ROS Workshop - Tutorial 7 - Turtlebot3 in Brown Hall

ME 4140 - Introduction to Robotics - Fall 2020

What is the goal of the. This tutorial comes from here.

Overview:

After completing *Tutorial 6 - Turtlebot Navigation*, You have learned some basics of ROS, and you have a for a more advanced robot. Next you are going to learn to load a custom world in Gazebo. Read more here.

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.
- **Workspace Setup:** The Turtlebot3 Simulator from tutorial 6 must be operational before completing tutorial 7.

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.
- ROSLAUNCH: This tutorial involves using the roslaunch command which runs a muliple of nodes at once as described in the launch file. We will learn more about this later.
- Mouse for 3D viewing: This simulator view is much easier use if you have a three button mouse plugged in, but this is not required.

Create a new node with the name of your choosing. You can put this node in the package you created previously for turtlebot3. Change directory to the source folder of the package and enter the following command to open a new file for your source code.

```
gedit ~/<workspace_name>/src/turtlebot3_control/src/publish_goal.cpp
```

Now copy the example code below into the the source file. Save the file after editing.

```
#include "ros/ros.h"
#include "geometry_msgs/PoseStamped.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::PoseStamped>("<topic_name>", 1000);
    ros::Rate loop_rate(10);

    geometry_msgs::PoseStamped msg;
    msg.header.stamp=ros::Time::now();
    msg.header.frame_id="map";

    int count = 0;
    while ((ros::ok())&&(count<5))
    {
        msg.pose.position.x = 3.0;
        msg.pose.position.y = 2.0;
        msg.pose.position.z = 0;
        msg.pose.orientation.w = 1.0;

        ttu_publisher.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        count++;
    }
}
```

Edit the *CMakeLists.txt* file for this specific package.

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

Add the following lines to the bottom of the file.

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

Compile the code with catkin_make before running. Turn Everything off and navigate to the workspace before you do this.

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

Now test your node. Start the turtlebot3 simulator first.

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Next, turn on naviagtion. This requires that you have previously made a map.

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=map.yaml
```

Finally run your new node and you should see your robot move to location programmed in your source file!

```
rosrun turtlebot3_control publish_goal
```

Now let us make a node called **subscribe_status** that can access information from the robot in the form of a topic. To do this we are going to make a new node that is part of the same package we just made/used. Open a new file in the proper src folder and insert the following cpp code.

```
gedit ~/<workspace_name>/src/turtlebot3_control/src/subscribe_status.cpp
```

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "actionlib_msgs/GoalStatusArray.h"

void statusCB(const actionlib_msgs::GoalStatusArray::ConstPtr& msg)
{
    ROS_INFO("Subscriber Callback Executed");
    if (!msg->status_list.empty())
    {
        actionlib_msgs::GoalStatus goalStatus = msg->status_list[0];
        ROS_INFO("Status Recieved: %i",goalStatus.status);
    }
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "subscribe_status");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("<topic_name>", 1000, statusCB);
    ros::spin();
    return 0;
}
```

Open the correct *CMakeLists.txt* file for this specific node as you did previously.

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

Add the following lines to the bottom.

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

Compile before running then test you new node! You should see the status information printed in the terminal window.