Programming Familiarization Part 2

ROS Familiarization (ROS message/comms protocols)

Goal

The main goal of this assignment is to familiarize you with ROS nodes, messages, and services, and the communication systems used by ROS. In this assignment, we will attempt to create a chat room with some low-level intelligent bots. The sample of the result is included in the assignment folder.

As explained in class, the ROS framework relies highly on the publish-and-subscribe system. In this case the chat room has 4 main nodes. The message_ui node is the GUI node where you can send messages to all the other nodes and see the replies given by the other nodes. The counter_node is the node that keeps track of the message traffic. The chatbot_node is the node that will answer any natural language message that you send. The arithmetic_node is the node that will compute and answer any math question that you send in the chat room. There are a few questions at the end that you will need to answer as well.

Deliverables:

1. Put the message_ui, chatbot_node, counter_node and arithmetic_node in a folder. Put the answers to the questions into a PDF file with the name Team[Letter]_AndrewID_Task5.Part2.pdf and put it in the same folder. Compress the folder in .zip format. Name the folder as Team[Letter]_AndrewID_Task5.Part2.zip (eg. TeamC_xxx_Task5.Part2.zip)

General Setup

1. Once you have unzipped the folder, copy the message_ui, chatbot_node, counter_node and arithmetic_node folders into your catkin_ws under the src directory. For example, if your catkin_ws is in the root folder, copy the four folders into ~/catkin_ws/src/.
2. Then move to catkin_ws and run `catkin_make install`. If you have installed ROS correctly, this step should proceed without any problems. If you find a dependency issue, try to fix it.
3. If the catkin_make install command fails, make sure you have sourced the right bash file. You should source the bash file in your devel folder as follows: `source ~/catkin_ws/devel/setup.bash`.

message_ui

1. You do not have to make any modifications in this package other than to uncomment certain lines as you proceed through the assignment.
2. This node is made as a plugin in rqt which serves as the general GUI for ROS.
3. Go through this node if you want to learn more about python, rqt and using ROS in rqt, but do not change this node other than to uncomment the necessary lines.
4. To run this plugin, first run `roscore` on a separate terminal, then run rqt using the command `rqt`. Then click the Plugins drop-down menu and choose Messages GUI. If you don't see the plugin listed, most likely your installation step failed or you have not sourced the correct bash file.

Part 1: 25%

chatbot_node

1. This node replies to all your natural language messages that are sent from the gui.
2. The skeleton code for this node has been provided. You will only need to modify chatbot_node.cpp.
3. The basic structure for the node is given. Here you are required to write code for 3 main components:
    a. Publisher – Publishes the reply message to message_ui
    b. Subscriber – Subscribes to the message from message_ui
    c. Param – Inputs your name as the param and reads it during runtime.
4. The specifications for all the components are as follows:
    a. Publisher:
        i. The topic name to publish to is “reply_msg”
        ii. The message type is reply_msg from the chatbot_node package. If you don’t know how to input the message type for the publisher, please refer to ROS examples found at https://goo.gl/B2NyYA and https://goo.gl/bmxRjQ.
        iii. You can find the message definition for reply_msg under the following path: your workspace/src/chatbot_node/msg
    b. Subscriber:
        i. The topic to subscribe to is "sent_msg"
        ii. The message type is sent_msg from the message_ui package. The message definition can be found under: your workspace/src/message_ui/msg
        iii. The minimum number of messages your subscriber must handle is 3 and one of the replies must be a message with your name. These are hard-coded string messages.
        iv. The three general messages that we recommend are:
            1. User message: Hello Reply message: Hello, your name
            2. User message: What is your name? Reply message: My name is MRSD Siri
            3. User message: How are you? Reply message: I am fine, thank you.
        v. A good reference for C++ string operators and functions: http://www.cplusplus.com/reference/string/string/.
    c. Param
        i. The param is stored in the launch file.
        ii. The name of the param must be “name”
        iii. Useful details about rosparam: https://goo.gl/EjRZyF

NOTE: The general convention is that sent_msg will always be from the message_ui node. The reply_msg will come from other nodes.

Part 2: 25%

counter_node

1. This node keeps track of the 5 main stats for the chat room. The stats are:
    a. Total number of messages (Both sent and replied)
    b. Total number of sent messages (Sent messages are user messages sent via the GUI)
    c. Total number of replied messages (Replied messages are reply messages sent by the chatbot_node and the arithmetic_node collectively)
    d. Time elapsed from last replied message: How long has it been since either the chatbot_node or the arithmetic_node replied to a user message?
    e. Time elapsed since the last sent message: How long has it been since the last message the user sent?
2. The skeleton code for this node also has been provided. All the stats calculations have been handled except for the time elapsed values. These can be calculated using ROS Time. More details: http://wiki.ros.org/roscpp/Overview/Time. Also see the message definitions to figure out how to get the message sent/received times.
3. The basic structure for this node is given. The main component that you will have to complete for this node is the service that this node provides.
4. The service must be named counter. The request must be an int16 value and must be named req_id. The response must be a float32 value and must be named reply. You will need to create the service file based on the instructions available at http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
5. Once the service has been created, you can write the code for it in the counter_node.cpp file.
6. The specifications for the service are as follows:
    a. There are 5 request ids:
        i. 0: Total number of messages
        ii. 1: Total number of replied messages (replies from both arithmetic_node and chatbot_node)
        iii. 2: Total number of sent messages (Messages sent by the user)
        iv. 3: Time elapsed since last reply message (reply from either chatbot_node or arithmetic_node, whichever is the latest)
        v. 4: Time elapsed since last user message
    b. Request for the counter values will be sent from the message_ui node. You will have to handle the request and send the response based on the request id received.
    c. The service must be advertised as “message_counter”
    d. More info on services available at: http://wiki.ros.org/roscpp/Overview/Services
7. To use counter_node, uncomment:
    a. message_ui/src/message_ui/message_gui.py:
        i. Line 45 - from counter_node.srv import *
        ii. Line 118 to 126 - Uncomment the whole function:
        def _on_send_request_pressed(self):
        iii. Line 102 -
        self._widget.send_request.pressed.connect(self._on_send_request_pressed)
    b. chatbot_node/launch/launch_chatbot.launch

i. Uncomment the <node> tag with the counter_node values:
<node name="counter_node" pkg="counter_node" type="counter_node"></node>

Part 3: 45%

arithmetic_node

1. No skeleton code will be provided for this node. You will have to create this node from scratch. It has similar structure to the chatbot_node. You should be able to re-use a lot of the components available from that node. This will help you gauge your understanding of ROS basics.
2. This node must adhere to the following specifications strictly. If the specifications are not followed, your final submission might fail our test cases and thus lose points. You are free to modify anything not mentioned as part of the specifications.
    a. This node and package must be named arithmetic_node. The package must be in C++.
    b. You must define a custom message under the name of "arithmetic_reply". This message must contain the following fields (Please follow the field names exactly. They are case-sensitive):
        i. Field type: Header Field name: header
        ii. Field type: string Field name: oper_type
        iii. Field type: float32 Field name: answer
        iv. Field type: float64 Field name: time_received
        v. Field type: float64 Field name: time_answered
        vi. Field type: float64 Field name: process_time
    c. This node will carry out 4 different operations:
        i. Add
        ii. Subtract
        iii. Multiply
        iv. Divide
    d. This node publishes its reply under the topic of "arithmetic_reply". This publisher publishes arithmetic_reply messages that you had defined earlier.
    e. This node subscribes to the "sent_msg" topic with message type sent_msg defined under the message_ui.
    f. The looping rate must be 20 so that it is consistent with the other nodes.
    g. The messages that will be sent through the message_ui will be of the following format:
        i. Add: "1+1"
        ii. Subtract: "1-1"
        iii. Multiply: "1*1"
        iv. Divide: "1/1"
        v. Remember to follow the same format when you decode the message in the arithmetic node. Note that there are no spaces in between the numbers and the operator.

h. The data that will be carried by the message are as follows:
    i. header: the time at which the message is sent must be recorded under header.stamp
    ii. oper_type: The type of operation. Based on the sign found in the user message, you need to input one of the following values in this field: “Add”, “Subtract”, “Multiply”, ”Divide”.
    iii. answer: Fill in the answer to the mathematical equation in this field.
    iv. time_received: The time at which the arithmetic node received the user message.
    v. time_answered: The time at which the calculation is complete.
    vi. process_time: The time difference between time_received and time_answered

3. To start using arithmetic_node, uncomment:
    a. counter_node/src/counter_node.cpp
        i. Uncomment the whole arithmetic_reply_msg_callback(const arithmetic_node::arithmetic_reply msg) function
        ii. Uncomment the line: arithmetic_reply_msg_sub = n.subscribe("arithmetic_reply", 1000, arithmetic_reply_msg_callback);
        iii. Uncomment the line: #include <arithmetic_node/arithmetic_reply.h>
    b. counter_node/CMakeLists.txt:
        i. Uncomment: add_dependencies(counter_node arithmetic_node_generate_messages_cpp)
    c. chatbot_node/launch/launch_chatbot.launch:
        i. Uncomment the <node> tag with the counter_node values: <node name="arithmetic_node" pkg="arithmetic_node" type="arithmetic_node"></node>
    d. message_ui/src/message_ui/message_gui.py:
        i. Lines 53 to 59: Uncomment the function def arithmetic_reply_msg_callback(self, msg_in):
        ii. Line 43: from arithmetic_node.msg import arithmetic_reply
        iii. Line 83: rospy.Subscriber("arithmetic_reply", arithmetic_reply, self.arithmetic_reply_msg_callback)

Part 4: 5%

Questions:

1. Briefly explain how services in ROS work. A few lines will suffice.
2. Include a ROS node graph of your nodes. This can be automatically generated using rqt (http://wiki.ros.org/rqt_graph). This graph needs to include all the nodes and topics.
3. Do you think the assignment was helpful for you to get started in ROS? If not, how would you improve it?
4. Was the assignment difficulty manageable?

Bonus: Not Graded

This part is only for your own enrichment. Try to figure out how to run the message_ui on one computer and the other nodes on another computer.