Note: This tutorial assumes that you have completed the previous tutorials: ROS tutorials (/ROS/Tutorials).

Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Sending Goals to the Navigation Stack

Description: The Navigation Stack serves to drive a mobile base from one location to another while safely avoiding obstacles. Often, the robot is tasked to move to a goal location using a pre-existing tool such as rviz in conjunction with a map. For example, to tell the robot to go to a particular office, a user could click on the location of the office in a map and the robot would attempt to go there. However, it is also important to be able to send the robot goals to move to a particular location using code, much like rviz does under the hood. For example, code to plug the robot in might first detect the outlet, then tell the robot to drive to a location a foot away from the wall, and then attempt to insert the plug into the outlet using the arm. The goal of this tutorial is to provide an example of sending the navigation stack a simple goal from user code.

Tutorial Level: BEGINNER

Contents

catkin | rosbuild

1. Pre-Requisites

This tutorial assumes basic knowledge of how to bring up and configure the navigation stack. This may be different depending on the platform that you're running on. Below is a list of tutorials for known platforms at the time of this writing. Please feel free to update it as more tutorials for different robots become available.

- Running the navigation stack on the PR2 platform (/pr2_2dnav)
- Simulating the 2dnav Stack (/pr2_2dnav_gazebo/Tutorials/Simulating%20the%202dnav%20Stack)

2. Some ROS Setup

In order to create a ROS node that sends goals to the navigation stack, the first thing we'll need to do is create a package. To do this we'll use the handy command where we want to create the package directory with a dependency on the move_base_msgs, actionlib, and roscpp packages as shown below:

```
$ catkin_create_pkg simple_navigation_goals move_base_msgs actionlib roscpp
```

After this is done we'll need to roscd to the package we created, since we'll be using it as our workspace

```
roscd simple_navigation_goals
```

3. Creating the Node

Now that we have our package, we need to write the code that will send goals to the base. Fire up a text editor and paste the following into a file called **src/simple_navigation_goals.cpp**. Don't worry if there are things you don't understand, we'll walk through the details of this file line-by-line shortly.

Toggle line numbers

```
 1 #include <ros/ros.h>
 2 #include <move_base_msgs/MoveBaseAction.h>
 3 #include <actionlib/client/simple_action_client.h>
 4
 5 typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> Mov
eBaseClient;
 6
 7 int main(int argc, char** argv){
 8   ros::init(argc, argv, "simple_navigation_goals");
 9
10   //tell the action client that we want to spin a thread by default
11   MoveBaseClient ac("move_base", true);
12
13   //wait for the action server to come up
14   while(!ac.waitForServer(ros::Duration(5.0))){
15     ROS_INFO("Waiting for the move_base action server to come up");
16   }
17
18   move_base_msgs::MoveBaseGoal goal;
19
20   //we'll send a goal to the robot to move 1 meter forward
21   goal.target_pose.header.frame_id = "base_link";
22   goal.target_pose.header.stamp = ros::Time::now();
23
24   goal.target_pose.pose.position.x = 1.0;
25   goal.target_pose.pose.orientation.w = 1.0;
26
27   ROS_INFO("Sending goal");
28   ac.sendGoal(goal);
29
30   ac.waitForResult();
31
32   if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
33     ROS_INFO("Hooray, the base moved 1 meter forward");
34   else
35     ROS_INFO("The base failed to move forward 1 meter for some reason");
36
37   return 0;
38 }
```

Now, we'll break the code down line by line:

Toggle line numbers

```
 2 #include <move_base_msgs/MoveBaseAction.h>
 3
```

This line includes the action specification for move_base which is a ROS action that exposes a high level interface to the navigation stack. Essentially, the move_base action accepts goals from clients and attempts to move the robot to the specified position/orientation in the world. For a detailed discussion of

ROS actions see the actionlib documentation (/actionlib).

Toggle line numbers

```
   5 typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> Mov
eBaseClient;
```

This line creates a convenience typedef for a SimpleActionClient that will allow us to communicate with actions that adhere to the MoveBaseAction action interface.

Toggle line numbers

```
10   //tell the action client that we want to spin a thread by default
11   MoveBaseClient ac("move_base", true);
```

This line constructs an action client that we'll use to communicate with the action named "move_base" that adheres to the MoveBaseAction interface. It also tells the action client to start a thread to call ros::spin() so that ROS callbacks will be processed by passing "true" as the second argument of the MoveBaseClient constructor.

Toggle line numbers

```
13   //wait for the action server to come up
14   while(!ac.waitForServer(ros::Duration(5.0))){
15     ROS_INFO("Waiting for the move_base action server to come up");
16   }
```

These lines wait for the action server to report that it has come up and is ready to begin processing goals.

Toggle line numbers

```
18   move_base_msgs::MoveBaseGoal goal;
19
20   //we'll send a goal to the robot to move 1 meter forward
21   goal.target_pose.header.frame_id = "base_link";
22   goal.target_pose.header.stamp = ros::Time::now();
23
24   goal.target_pose.pose.position.x = 1.0;
25   goal.target_pose.pose.orientation.w = 1.0;
26
27   ROS_INFO("Sending goal");
28   ac.sendGoal(goal);
```

Here we create a goal to send to move_base using the `move_base_msgs::MoveBaseGoal` message type which is included automatically with the MoveBaseAction.h header. We'll just tell the base to move 1 meter forward in the "base_link" coordinate frame. The call to `ac.sendGoal` will actually push the goal out over the wire to the move_base node for processing.

Toggle line numbers

```
30   ac.waitForResult();
31
32   if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
33     ROS_INFO("Hooray, the base moved 1 meter forward");
34   else
35     ROS_INFO("The base failed to move forward 1 meter for some reason");
```

The only thing left to do now is to wait for the goal to finish using the `ac.waitForGoalToFinish` call which will block until the move_base action is done processing the goal we sent it. After it finishes, we can check if the goal succeded or failed and output a message to the user accordingly.

4. Building and Running

Now that we have a package and a source file, we'll want to build and then try things out. The first step will be to add our `src/simple_navigation_goals.cpp` file to our `CMakeLists.txt` file to get it to build. Open up `CMakeLists.txt` in your editor of choice and add the following line to the bottom of the file.

```
add_executable(simple_navigation_goals src/simple_navigation_goals.cpp)
target_link_libraries(simple_navigation_goals ${catkin_LIBRARIES})
```

Once this is done, we can build our executable by typing make.

```
$ catkin_make
```

Now we're ready to run. At this point, we'll assume that the navigation stack has been brought up according to one of the tutorials listed above (/navigation/Tutorials/SendingSimpleGoals#Pre-Requisites). One important thing to check is that the name of the action, we've assumed "move_base" in this tutorial, matches the name used in your code. To check this, we'll execute the following rostopic command:

```
rostopic list | grep move_base/goal
```

If something shows up, then you're good to go. Otherwise, you need to find the name of the action that the navigation stack is using, a common alternative on the PR2 platform is "move_base_local," and update all references in the `src/simple_navigation_goals.cpp` file accordingly.

After this, its as simple as running the executable we created.

```
./bin/simple_navigation_goals
```

And, if all goes well, the robot should begin to move a meter forward.

5. Learning more about map navigation

You can learn more about map navigation through Gaitech EDU (http://edu.gaitech.hk/) tutorial on Map Navigation (http://edu.gaitech.hk/turtlebot/map-navigation.html)

Wiki: navigation/Tutorials/SendingSimpleGoals (last edited 2016-08-25 10:01:05 by AnisKoubaa (/AnisKoubaa))