Note: This tutorial assumes that you have completed the previous tutorials: Using the robot base controllers to drive the robot (/pr2_controllers/Tutorials/Using%20the%20robot%20base%20controllers%20to%20drive%20the%20robot) and have a basic understanding of the Transform Library (/tf).

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 base controller with odometry and transform information

Description: We move the robot forward by a specified amount by using the low-level base controller together with transform information from odometry.

Keywords: move base, odometry

Tutorial Level: INTERMEDIATE

Contents

0.1 Overview

In running drive_base in the previous tutorial (Using the robot base controllers to drive the robot (/pr2_controllers/Tutorials/Using%20the%20robot%20base%20controllers%20to%20drive%20the%20robot)), you might have noticed that the robot moves a tiny distance and then stops abruptly after each command, which makes control rather jerky. This is because in order to keep driving, the controller needs to keep receiving commands, or it will assume that something has died and stop the robot.

In this tutorial, we will create a loop that sends velocity commands continuously, until the desired distance has been traveled. We will use the information about the distance traveled so far from the odometry.

0.1 Odometry and tf

Recall that in the previous tutorial we checked the base odometry controller along with the base velocity controller. However, we did not use odometry information in any way. We will in this tutorial; but the good news is that we don't have to bother talking to the odometry node directly. The odometry controller just needs to be running; it will then publish all its information into the transform tree which is accessible using the Transform Library tf (/tf).

To ensure that both required controllers are running, you can go back to the **The base velocity and odometry controller** step in the previous tutorial (/pr2_controllers/Tutorials/Using%20the%20robot%20base%20controllers%20to%20drive%20the%20robot).

Once we have made sure that the odometry controller is running, all that we need to do is ask tf (/tf) to tell us how the frame of the base of the robot has moved relative to a fixed odometry frame. One of the benefits is that the node we are writing here doesn't need to know where this information is coming from; all that matters is that the tf (/tf) transform tree has it.

0.1 Writing the node

We will start from the node you have created in the previous tutorial (Using the robot base controllers to drive the robot (/pr2_controllers/Tutorials/Using%20the%20robot%20base%20controllers%20to%20drive%20the%20robot)); however, since we will be using tf (/tf) we must add a dependency on the tf (/tf) package. Edit the `manifest.xml` file in your package and add the following dependency:

```
<depend package="tf" />
```

We will need to add some functionality to our `RobotDriver` class. More specifically, we are adding a function for **driving forward a certain distance**, based on odometry information. At the end of this tutorial you will also find a similar example for **turning**.

The key code snippets for achieving the desired functionality are:

Toggle line numbers

```
#include <tf/transform_listener.h>

```

We will be using information from tf (/tf) so we need to include the appropriate header.

```
  tf::TransformListener listener_;
```

Our `RobotDriver` class will use an instance of a tf (/tf) Listener to receive information from tf (/tf).

Toggle line numbers

```
  listener_.waitForTransform("base_footprint", "odom_combined",
                             ros::Time(0), ros::Duration(1.0));
```

The transform that we are interested in is the one from the `base_footprint` frame to the `odom_combined` frame. The odometry node will continuously update this information. Before we do anything, we have to wait for tf (/tf) to begin receiving this information about this transform from the odometry node, hence the `waitForTransform` command.

Toggle line numbers

```
//record the starting transform from the odometry to the base frame
listener_.lookupTransform("base_footprint", "odom_combined",
                          ros::Time(0), start_transform);
```

The main command for actually receiving the transform information from tf (/tf). We will use the same command inside our loop to see when we have travelled the specified distance.

Here is the complete class:

Toggle line numbers

```
#include <iostream>

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

class RobotDriver
{
private:
  //! The node handle we'll be using
  ros::NodeHandle nh_;
  //! We will be publishing to the "cmd_vel" topic to issue commands
  ros::Publisher cmd_vel_pub_;
  //! We will be listening to TF transforms as well
  tf::TransformListener listener_;

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>("cmd_vel", 1);
  }

  //! Drive forward a specified distance based on odometry information
  bool driveForwardOdom(double distance)
  {
    //wait for the listener to get the first message
    listener_.waitForTransform("base_footprint", "odom_combined",
                               ros::Time(0), ros::Duration(1.0));

    //we will record transforms here
    tf::StampedTransform start_transform;
    tf::StampedTransform current_transform;

    //record the starting transform from the odometry to the base frame
    listener_.lookupTransform("base_footprint", "odom_combined",
                              ros::Time(0), start_transform);

    //we will be sending commands of type "twist"
    geometry_msgs::Twist base_cmd;
    //the command will be to go forward at 0.25 m/s
    base_cmd.linear.y = base_cmd.angular.z = 0;
    base_cmd.linear.x = 0.25;

    ros::Rate rate(10.0);
    bool done = false;
    while (!done && nh_.ok())
    {
      //send the drive command
      cmd_vel_pub_.publish(base_cmd);
      rate.sleep();
      //get the current transform
      try
      {
        listener_.lookupTransform("base_footprint", "odom_combined",
                                  ros::Time(0), current_transform);
      }
      catch (tf::TransformException ex)
      {
        ROS_ERROR("%s",ex.what());
        break;
      }
      //see how far we've traveled
      tf::Transform relative_transform =
        start_transform.inverse() * current_transform;
      double dist_moved = relative_transform.getOrigin().length();

      if(dist_moved > distance) done = true;
    }
    if (done) return true;
    return false;
  }
};

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

  RobotDriver driver(nh);
  driver.driveForwardOdom(0.5);
}
```

0.1 Compiling and Running

Make sure the following line is in your **CMakeLists.txt** (it is the same from the previous tutorial):

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

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

To run, just as in the previous tutorial, bring up the robot (hardware or simulation), start the controllers (if you have just run the previous tutorial they should still be up and running), and run

```
bin/drive_base cmd_vel:=base_controller/command
```

You should see the robot move forward 0.5 meters.

0.1 Turning

We could just as easily add a function to our class that turns the robot a specified amount. Note that much of the code is similar to the function for driving forward.

Toggle line numbers

```
bool turnOdom(bool clockwise, double radians)
{
  while(radians < 0) radians += 2*M_PI;
  while(radians > 2*M_PI) radians -= 2*M_PI;

  //wait for the listener to get the first message
  listener_.waitForTransform("base_footprint", "odom_combined",
                             ros::Time(0), ros::Duration(1.0));

  //we will record transforms here
  tf::StampedTransform start_transform;
  tf::StampedTransform current_transform;

  //record the starting transform from the odometry to the base frame
  listener_.lookupTransform("base_footprint", "odom_combined",
                            ros::Time(0), start_transform);

  //we will be sending commands of type "twist"
  geometry_msgs::Twist base_cmd;
  //the command will be to turn at 0.75 rad/s
  base_cmd.linear.x = base_cmd.linear.y = 0.0;
  base_cmd.angular.z = 0.75;
  if (clockwise) base_cmd.angular.z = -base_cmd.angular.z;

  //the axis we want to be rotating by
  tf::Vector3 desired_turn_axis(0,0,1);
  if (!clockwise) desired_turn_axis = -desired_turn_axis;

  ros::Rate rate(10.0);
  bool done = false;
  while (!done && nh_.ok())
  {
    //send the drive command
    cmd_vel_pub_.publish(base_cmd);
    rate.sleep();
    //get the current transform
    try
    {
      listener_.lookupTransform("base_footprint", "odom_combined",
                                ros::Time(0), current_transform);
    }
    catch (tf::TransformException ex)
    {
      ROS_ERROR("%s",ex.what());
      break;
    }
    tf::Transform relative_transform =
      start_transform.inverse() * current_transform;
    tf::Vector3 actual_turn_axis =
      relative_transform.getRotation().getAxis();
    double angle_turned = relative_transform.getRotation().getAngle();
    if ( fabs(angle_turned) < 1.0e-2) continue;

    if ( actual_turn_axis.dot( desired_turn_axis ) < 0 )
      angle_turned = 2 * M_PI - angle_turned;

    if (angle_turned > radians) done = true;
  }
  if (done) return true;
  return false;
}
```

Wiki: pr2_controllers/T utorials/Using the base controller with odometry and transform information (last edited 2013-01-20 05:02:11 by Furushchev (/Furushchev))

Except where otherwise noted, the ROS wiki is licensed under the
Creative Commons Attribution 3.0 (http://creativecommons.org/licenses/by/3.0/) | Find us on Google+ (https://plus.google.com/113789706402978299308)

Brought to you by: Open Source Robotics Foundation

(http://www.osrfoundation.org)