ROS Workshop - Tutorial 4 - Create Package

ME 4140 - Introduction to Robotics - Fall 2020

Overview:

After completing *Tutorial 3 - Turtlesim*, You have begun learning ROS and you can are ready to create a custom C++ package. You can read more here on the wiki.

System Requirements:

- **ROS+OS**: This tutorial is intended for a system with ROS Melodic installed on the Ubuntu 18.04 LTS operating system. Alternate versions of ROS (i.e. - Kinetic, Noetic, etc.) may work but have not been tested. Versions of ROS are tied to versions of Ubuntu.
- **ROS:** Your computer must be connected to the internet to proceed. Update the system before you begin.

Disclaimer:

- **Backup the System:** If you are using a virtual machine, it is recommend to make a snaphot of your virtual machine before you start each module. In the event of an untraceable error, you can restore to a previous snapshot.
- **Workspace Setup:** In Part I you will setup a Catkin Workspace as your working directory for creating packages. This only needs to be done once.

Important Note on Naming:

In this tutorial you will replace several <fields> with names of your choice. These are general guidlines for naming in ROS.

- use descriptive names, very long or very short names are hard to read
- **do not** include the < > symbols
- **do not** use spaces, UPPER CASE letters, or special characters ($@, \$, *, etc.$)
- the underscore _ character **is** allowed

<workspace_name> - name of your workspace

<package_name> - name of your package

<node_name> - name of your node

<user_name> - ubuntu user name

Part I - Setup the Workspace: (Part I only needs to be done once Fall2020.)

Before building a custom ROS package you need to setup a *catkin workspace* as the working directory. Catkin is the program that manages the file system behind the scenes and compiles your .cpp code.

Step 1: Source the installation files needed to create a workspace. This requires ROS to be previously installed.

```
source /opt/ros/melodic/setup.bash
```

Step 2: Open a new terminal and navigate to the future location of your workspace.

```
cd ~        OR        cd /home/<user_name>
```

Step 3: Choose a workspace name and create a workspace and source directory with *mkdir*. This step determines the location of your new workspace.

```
mkdir -p ~/<workspace_name>/src
```

Step 4: Navigate to the top of your workspace directory and build your workspace.

```
cd ~/<workspace_name>
```

```
catkin_make
```

Step 5: Now add your workspace directory to .bashrc and source the script.

```
echo "source ~/<workspace_name>/devel/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

Open the **.bashrc** file with the gedit text editor. You can see the lines you have added with **echo** » at the bottom of the file. Close the file.

```
gedit ~/.bashrc
```

Part II - Create A Publisher Node:

You can write custom nodes for your ROS system in C++, Python, or Lisp. These documents will support C++.

Step 1: Create a new package in your workspace for your new node to belong to. Make sure to do this in the correct parent directory .

```
cd ~/<workspace_name>/src
```

```
catkin_create_pkg <package_name> std_msgs rospy roscpp
```

Step 2: Back out to the workspace directory then compile your package with catkin_make

```
cd ~/<workspace_name>          OR          cd ..
```

```
catkin_make
```

If you get here with no errors, your workspace is setup, and you are ready to write some code and test your new package!

Step 3: Create a new file for your C++ **publisher node** from the command line. The text editor *gedit* will create and open a new file named <node_name>in the current directory.

```
gedit ~/<workspace_name>/src/<package_name>/src/<node_name>.cpp
```

Copy the code below into the source file. .

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include <sstream>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "replace_with_your_node_name");
    ros::NodeHandle n;
    ros::Publisher ttu_publisher =
        n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1000);
    ros::Rate loop_rate(10);

    int count = 0;
    while (ros::ok())
    {
        geometry_msgs::Twist msg;
        msg.linear.x = 2+0.01*count;
        msg.angular.z = 2;
        ttu_publisher.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        count++;
    }
}
```

Save and close the file. It must be saved as a <node_name>.cpp in the src directory of the package your created in previously in **Part I**

Step 4: Before we can compile the node we have to modify the file below.

```
gedit ~/<workspace_name>/src/<package_name>/CMakeLists.txt
```

Add the following lines to the bottom of the file and save.

```
add_executable(<node_name> src/<node_name>.cpp)
target_link_libraries(<node_name> ${catkin_LIBRARIES})
```

Step 5: Compile and test the new publisher node. This will compile and build your source code as well as check for errors in your entire workspace.

```
cd ~/<workspace_name>
```

```
catkin_make
```

Start a core

```
roscore
```

Turn on a turtle.

```
rosrun turtlesim turtlesim_node
```

Start your new node

```
rosrun <package_name> <node_name>
```

Use rostopic to view current topics.

```
rostopic list
```

Close your node and start it again with the cmd_vel topic mapped to the turtle like we did previously.

```
rosrun <package_name> <node_name> /cmd_vel:=/turtle1/cmd_vel
```

Part III - Create A Subscriber Node:

Now create a **subscriber node** in the same package as the previous node.

Step 1: Use the code below called **turtlesim_subscriber.cpp** to start.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
/**
* This tutorial demonstrates simple receipt of messages over the ROS system.
*/
void dataCallback(const geometry_msgs::Twist::ConstPtr& msg)
{
   ROS_INFO("I heard: [%f]", msg->linear.x);
}
int main(int argc, char **argv)
{
   ros::init(argc, argv, "turtlesim_subscriber");
   ros::NodeHandle n;
   ros::Subscriber sub = n.subscribe("/cmd_vel", 1000, dataCallback);
   ros::spin();
   return 0;
}
```

Step 2: Modify the appropriate CMakeLists.txt file as you did previously.

Step 3: Compile the new subscriber node using catkin.

Step 4: Test the new node. Does it work? How do you know?

Tutorial Complete:

After completing *Tutorial 4 - Create Package*, you are finally ready for a more advanced robot simulator.

Bonus Excercise: Install the JoyStick Teleop Node to drive the turtle with a USB joystick.