Note: This tutorial assumes that you have completed the Transform Configuration (/navigation/Tutorials/RobotSetup/TF) tutorial. All the code for this tutorial is available in the odometry_publisher_tutorial (/odometry_publisher_tutorial) package..

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.

Publishing Odometry Information over ROS

Description: This tutorial provides an example of publishing odometry information for the navigation stack. It covers both publishing the nav_msgs/Odometry message over ROS, and a transform from a "odom" coordinate frame to a "base_link" coordinate frame over tf.

Tutorial Level: BEGINNER

1. Publishing Odometry Information Over ROS

The navigation stack uses tf (/tf) to determine the robot's location in the world and relate sensor data to a static map. However, tf does not provide any information about the velocity of the robot. Because of this, the navigation stack requires that any odometry source publish both a transform and a `nav_msgs/Odometry` message over ROS that contains velocity information. This tutorial explains the `nav_msgs/Odometry` message and provides example code for publishing both the message and transform over ROS and tf respectively.

Contents

2. The nav_msgs/Odometry Message

The `nav_msgs/Odometry` message stores an estimate of the position and velocity of a robot in free space:

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by
 header.frame_id.
# The twist in this message should be specified in the coordinate frame given by
 the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

The pose in this message corresponds to the estimated position of the robot in the odometric frame along with an optional covariance for the certainty of that pose estimate. The twist in this message corresponds to the robot's velocity in the child frame, normally the coordinate frame of the mobile base, along with an optional covariance for the certainty of that velocity estimate.

3. Using tf to Publish an Odometry transform

As discussed in the Transform Configuration (/navigation/Tutorials/RobotSetup/TF) tutorial, the "tf (/tf)" software library is responsible for managing the relationships between coordinate frames relevant to the robot in a transform tree. Therefore, any odometry source must publish information about the coordinate frame that it manages. The code below assumes a basic knowledge of tf, reading the Transform Configuration (/navigation/Tutorials/RobotSetup/TF) tutorial should be sufficient.

4. Writing the Code

In this section we'll write some example code for publishing a `nav_msgs/Odometry` message over ROS and a transform using tf (/tf) for a fake robot that just drives in a circle. We'll show the code in its entirety first, with a piece-by-piece explanation below.

Add the dependancy to your package's manifest.xml

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

Toggle line numbers

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>

int main(int argc, char** argv){
  ros::init(argc, argv, "odometry_publisher");

  ros::NodeHandle n;
  ros::Publisher odom_pub = n.advertise<nav_msgs::Odometry>("odom", 50);
  tf::TransformBroadcaster odom_broadcaster;

  double x = 0.0;
  double y = 0.0;
  double th = 0.0;

  double vx = 0.1;
  double vy = -0.1;
  double vth = 0.1;

  ros::Time current_time, last_time;
  current_time = ros::Time::now();
  last_time = ros::Time::now();

  ros::Rate r(1.0);
  while(n.ok()){

    ros::spinOnce();               // check for incoming messages
    current_time = ros::Time::now();

    //compute odometry in a typical way given the velocities of the robot
    double dt = (current_time - last_time).toSec();
    double delta_x = (vx * cos(th) - vy * sin(th)) * dt;
    double delta_y = (vx * sin(th) + vy * cos(th)) * dt;
    double delta_th = vth * dt;

    x += delta_x;
    y += delta_y;
    th += delta_th;

    //since all odometry is 6DOF we'll need a quaternion created from yaw
    geometry_msgs::Quaternion odom_quat =
tf::createQuaternionMsgFromYaw(th);

    //first, we'll publish the transform over tf
    geometry_msgs::TransformStamped odom_trans;
    odom_trans.header.stamp = current_time;
    odom_trans.header.frame_id = "odom";
    odom_trans.child_frame_id = "base_link";

    odom_trans.transform.translation.x = x;
    odom_trans.transform.translation.y = y;
    odom_trans.transform.translation.z = 0.0;
    odom_trans.transform.rotation = odom_quat;

    //send the transform
    odom_broadcaster.sendTransform(odom_trans);

    //next, we'll publish the odometry message over ROS
    nav_msgs::Odometry odom;
    odom.header.stamp = current_time;
    odom.header.frame_id = "odom";

    //set the position
    odom.pose.pose.position.x = x;
    odom.pose.pose.position.y = y;
    odom.pose.pose.position.z = 0.0;
    odom.pose.pose.orientation = odom_quat;

    //set the velocity
    odom.child_frame_id = "base_link";
    odom.twist.twist.linear.x = vx;
    odom.twist.twist.linear.y = vy;
    odom.twist.twist.angular.z = vth;

    //publish the message
    odom_pub.publish(odom);

    last_time = current_time;
    r.sleep();
  }
}
```

Ok, now that you've had a chance to look over everything, let's break down the important parts of the code in detail.

Toggle line numbers

```
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>

```

Since we're going to be publishing both a transfrom from the "odom" coordinate frame to the "base_link" coordinate frame and a `nav_msgs/Odometry` message, we need to include the relevant header files.

Toggle line numbers

```
  ros::Publisher odom_pub = n.advertise<nav_msgs::Odometry>("odom", 50);
  tf::TransformBroadcaster odom_broadcaster;
```

We need to create both a `ros::Publisher` and a `tf::TransformBroadcaster` to be able to send messages out using ROS and tf respectively.

Toggle line numbers

```
double x = 0.0;
double y = 0.0;
double th = 0.0;
```

We'll assume that the robot starts at the origin of the "odom" coordinate frame initially.

Toggle line numbers

```
double vx = 0.1;
double vy = -0.1;
double vth = 0.1;
```

Here we'll set some velocities that will cause the "base_link" frame to move in the "odom" frame at a rate of 0.1m/s in the x direction, -0.1m/s in the y direction, and 0.1rad/s in the th direction. This will more or less cause our fake robot to drive in a circle.

Toggle line numbers

```
ros::Rate r(1.0);
```

We'll publish odometry information at a rate of 1Hz in this example to make introspection easy, most systems will want to publish odometry at a much higher rate.

Toggle line numbers

```
  //compute odometry in a typical way given the velocities of the robot
  double dt = (current_time - last_time).toSec();
  double delta_x = (vx * cos(th) - vy * sin(th)) * dt;
  double delta_y = (vx * sin(th) + vy * cos(th)) * dt;
  double delta_th = vth * dt;

  x += delta_x;
  y += delta_y;
  th += delta_th;
```

Here we'll update our odometry information based on the constant velocities we set. A real odometry system would, of course, integrate computed velocities instead.

Toggle line numbers

```
  //since all odometry is 6DOF we'll need a quaternion created from yaw
  geometry_msgs::Quaternion odom_quat =
tf::createQuaternionMsgFromYaw(th);
```

We generally try to use 3D versions of all messages in our system to allow 2D and 3D components to work together when appropriate, and to keep the number of messages we have to create to a minimum. As such, it is necessary to convert our yaw value for odometry into a Quaternion to send over the wire. Fortunately, tf provides functions allowing easy creation of Quaternions from yaw values and easy access to yaw values from Quaternions.

Toggle line numbers

```
    //first, we'll publish the transform over tf
    geometry_msgs::TransformStamped odom_trans;
    odom_trans.header.stamp = current_time;
    odom_trans.header.frame_id = "odom";
    odom_trans.child_frame_id = "base_link";
```

Here we'll create a `TransformStamped` message that we will send out over tf. We want to publish the transform from the "odom" frame to the "base_link" frame at `current_time`. Therefore, we'll set the header of the message and the child_frame_id accordingly, making sure to use "odom" as the parent coordinate frame and "base_link" as the child coordinate frame.

Toggle line numbers

```
    odom_trans.transform.translation.x = x;
    odom_trans.transform.translation.y = y;
    odom_trans.transform.translation.z = 0.0;
    odom_trans.transform.rotation = odom_quat;

    //send the transform
    odom_broadcaster.sendTransform(odom_trans);
```

Here we fill in the transform message from our odometry data, and then send the transform using our `TransformBroadcaster`.

Toggle line numbers

```
    //next, we'll publish the odometry message over ROS
    nav_msgs::Odometry odom;
    odom.header.stamp = current_time;
    odom.header.frame_id = "odom";
```

We also need to publish a `nav_msgs/Odometry` message so that the navigation stack can get velocity information from it. We'll set the header of the message to the current_time and the "odom" coordinate frame.

Toggle line numbers

```
    //set the position
    odom.pose.pose.position.x = x;
    odom.pose.pose.position.y = y;
    odom.pose.pose.position.z = 0.0;
    odom.pose.pose.orientation = odom_quat;

    //set the velocity
    odom.child_frame_id = "base_link";
    odom.twist.twist.linear.x = vx;
    odom.twist.twist.linear.y = vy;
    odom.twist.twist.angular.z = vth;
```

This populates the message with odometry data and sends it out over the wire. We'll set the child_frame_id of the message to be the "base_link" frame since that's the coordinate frame we're sending our velocity information in.