Module 3 - Python3 for Robotics

Lesson Objectives:

1. Use Object Oriented Programming (OOP) to develop advanced chat client

Agenda:

1. In Class Exercise.

1 In Class Exercise.

1. On the master, create a package called `ice_python`.

```
cd ~/master_ws/src/ece495_fall2021-USERNAME/master/
catkin_create_pkg ice_python rospy std_msgs
cd ~/master_ws
catkin_make
source ~/.bashrc
```

2. Create two chat files

```
cd ~/master_ws/src/ece495_fall2021-USERNAME/master/ice_python/src
touch chat_client.py
touch chat_server.py
chmod +x *.py # makes all python files executable
```

3. Edit the chat_client.py (double click in file browser)

4. Add necessary include statements

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
```

5. Create a Client class with a constant class dictionary that is used to map numbers to messages. For example:

```
class Client:
    MESSAGE = {1: "Hello!", 2: "How are you?", 3: "Where are you from?",
        4: What are you doing today?"}
```

6. Initialize a class variable to store the message and the publisher to publish the message to the "client" topic.

```
def __init__(self):
    self.message = String()
    self.pub = rospy.Publisher('client', String, queue_size=10)
```

7. Initialize the subscriber that will receive String messages over the "server" topic. When the message is received, it should call the callback function "callback_received".

```
rospy.Subscriber('server', String, self.callback_received)
```

8. Initialize a timer that calls a function "callback_input" every second.

```
rospy.Timer(rospy.Duration(1), self.callback_input)
```

9. Create the "callback_input" class function that has two parameters: the class and an TimerEvent object. The TimerEvent object provides timing information for the callback. This website has more information on the ROS Timer `http://wiki.ros.org/rospy/Overview/Time#Timer`

   (a) This function should tell the user to input a number that corresponds to a message.

(b) The function should check the user input to ensure that it is a valid number and if not, then continue to ask the user for input

```
def callback_input(self, event):
    print("Using the number keys, input a value that corresponds to one
        of the messages:", self.MESSAGE)
    valid = False
    while not valid:
        chat_str = input()
        try:
           val = int(chat_str)
           if(0 < val < 5):
               self.message = self.MESSAGE[val]
               self.pub.publish(self.message)
               valid = True
           else:
               print("This is not a valid input, please input a number
                   that corresponds to the messages:", self.MESSAGE)
       except ValueError:
           print("This is not a valid input, please input a number that
               corresponds to the messages:", self.MESSAGE)
```

10. Create the "callback_received" class function that is called when the client receives a response from the server, this function will have two parameters: the class and the data received. The function should print both the message sent and the message received.

```
def callback_received(self, data):
    print("Client sent", self.message)
    print("Server responded", data.data)
```

11. Create the main function that initializes the client node and class and runs forever.

```
if __name__ == "__main__":
    rospy.init_node('client', anonymous = True)

    c = Client()
    rospy.spin()
```

12. At this point your client should be complete. Note that you have both a publisher and subscriber within one node. Using a Class allows us to facilitate this passing of information among functions.

13. Answer the following questions:

    (a) Does anyone notice an issue with where we are requesting user input?

    (b) Could there be any timing issues with our request for user input and the response from the server?

    (c) What may be a better method to implement a chat client like this other than subscriber/publisher?

14. Edit the chat_server.py file (double click in file browser)
15. Add necessary include statements

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
```

16. Create a Server class with a constant dictionary used to map messages to responses. For example:

```
class Server:
    MESSAGE = {"Hello!": "Hi there!",
        "How are you?": "I am great, thanks for asking!",
        "Where are you from?": "I am from Las Vegas.",
        "What are you doing today?": "ECE495!"}


    def __init__(self):
```

17. Initialize a class variable to store the message and the publisher to publish the response to the "server" topic.
18. Initialize the subscriber that will receive String messages over the "client" topic. When the message is received, it should call the callback function "callback_received".
19. Create the "callback_received" class function that is called when the server receives a message from the client; this function will have two parameters: the class and the data received. The function should print the message from the client, the appropriate response from the dictionary, and then publish the response.
20. Create the main function that initializes the server node and class and runs forever.
21. At this point your server should be complete.
22. Open a new terminal and run roscore.
23. Open a new terminal and run the "chat_client.py" node.
24. Open a new terminal and run the "chat_server.py" node.
25. Test the operation of your chat bot. Ensure that it still works with invalid input.

Checkpoint. Take a screenshot or show the instructor the following:

1. List of running topics.
2. The *rqt_graph* for the nodes and topics currently running.
3. An echo of messages sent via the chat topic.
4. Show what type of message is sent over the chat topic (Hint: use the rostopic command).

2 Assignments.

☐ Finish Unit 5 and EXAM

3 Next time.

- Module 4: Driving the Robot