ROS - Topics, Publishing and Subscribing

ME 4140 - Introduction to Robotics - Fall 2019

- **Topics:** ROS nodes can communicate by **publishing** and **subscribing** to topics. A topic is information generated by a publishing node that is made available to a subscribing node or nodes in the ROS system.
  - A node can publish a topic. This node is a publisher.
  - A node can subscribe to a topic. This node is a subscriber.
  - Most nodes publish and subscribe to multiple topics.
  - Integrate built-in ROS nodes and modify your own ROS nodes in C++, Python, and even MATLAB

- **Setup the Workspace:** To our our own ROS nodes we need to setup a *catkin workspace*. Catkin is the program that manages the file system behind the scenes. It is our working directory or environment in which we can customize our ROS system.

  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 /<user_name>/home
  ```

  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: Before continuing test that your ROS system is setup correctly.

```
source devel/setup.bash
```

```
echo $ROS_PACKAGE_PATH
```

You should see somthing like this in the terminal. This is where ROS is installed.

```
/home/<user_name>/<workspace_name>/src:/opt/ros/melodic/share
```

- **Create Your Own 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 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
  ```

  Step 3: Now source the workspace directory.

  ```
  source ~/<workspace_name>/devel/setup.bash
  ```

  Step 4: Open the **.bashrc** file text editor. Modify the file so that this happens each time you start a new terminal.

  ```
  gedit ~/.bashrc
  ```

  Step 5: Add the following line to the bottom of the file. It may already be there. Save and close the file.

  ```
  source ~/<workspace_name>/devel/setup.bash
  ```

Step 6: Write the **publisher node** in C++. It will start as C++ code and then it will be compiled into an executable. Create a file from the sample code **ttu_publisher.cpp** It needs to be saved in the correct directory.

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

Copy the code below into the source file. It must be saved as a <node_name>.cpp

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

int main(int argc, char **argv)
{
    ros::init(argc, argv, "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++;
    }
}
```

Step 7: 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})
add_dependencies(<node_name> beginner_tutorials_generate_messages_cpp)
```

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
```

Start your new node

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

Use rostopic to view current topics.

```
rostopic list
```

Now lets do something more fun. Turn on a turtle.

```
rosrun turtlesim turtlesim_node
```

Now start your publisher node with the cmd_vel topic patched through to the turtle like we did previously.

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

- Now create a **subscriber node** in the same package as the previous node. You can follow the tutorial here.

  Step 1: Use the code below called **ttu_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, "ttu_subscriber");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("/cmd_vel", 1000, dataCallback);
  ros::spin();
  return 0;
}
```

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

  Step 3: Compile the new subscriber node.

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

- Bored with all of that? Try this JoyStick Teleop Node for use with a Linux compatable joystick.