Workshop 3 Creating a ROS Package

Aims and Objectives

Understanding the ROS package structure and creating a ROS package.

Background Theory

1. ROS package structure
2. Integration and programming with Eclipse
3. ROS C++ client library (roscpp)
4. ROS subscribers and publishers
5. ROS parameter server
6. RViz visualization

Exercise

Please note that you should replace the MystudentID with your ID with your initials. For example, if ID is 123456 and my initial is z. I will need to replace MystudentID with z123456. This will apply to all the labs and all the code which has text string "MystudentID" in it.

1. Creating a ROS package

In this exercise, you will learn step by step how to create your first ROS package. The package should in the end be able to subscribe to a laser scan message from the Husky robot and process the incoming data. This node will be the basis for the next exercises. Use Eclipse already installed to edit your package.

Open 4 terminals as demonstrated by the instructor in the class and do:

a. cd ~/catkin_ws/src

b. catkin_create_pkg ***MystudentID***_demo_pkg roscpp std_msgs actionlib actionlib_msgs

c. cd ~/catkin_ws

d. catkin_make

e. source devel/setup.bash

f. cd ~/catkin_ws/src/MystudentID_demo_pkg/src

g. Create two files named: demo_topic_publisher.cpp and demo_topic_subscriber.cpp as follows. You can copy and paste the codes to your program.

demo_topic_publisher.cpp

```
#include "ros/ros.h"

#include "std_msgs/Int32.h"

#include <iostream>

int main(int argc, char **argv)
```

```
{
  ros::init(argc, argv, "demo_topic_publisher");
  ros::NodeHandle node_obj;
  ros::Publisher number_publisher =
    node_obj.advertise<std_msgs::Int32>("/numbers", 10);
  ros::Rate loop_rate(10);
  int number_count = 0;
  while (ros::ok())
  {
    std_msgs::Int32 msg;
    msg.data = number_count;
    ROS_INFO("%d", msg.data);
    number_publisher.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++number_count;
  }
  return 0;
}
```

demo_topic_subscriber.cpp

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>

//Callback of the topic /numbers
void number_callback(const std_msgs::Int32::ConstPtr& msg)
```

```
{
        ROS_INFO("Recieved  [%d]",msg->data);
}


int main(int argc, char **argv)
{


        //Initializing ROS node with a name of demo_topic_subscriber
        ros::init(argc, argv,"demo_topic_subscriber");
        //Created a nodehandle object
        ros::NodeHandle node_obj;
        //Create a publisher object
        ros::Subscriber number_subscriber =
node_obj.subscribe("/numbers",10,number_callback);
        //Spinning the node
        ros::spin();
        return 0;
}
```

h. **cd ~/catkin_ws/src/MystudentID_demo_pkg and edit the CMakeLists.txt file by adding the following codes to it (copy and paste, please remember to change MystudentID !)**

```
#This will create executables of the nodes
add_executable(demo_topic_publisher src/demo_topic_publisher.cpp)
add_executable(demo_topic_subscriber src/demo_topic_subscriber.cpp)
#This will generate message header file before building the target
add_dependencies(demo_topic_publisher MystudentID_ros_demo_pkg_generate_messages_cpp)
add_dependencies(demo_topic_subscriber MystudentID _ros_demo_pkg_generate_messages_cpp)
#This will link executables to the appropriate libraries
```

target_link_libraries(demo_topic_publisher ${catkin_LIBRARIES})

target_link_libraries(demo_topic_subscriber ${catkin_LIBRARIES})

- ***i. cd ~/catkin_ws***
- ***j. catkin_make***
- ***k. in one of the terminal run command: $ roscore***
- ***l. in terminal 2 run command: $*** **rosrun *MystudentID*_ros_demo_package demo_topic_publisher**
- ***m. In terminal 3 run: $*** **rosrun *MystudentID*_ros_demo_package demo_topic_subscriber**
- ***n. Snapshot the output of each window and put to your logbook.***
- ***o. In terminal 4 run: $ rqt_graph and snapshot the output and record it to your logbook***

2. Working with messages

we will look at how to create custom messages and services definitions in the current package. The message definitions are stored in a .msg file and the service definitions are stored in a .srv file. Msg files will be saved to msg folder and srv files will be saved to srv folder.

- ***a. cd ~/catkin_ws/src/MystudentID_ros_demo_pkg/***
- ***b. mkdir msg***
- ***c. mkdir srv***
- ***d. echo "string greeting" > msg/demo_msg.msg***
- ***e. echo "int32 number" >> msg/demo_msg.msg***
- ***f. echo "string in" > srv/demo_srv.srv***
- ***g. echo "---">> srv/demo_srv.srv***
- ***h. echo "string out" >> srv/demo_srv.srv***

Up to now, we have created the msg and srv subfolder and files in each folder, please check if it is what you want. If we want to use not just standard messages, we should also make some change of the **package.xml** file (be patient at the beginning, here we learn step by step how to program. In the later workshops, I should provide the files for you to run straight away).

Edit the package.xml file of the current package and uncomment the lines <build_depend>message_generation</build_depend> and <exec_depend>message_runtime</exec_depend>.
Edit the current CMakeLists.txt and add the message_generation line, as follows:

```
find_package(catkin REQUIRED COMPONENTS
roscpp
rospy
std_msgs
actionlib
actionlib_msgs
message_generation
)
```

Uncomment the following line and add the custom message file:

```
add_message_files(
FILES
demo_msg.msg
)
## Generate added messages and services with any dependencies listed here
generate_messages(
DEPENDENCIES
std_msgs
actionlib_msgs
)
```

After these steps, we can compile and build the package

i. ***cd ~/catkin_ws/***
j. ***catkin_make***
k. **run: *rosmsg show MystudentID_ros_demo_pkg/demo_msg*** *and record the output to your logbook*
l. copy **demo_msg_publisher.cpp and demo_msg_subscriber.cpp files to your src folder of MystudentID_ros_demo_pkg package (The source files are in ~/Eng_7_rob/workshop_3/MStudentID_demo_pkg/src folder)**

m. ***modify CMakeLists.txt and add the flowing codes to it:***

```
add_executable(demo_msg_publisher src/demo_msg_publisher.cpp)
add_executable(demo_msg_subscriber src/demo_msg_subscriber.cpp)
add_dependencies(demo_msg_publisher
MystudentID_ros_demo_pkg_generate_messages_cpp)
add_dependencies(demo_msg_subscriber
MystudentID_ros_demo_pkg_generate_messages_cpp)
target_link_libraries(demo_msg_publisher ${catkin_LIBRARIES})
target_link_libraries(demo_msg_subscriber ${catkin_LIBRARIES})
```

n. ***build the package using catkin_make and follow the similar steps of running demo_topic_publisher etc. to run the demo_msg program by:***
o. *in one of the terminal run command: $ roscore*
p. *in terminal 2 run command: $* rosrun *MystudentID*_ros_demo_package demo_**msg**_publisher
q. *In termimal 3 run: $* rosrun *MystudentID*_ros_demo_package demo_msg_subscriber
r. *Snapshot the output of each window and put to your logbook and you will see the difference.*
s. *In terminal 4 run: $ rqt_graph and snapshot the output and record it to your logbook*

3. Working with services

We have added the srv in last exercise. We should make some change of package.xml and CMakeLists.txt file. To simply please copy these two files from **~/Eng_7_rob/workshop_3/MStudentID_demo_pkg/** and replace the files in current MystudentID_ros_demo_pkg/ folder and copy demo_service_server.cpp and demo_service_client.cpp to MystudentID_ros_demo_pkg/src/ folder. Follow the first two exercises to build the package and run service server and client program by:

```
$ rosrun MystudentID_ros_demo_pkg demo_service_server
$ rosrun MystudentID_ros_demo_pkg demo_service_client
```

Record the result to your logbook