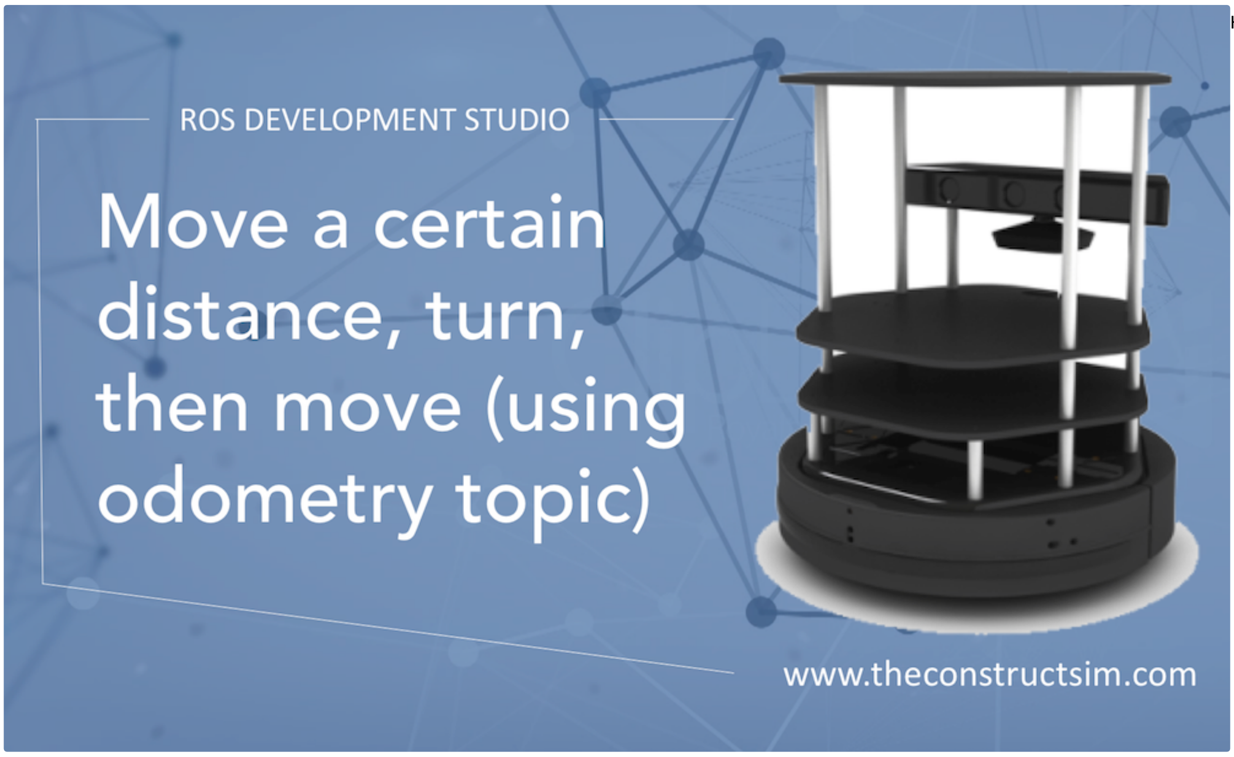
ROS Q&A | Move a certain distance, turn, then move (using odometry topic)

BY MARCO ARRUDA / MONDAY, 28 AUGUST 2017 / PUBLISHED IN ROS


ROS DEVELOPMENT STUDIO
Move a certain distance, turn, then move (using odometry topic)
www.theconstructsim.com

Hello Developers,

In a quick approach to make a robot move, we can start using some determined points or behaviors. In this post, we are going to test a simple algorithm that makes Turtlebot 2 performs a movement in a straight line, turn right and go straight again. In order to achieve a given point, we are going to use the Odometry, so the robot can localize itself while moves.

To do so, let's create a package able to interact with the topics **/odom** (*nav_msgs/Odometry*) and **/cmd_vel** (*geometry_msgs/Twist*). The following command can do that:

```
catkin_create_pkg turtlebot2_move roscpp geometry_msgs nav_msgs tf
```

The **tf** package was added as a dependency too because we will convert some data from **quaternion to RPY**.

Now, with the package created, let's take a look on the node programming:

```
#include <ros/ros.h>
#include <tf/tf.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/Pose2D.h>
#include <nav_msgs/Odometry.h>
```

```
#include <math.h>

geometry_msgs::Pose2D current_pose;
ros::Publisher pub_pose2d;

void odomCallback(const nav_msgs::OdometryConstPtr& msg)
{
    // linear position
    current_pose.x = msg->pose.pose.position.x;
    current_pose.y = msg->pose.pose.position.y;

    // quaternion to RPY conversion
    tf::Quaternion q(
        msg->pose.pose.orientation.x,
        msg->pose.pose.orientation.y,
        msg->pose.pose.orientation.z,
        msg->pose.pose.orientation.w);
    tf::Matrix3x3 m(q);
    double roll, pitch, yaw;
    m.getRPY(roll, pitch, yaw);

    // angular position
    current_pose.theta = yaw;
    pub_pose2d.publish(current_pose);
}

int main(int argc, char **argv)
{
    const double PI = 3.14159265358979323846;

    ROS_INFO("start");

    ros::init(argc, argv, "move_pub");
    ros::NodeHandle n;
    ros::Subscriber sub_odometry = n.subscribe("odom", 1, odomCallback);
    ros::Publisher movement_pub = n.advertise("cmd_vel",1); //for sensors the value after , should be higher to get a more accurate
result (queued)
    pub_pose2d = n.advertise("turtlebot_pose2d", 1);
    ros::Rate rate(10); //the larger the value, the "smoother" , try value of 1 to see "jerk" movement

    //move forward
    ROS_INFO("move forward");
    ros::Time start = ros::Time::now();
    while(ros::ok() && current_pose.x < 1.5) { geometry_msgs::Twist move; //velocity controls move.linear.x = 0.2; //speed value m/
s move.angular.z = 0; movement_pub.publish(move); ros::spinOnce(); rate.sleep(); } //turn right ROS_INFO("turn right"); ros::Time s
tart_turn = ros::Time::now(); while(ros::ok() && current_pose.theta > -PI/2)
    {
        geometry_msgs::Twist move;
        //velocity controls
        move.linear.x = 0; //speed value m/s
        move.angular.z = -0.3;
        movement_pub.publish(move);

        ros::spinOnce();
        rate.sleep();
    }
    //move forward again
    ROS_INFO("move forward");
    ros::Time start2 = ros::Time::now();
    while(ros::ok() && current_pose.y > -1.5)
    {
        geometry_msgs::Twist move;
        //velocity controls
        move.linear.x = 0.2; //speed value m/s
        move.angular.z = 0;
        movement_pub.publish(move);

        ros::spinOnce();
        rate.sleep();
    }

    // just stop
    while(ros::ok()) {
        geometry_msgs::Twist move;
        move.linear.x = 0;
        move.angular.z = 0;
        movement_pub.publish(move);

        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}
```

Basically, the code is creating a publisher and a subscriber objects, at the callback of the odometry information we are updating the knowledge of current pose the node has.

In the control loop, it's used this **current_pose** variable to compare with the goal.

There are 3 loops that are sending the robot velocity messages (go straight, turn right, go straight).

That's it!

It can't be considered a final solution, after all the starting position is not considered, we are considering the robot starts from a known point, so it can't be used to any case, but for a very specific one. Although, if we are working with a new robot and trying to move it, to make sure we can work and develop more stuff for it, this code is a good starting point.

This post was created as an answer to the following question in ROS Answers Forum: Move a certain distance, turn, then move (Odometry topic)

ABOUT MARCO ARRUDA

WHAT YOU CAN READ NEXT

ROS Q&A | DIFFERENT FRICTIONS FOR X AND Y AXIS

[ROS Q&A] – SERVICE GLOBAL NAME ERROR

ROS DEVELOPMENT STUDIO
How to have the RRT_Exploration package working in 5 minutes
www.theconstructsim.com

ROS Q&A | HOW TO HAVE THE RRT_EXPLORATION PACKAGE WORKING IN 5 MINUTES (OR LESS)

One Response to "ROS Q&A | Move a certain distance, turn, then move (using odometry topic)"

***Anonymous* says :**

25/10/2017 at 6:52 pm

How would I do this in python?

REPLY

Leave a Reply

Your email address will not be published.

Comment

Message:

Name **Email** **Website**

Name	Email	Website

Latest ROS Courses

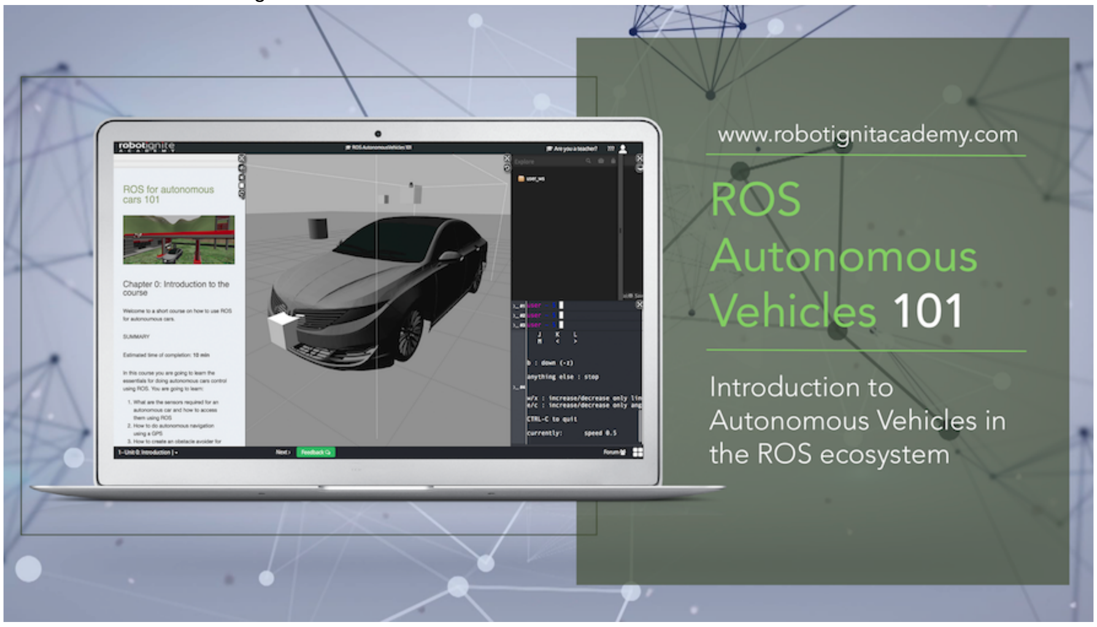
ROS Basics for Beginner

ROS IN 5 DAYS
Learn ROS in an effective way
ROS
LEARN ROS IN 5 DAYS
For Beginners
Yellow Level: Entering ROS in 5 days
www.robotigniteacademy.com

Mastering with ROS: Jackal

ROBOT IGNITE ACADEMY
www.robotigniteacademy.com
Mastering with ROS: Jackal
Learn how to program the Jackal robot for real world applications
CLEARPATH ROBOTICS™
robotignite ACADEMY

Learn ROS for Self-Driving Cars


www.robotignitacademy.com
ROS Autonomous Vehicles 101
Introduction to Autonomous Vehicles in the ROS ecosystem