CENG483: Behavioural Robotics
Izmir Institute of Technology
Fall 2023
Instructor: Dr. Berat Alper Erol

Homework Set: 03

Ozan Gülbaş

This homework answers the problem set sequentially.

1. *What is ROS publisher and subscriber? Explain how they are used.*

   In ROS a publisher/subscriber system consists of two elements, the producers of data (publishers) and the consumers of data (subscribers). Publishers and subscribers know how to communicate with each other through topics, which is a known name so that the entities can find each other. Publishers and subscribers on the same topic can directly communicate with each other. When data is published on the topic by any of the publishers, all subscribers in the system will receive the data.

2. *What are the differences between ROS service and messages?*

   In ROS, a service denotes a remote procedure call which means, a node can make a remote procedure call to another node which will do a computation and return a result. Services have two parts, a server service and a client service. A service server is an entity that accepts a remote procedure request and performs some computation on it whereas, a service client is an entity that requests a remote service server to perform a computation on its behalf.

   Messages are a way for a ROS node to send data on the network to other ROS nodes, with no response expected. The main difference between service and messages is that service always exists with a server and client, sending data and responding to each other. However, a message is used when the data to be sent expects no response.

3. *What is ROS node and topic? Explain how they are used.*

   Nodes can be thought of as a collection of systems dedicated to one specific purpose in ROS. Nodes can publish to named topics to deliver data to other nodes or subscribe to named topics to get data from other nodes. They can also act as a service client to have another node perform a computation on their behalf, or as a service server to provide functionality to other nodes.

   Topics can be thought of as a middleman for data communication in a ROS publisher/subscriber system. Topics are a vital element of the ROS that acts as a medium (bus) for nodes to exchange messages. A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.

4. *How to create a ROS Service file? Explain.*

   Assuming we already have a workspace and, for this question we will call it as $ros2\_ws$, and created packages in $ros2\_ws/src$ with the following command:

```
ros2 pkg create --build-type ament_python py_srvcli --dependencies
    rclpy example_interfaces
```

   First, we need to fill the **maintainer**, **maintainer_email**, **description** and **license** fields according to the automatically created*package.xml* file.

```
<description>Python client server tutorial</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

   Fill the *setup.py* according to the *package.xml* file.

```
maintainer='YourName',
maintainer_email='you@email.com',
description='Examples of minimal publisher/subscriber using rclpy',
license='Apache License 2.0',
```

   Next, we need to write the service node. Inside the $ros2\_ws/src/py\_srvcli/py\_srvcli$ directory, create a new file called $service\_member\_function.py$. The first import statement imports the **AddTwoInts** service type from the **example_interfaces** package. The following import statement imports the ROS 2 Python client library, specifically the Node class.

```
from example_interfaces.srv import AddTwoInts

import rclpy
from rclpy.node import Node
```

   The **MinimalService** class constructor initializes the node with the name **minimal_service**. Then, it creates a service and defines the type, name, and callback.

```
def __init__(self):
    super().__init__('minimal_service')
    self.srv = self.create_service(
        AddTwoInts,
        'add_two_ints',
        self.add_two_ints_callback)
```

   The definition of the service callback receives the request data, sums it, and returns the sum as a response.

```
def add_two_ints_callback(self, request, response):
    response.sum = request.a + request.b
    self.get_logger()
        .info('Incoming request\na: %d b: %d' % (request.a, request.b))

    return response
```

Finally, the main class initializes the ROS 2 Python client library, instantiates the MinimalService class to create the service node and spins the node to handle callbacks.

After this, we need to add an entry point to our *setup.py* file. Add the following line between the **console_scripts**: brackets:

```
'service = py_srvcli.service_member_function:main',
```

Next, we need to write the client node. As with the service code, we first import the necessary libraries.

```
import sys

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node
```

The MinimalClientAsync class constructor initializes the node with the name **minimal_client_async**. The constructor definition creates a client with the same type and name as the service node. The type and name must match the client and service to be able to communicate. The while loop in the constructor checks if a service matching the type and name of the client is available once a second. Finally, it creates a new AddTwoInts request object.

```
def __init__(self):
    super().__init__('minimal_client_async')
    self.cli = self.create_client(AddTwoInts, 'add_two_ints')
    while not self.cli.wait_for_service(timeout_sec=1.0):
        self.get_logger()
            .info('service not available, waiting again...')
    self.req = AddTwoInts.Request()
```

Below the constructor is the **send_request** method, which will send the request and spin until it receives the response or fails.

```
def send_request(self, a, b):
    self.req.a = a
    self.req.b = b
    self.future = self.cli.call_async(self.req)
    rclpy.spin_until_future_complete(self, self.future)
    return self.future.result()
```

Finally, we have the main method, which constructs a **MinimalClientAsync** object, sends the request using the passed-in command-line arguments, and logs the results.

```
def main():
    rclpy.init()

    minimal_client = MinimalClientAsync()
    response = minimal_client.send_request(
        int(sys.argv[1]),
        int(sys.argv[2]))
    minimal_client.get_logger().info(
        'Result of add_two_ints: for %d + %d = %d' %
        (int(sys.argv[1]), int(sys.argv[2]), response.sum))

    minimal_client.destroy_node()
    rclpy.shutdown()
```

We add the entry point to our *setup.py* file as well:

```
entry_points={
    'console_scripts': [
        'service = py_srvcli.service_member_function:main',
        'client = py_srvcli.client_member_function:main',
    ],
},
```

Since we have written our client and server service we can build and run with the commands:

```
colcon build --packages-select py_srvcli
ros2 run py_srvcli service
```

Open another terminal and source the setup files from inside *ros*2_*ws* again. Start the client node, followed by any two integers separated by a space:

```
ros2 run py_srvcli client 2 3
```

The client would receive a response like this:

```
[INFO] [minimal_client_async]: Result of add_two_ints: for 2 + 3 = 5
```

Return to the terminal where your service node is running. We will see that it published log messages when it received the request:

```
[INFO] [minimal_service]: Incoming request
a: 2 b: 3
```

5. *How to use ROS publishers and subscribers with Pyhton?*

   Assuming we already have a workspace and, for this question we will call it as $ros2\_ws$, and created packages in $ros2\_ws/src$ with the following command:

   ```
   ros2 pkg create --build-type ament_python py_pubsub
   ```

   and have pre-written publisher and subscriber files such as $publisher\_member\_function.py$ and $subscriber\_member\_function.py$.

   - For the publisher part, first we need to add dependencies. Navigate to directory $ros2\_ws/src/py\_pubsub$, where the *setup.py*, $setup.cfg$, and *package.xml* files that are created with the **ros2 pkg create** command. We need to add **rclpy** and **std_msgs** packages to *package.xml* file. Also, make sure we fill <**description**>, <**maintainer**>, and <**license**> tags in *package.xml* file. Add the dependencies like this:

     ```
     <exec_depend>rclpy</exec_depend>
     <exec_depend>std_msgs</exec_depend>
     ```

     This declares the package needs **rclpy** and **std_msgs** when its code is executed. Next, we need to add an entry point to our **setup.py**. Fill the **maintainer**, **maintainer_email**, **description** and **license** fields according to previously filled *package.xml* file.

     ```
     maintainer='YourName',
     maintainer_email='you@email.com',
     description='Examples of minimal publisher/subscriber using rclpy',
     license='Apache License 2.0',
     ```

     Add the following line within the **console_scripts** brackets of the **entry_points** field:

     ```
     entry_points={
         'console_scripts': [
                 'talker = py_pubsub.publisher_member_function:main',
         ],
     },
     ```

     Finally, we need to check setup.cfg to make sure that the contents of it correctly populated like:

     ```
     [develop]
     script_dir=$base/lib/py_pubsub
     [install]
     install_scripts=$base/lib/py_pubsub
     ```

- For the subscriber part there is nothing new to add to previously edited *package.xml* and *setup.cfg* files. We only need to add an entry point to the *setup.py* like:

```
entry_points={
    'console_scripts': [
            'talker = py_pubsub.publisher_member_function:main',
            'listener = py_pubsub.subscriber_member_function:main',
    ],
},
```

After building with **colcon** we run the talker node with:

```
ros2 run py_pubsub talker
```

The output on the terminal should be:

```
[INFO] [minimal_publisher]: Publishing: "Hello World: 0"
[INFO] [minimal_publisher]: Publishing: "Hello World: 1"
[INFO] [minimal_publisher]: Publishing: "Hello World: 2"
[INFO] [minimal_publisher]: Publishing: "Hello World: 3"
[INFO] [minimal_publisher]: Publishing: "Hello World: 4"
...
```

In another terminal we start the listener node with:

```
ros2 run py_pubsub listener
```

The output on the terminal should be:

```
[INFO] [minimal_subscriber]: I heard: "Hello World: 10"
[INFO] [minimal_subscriber]: I heard: "Hello World: 11"
[INFO] [minimal_subscriber]: I heard: "Hello World: 12"
[INFO] [minimal_subscriber]: I heard: "Hello World: 13"
[INFO] [minimal_subscriber]: I heard: "Hello World: 14"
...
```

6. *Which node should be published to give a robot speed command in Python?*

   To give a robot speed command, /**cmd_vel** node is used. As it was in the EvaSec example, we can create a publisher for /**cmd_vel** node to publish necessary speed message like:

```
self.robot_speed_pub = rospy.Publisher(
    '/cmd_vel',
    Twist,
    que_size=10
)

robot_speed_msg = Twist()

robot_speed_msg.linear.x = 0.0
robot_speed_msg.angular.z = angular_speed

self.robot_speed_pub.publish(robot_speed_msg)
```

1 References

Ros 2 documentation. ROS 2 Documentation - ROS 2 Documentation: Iron documentation. (n.d.). https://docs.ros.org/en/iron/index.html
Yayan, U., & Erdoğmuş, A. K. (2021). ROS ile Robotik Uygulamalar (1st ed., Vol. 1). Nobel Akademi Yayıncılık.