main ':
16
        rospy.init node('ExampleServiceServerNode')
        # Declare a new service called 'ExampleService', of type ExampleService, using function handleRequest to handle requests
        srv = rospy.Service('ExampleService', ExampleService, handleRequest)
        rospy.spin()
```

Creating a Server Node

- The Server node:
 - Defines function 'handleRequest', which shall be used to process incoming requests

```
# Call requests are passed into this function
def handleRequest(req):
    x = req.input_number  # Extract number to be squared
    rospy.loginfo("Order Received: %d",x)
    x_squared = x*x  # Square the number
    rospy.loginfo("Processed Order: %d \n ---",x_squared)
    return ExampleServiceResponse(x_squared)  # Return square number to client
```

 Sets up a service of type ExampleService (which we shall also name 'ExampleService') and starts listening for requests

Declare a new service called 'ExampleService', of type ExampleService, using function handleRequest to handle requests srv = rospy.Service('ExampleService', ExampleService, handleRequest)

Creating a Client Node

 In tutorial_scripts' scripts folder, create the file Example_Service_Client.py:

```
#! /usr/bin/env python3
     import rospy
     import random
     from tutorial scripts.srv import ExampleService # Import message types for our Example ROS service
     rospy.init node('ExampleServiceClientNode')
     r = rospy.Rate(2) # 2 Hz
     while not rospy.is shutdown():
13
         order = random.randint(1,10)
                                       # Randomly generate a new order
14
         rospy.loginfo("Number to be Squared: %d",order)
         rospy.wait for service('ExampleService')  # Wait for 'ExampleService' to become available
         try:
             request = rospy.ServiceProxy('ExampleService', ExampleService) # Create a handle by which to call the service
20
21
             receiver = request(order) # Call the service, using our randomly-generated order as an input
             rospy.loginfo("Square Number Received: %d \n ---", receiver.output number)
22
23
24
         except rospy. Service Exception as e: # An exception will be raised if the call fails
             rospy.logerr("Service call failed: %s \n ---",e)
         r.sleep()
```

Creating a Client Node

- The Client node:
 - Waits for 'ExampleService' to become available
 - Sets up a handle by which to call the service
 - Calls the service and waits for a response (stored in variable 'receiver')

```
rospy.wait_for_service('ExampleService')  # Wait for 'ExampleService' to become available

try:
    request = rospy.ServiceProxy('ExampleService', ExampleService)  # Create a handle by which to call the service
    receiver = request(order)  # Call the service, using our randomly-generated order as an input
    rospy.loginfo("Square Number Received: %d \n ---",receiver.output_number)

except rospy.ServiceException as e: # An exception will be raised if the call fails
    rospy.logerr("Service call failed: %s \n ---",e)
```

Testing Our Code

- Remember to give permissions to our code
- Now, in three separate Terminals, run the following:
 - 1. roscore
 - 2. rosrun tutorial_scripts Example_Service_Server.py
 - 3. rosrun tutorial_scripts Example_Service_Client.py

Testing Our Program

You should see something like this:

```
[INFO] [1664557330.639855]: Number to be Squared: 10
[INFO] [1664557330.647969]: Square Number Received: 100
---
[INFO] [1664557331.141131]: Number to be Squared: 3
[INFO] [1664557331.154928]: Square Number Received: 9
---
[INFO] [1664557331.640794]: Number to be Squared: 8
[INFO] [1664557331.653462]: Square Number Received: 64
---
[INFO] [1664557332.141166]: Number to be Squared: 1
[INFO] [1664557332.151815]: Square Number Received: 1
---
[INFO] [1664557332.640750]: Number to be Squared: 7
[INFO] [1664557332.652888]: Square Number Received: 49
---
[INFO] [1664557333.140607]: Number to be Squared: 9
[INFO] [1664557333.159194]: Square Number Received: 81
```

```
[INFO] [1664557330.645123]: Order Received: 10
[INFO] [1664557330.646329]: Processed Order: 100
---
[INFO] [1664557331.150879]: Order Received: 3
[INFO] [1664557331.153457]: Processed Order: 9
---
[INFO] [1664557331.649865]: Order Received: 8
[INFO] [1664557331.651667]: Processed Order: 64
---
[INFO] [1664557332.149097]: Order Received: 1
[INFO] [1664557332.150807]: Processed Order: 1
---
[INFO] [1664557332.649763]: Order Received: 7
[INFO] [1664557332.651352]: Processed Order: 49
---
[INFO] [1664557333.154606]: Order Received: 9
[INFO] [1664557333.157303]: Processed Order: 81
```

Client

Server

- The client node randomly generates a number to be squared and sends it to the server node
- The server node squares the number then sends it back

Closing Thoughts

- A much better solution for our Square Number Factory!
- ROS Services are great for when the requested process is fast
 - e.g. retrieving information about a robot's pose, or spawning an object in a simulation
- What about if the process will take an extended period of time?
 - e.g. commanding a robot arm to move to a specific pose
- For those cases, it is better to use ROS Actions
 - More on these next time!

Summary

We covered:

- Overview for ROS Services
- Defining a new ROS Service
- Creating our first ROS Service Client/Server pair

Homework: Revise ROS Services (p.4), go over the code and make sure you can follow what's going on

Next time, we will learn about ROS Actions!







Thank You!

Any Questions? Contact jacques.cloete@stx.ox.ac.uk