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.

Using the low-level robot base controllers to drive the robot

Description: This tutorial teaches you how to start up and control the default robot base controllers (pr2_base_controller and pr2_odometry) directly, rather than at a high level (using move_base (/move_base)).

Keywords: move, moving, base, casters, controllers, driving, drive, robot, PR2, wheels

Tutorial Level: BEGINNER

Next Tutorial: Using the base controller with odometry and transform information pr2_controllers/Tutorials/Using the base controller with odometry and transform information (/pr2_controllers/Tutorials/Using%20the%20base%20controller%20with%20odometry%20and%20transform%20information)

Contents

0.1 Overview and prerequisites

In order to send velocity commands to the base, we need the following components:

- the base velocity controller that directly commands the motors
- the base odometry controller
- a higher-level program to send the velocity commands

The first two components above are available in ROS. In this tutorial we will show how to use them by writing the third component, the higher-level program.

Bring up a robot, either on the hardware (/pr2_bringup/Tutorials) or in gazebo (/pr2_simulator).

Note that, in standard robot start-up, the arms are not controlled and free to move. As a result, they might move around due to inertia as you are driving the base.

0.1 The base velocity and odometry controller

In this tutorial, we will be moving the base by using the base velocity and odometry controllers. Both of these controllers are normally loaded on robot start-up. To check, after starting the robot, use the `pr2_controller_manager` :

```
rosrun pr2_controller_manager pr2_controller_manager list
```

If you see both

- `base_controller (running)` and
- `base_odometry (running)`

then the controllers are ready and we can proceed to use them.

If you see no mention of either `base_controller` or `base_odometry`, then something went wrong during robot start-up. Abort this tutorial and investigate the problem.

0.1 The controller message

The base controller receives CodeAPI: Twist (http://www.ros.org/doc/api/geometry_msgs/html/msg/Twist.html) messages on the '/base_controller/command' topic, which has two CodeAPI: Vector3 (http://www.ros.org/doc/api/geometry_msgs/html/msg/Vector3.html) fields: linear and angular.

Nonzero entries in the x and y fields of linear causes the robot to move forwards and backwards (x), or strafe left and right (y), in the robot's base odometry frame. A nonzero entry in the z field of angular causes the robot to turn (yaw). The size of the entries changes the velocity of the robot, and a single command will only move the robot for a short period of time before stopping, so it does not run off into the wall (or you) when commands stop coming for any reason. (For the PR2, linear z and angular x and y are just ignored.)

Velocities are in units of m/s and rad/s, as in all of ROS.

Note that, for now, you don't have to worry about commanding the `odometry controller`; it just needs to be running. We will use it explicitly in the next tutorial.

0.1 Package setup

In order to create a ROS node that sends goals to the trajectory controller, the first thing we'll need to do is create a package. To do this we'll use the handy roscreate-pkg command where we want to create the package directory:

```
roscreate-pkg drive_base_tutorial roscpp geometry_msgs
```

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

```
roscd drive_base_tutorial
```

0.1 Creating the node (C++)

In this tutorial, we will write code in C++ for moving the robot using keyboard commands. For this, we do not need to worry about odometry, distance traveled, etc. An example showing how to have the robot move a certain distance based on odometry information will be shown in the next tutorial (/pr2/Tutorials/Using%20the%20base%20controller%20with%20odometry%20and%20transform%20information).

Put the following into **src/drive_base.cpp**:

Toggle line numbers

```
#include <iostream>

#include <ros/ros.h>
#include <geometry_msgs/Twist.h>

class RobotDriver
{
private:
  //! The node handle we'll be using
  ros::NodeHandle nh_;
  //! We will be publishing to the "/base_controller/command" topic to issue commands
  ros::Publisher cmd_vel_pub_;

public:
  //! ROS node initialization
  RobotDriver(ros::NodeHandle &nh)
  {
    nh_ = nh;
    //set up the publisher for the cmd_vel topic
    cmd_vel_pub_ = nh_.advertise<geometry_msgs::Twist>("/base_controller/command", 1);
  }

  //! Loop forever while sending drive commands based on keyboard input
  bool driveKeyboard()
  {
    std::cout << "Type a command and then press enter. "
      "Use '+' to move forward, 'l' to turn left,"
      "'r' to turn right, '.' to exit.\n";

    //we will be sending commands of type "twist"
    geometry_msgs::Twist base_cmd;

    char cmd[50];
    while(nh_.ok()){

      std::cin.getline(cmd, 50);
      if(cmd[0]!='+' && cmd[0]!='l' && cmd[0]!='r' && cmd[0]!='.')
      {
        std::cout << "unknown command:" << cmd << "\n";
        continue;
      }

      base_cmd.linear.x = base_cmd.linear.y = base_cmd.angular.z = 0;
      //move forward
      if(cmd[0]=='+'){
        base_cmd.linear.x = 0.25;
      }
      //turn left (yaw) and drive forward at the same time
      else if(cmd[0]=='l'){
        base_cmd.angular.z = 0.75;
        base_cmd.linear.x = 0.25;
      }
      //turn right (yaw) and drive forward at the same time
      else if(cmd[0]=='r'){
        base_cmd.angular.z = -0.75;
        base_cmd.linear.x = 0.25;
      }
      //quit
      else if(cmd[0]=='.'){
        break;
      }

      //publish the assembled command
      cmd_vel_pub_.publish(base_cmd);
    }
    return true;
  }

};

int main(int argc, char** argv)
{
  //init the ROS node
  ros::init(argc, argv, "robot_driver");
  ros::NodeHandle nh;

  RobotDriver driver(nh);
  driver.driveKeyboard();
}
```

0.1 Building and running

Add the following line to the CMakeLists.txt:

```
rosbuild_add_executable(drive_base src/drive_base.cpp)
```

and make the binary by typing `make` in the drive_base_tutorial directory.

Run the drive_base program:

```
bin/drive_base
```

You should now be able to drive the robot around in the window running `drive_base` . You may jump to the next tutorial if you are not interested in doing the same with Python.

0.1 Creating the node (Python)

The code given below is parallel to the above code, just that this works in Python. The code can be downloaded from the following Github link. Base_Driver Python (https://github.com/lharikrishnan1993/PR2controllerspy/blob/master/teleop_py.py) or can be copied from here.

Put the code into **src/teleop_py.py**:

```
import rospy
from geometry_msgs.msg import Twist
import sys

twist = Twist()

def values():
    print '(w for forward, a for left, s for reverse, d for right,k for turning left,l fo
r turning right and . to exit)' + '\n'
    s = raw_input(':- ')
    if s[0] == 'w':
        twist.linear.x = 1.0
        twist.angular.z = 0.0
        twist.linear.y = 0.0
    elif s[0] == 's':
        twist.linear.x = -1.0
        twist.angular.z = 0.0
        twist.linear.y = 0.0
    elif s[0] == 'd':
        twist.linear.y = -1.0
        twist.angular.z = 0.0
        twist.linear.x = 0.0
    elif s[0] == 'a':
        twist.linear.y = 1.0
        twist.angular.z = 0.0
        twist.linear.x = 0.0
    elif s[0] == 'k':
        twist.angular.z = 2.0
        twist.linear.x = twist.linear.y = 0.0
    elif s[0] == 'l':
        twist.angular.z = -2.0
        twist.linear.x = twist.linear.y = 0.0
    elif s[0] == '.':
        twist.angular.z = twist.linear.x = twist.linear.y = 0.0
        sys.exit()
    else:
        twist.linear.x = twist.linear.y = twist.angular.z = 0.0
        print 'Wrong command entered \n'
    return twist

def keyboard():
    pub = rospy.Publisher('base_controller/command',Twist, queue_size=1)
    rospy.init_node('teleop_py',anonymous=True)
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
        twist = values()
        pub.publish(twist)
        rate.sleep()

if __name__ == '__main__':
    try:
        keyboard()
    except rospy.ROSInterruptException:
        pass
```

0.1 Building and running

Type the following in the command file to make the python file executable. (Do this in the directory where you have stored the Python script)

```
$ chmod +x teleop_py.py
```

After this is done remember to do catkin_make to build your files. This can be done as follows:

```
$ cd ~/catkin_ws
$ catkin_make
```

Now you should be able to run the teleop_py.py in the src folder. Type the following command in the right directory to get the file running:

```
$ python teleop_py.py
```

In the next tutorial, we will add information from odometry to make the robot travel for a specified distance: Using the base controller with odometry and transform information (/pr2_controllers/Tutorials/Using%20the%20base%20controller%20with%20odometry%20and%20transform%20information)