!

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.

Go to Goal

Description: This tutorial is based on Turtlesim Video Tutorials (/turtlesim/Tutorials)

Tutorial Level: INTERMEDIATE

Contents

You can find the complete package at: https://github.com/clebercoutof/turtlesim_cleaner (https://github.com/clebercoutof/turtlesim_cleaner)

Now we are going to move the turtle to a specified location.

1. Preparing for work

Let's create our file `gotogoal.py` (or any name you want) and paste it in the source directory of our package: if you followed the past tutorial, it will be: `~/catkin_ws/src/turtlesim_cleaner/src`. Then, don't forget to make the node executable:

```
$ chmod u+x ~/catkin_ws/src/turtlesim_cleaner/src/gotogoal.py
```

2. Understanding the code

2.1 The TurtleBot Class

The class `TurtleBot` will contain all the aspects of our robot, such as pose, the publisher and subscriber, the subscriber callback function and the "move to goal" function.

2.2 The Subscriber

Our subscriber will subscribe to the topic `'/turtle1/pose'`, which is the topic to which the actual `turtlesim` position is published. The function `update_pose` is called when a message is received and saves the actual position in a class attribute called `pose`.

2.3 The euclidean_distance method

This method will use the previously saved turtle position (i.e. `self.pose`) and the argument (i.e. `data`) to calculate the point-to-point (Euclidean) distance between the turtle and the goal.

2.4 The PID Controller

In order for our robot to move, we will use a PID controller (https://en.wikipedia.org/wiki/PID_controller) for linear speed and angular velocity. The linear speed will consist of a constant multiplied by the distance between the turtle and the goal and the angular speed will depend on the arctangent of the distance in the y-axis by the distance in the x-axis multiplied by a constant. You can check the Go to Goal Turtlesim Video Tutorials (/turtlesim/Tutorials) for better explanation.

2.5 Tolerance

We have to create a tolerance zone around our goal point, since a really high precision to get exactly to the goal would be needed. In this code, if we use a really small precision the turtle would go crazy (you can have a try!). *In other words, the code and the simulator are simplified, so it won't work with full precision.*

3. The Code

Toggle line numbers

```
#!/usr/bin/env python
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
from math import pow, atan2, sqrt


class TurtleBot:

    def __init__(self):
        # Creates a node with name 'turtlebot_controller' and make sure it is a
        # unique node (using anonymous=True).
        rospy.init_node('turtlebot_controller', anonymous=True)

        # Publisher which will publish to the topic '/turtle1/cmd_vel'.
        self.velocity_publisher = rospy.Publisher('/turtle1/cmd_vel',
                                                  Twist, queue_size=10)

        # A subscriber to the topic '/turtle1/pose'. self.update_pose is called
        # when a message of type Pose is received.
        self.pose_subscriber = rospy.Subscriber('/turtle1/pose',
                                                Pose, self.update_pose)

        self.pose = Pose()
        self.rate = rospy.Rate(10)

    def update_pose(self, data):
        """Callback function which is called when a new message of type Pose is
        received by the subscriber."""
        self.pose = data
        self.pose.x = round(self.pose.x, 4)
        self.pose.y = round(self.pose.y, 4)

    def euclidean_distance(self, goal_pose):
        """Euclidean distance between current pose and the goal."""
        return sqrt(pow((goal_pose.x - self.pose.x), 2) +
                    pow((goal_pose.y - self.pose.y), 2))

    def linear_vel(self, goal_pose, constant=1.5):
        """See video: https://www.youtube.com/watch?v=Qh15Nol5htM."""
        return constant * self.euclidean_distance(goal_pose)

    def steering_angle(self, goal_pose):
        """See video: https://www.youtube.com/watch?v=Qh15Nol5htM."""
        return atan2(goal_pose.y - self.pose.y, goal_pose.x - self.pose.x)
```

```

    def angular_vel(self, goal_pose, constant=6):
        """See video: https://www.youtube.com/watch?v=Qh15Nol5htM."""
        return constant * (self.steering_angle(goal_pose) - self.pose.theta)

    def move2goal(self):
        """Moves the turtle to the goal."""
        goal_pose = Pose()

        # Get the input from the user.
        goal_pose.x = input("Set your x goal: ")
        goal_pose.y = input("Set your y goal: ")

        # Please, insert a number slightly greater than 0 (e.g. 0.01).
        distance_tolerance = input("Set your tolerance: ")

        vel_msg = Twist()

        while self.euclidean_distance(goal_pose) >= distance_tolerance:

            # Porportional controller.
            # https://en.wikipedia.org/wiki/Proportional_control

            # Linear velocity in the x-axis.
            vel_msg.linear.x = self.linear_vel(goal_pose)
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0

            # Angular velocity in the z-axis.
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = self.angular_vel(goal_pose)

            # Publishing our vel_msg
            self.velocity_publisher.publish(vel_msg)

            # Publish at the desired rate.
            self.rate.sleep()

        # Stopping our robot after the movement is over.
        vel_msg.linear.x = 0
        vel_msg.angular.z = 0
        self.velocity_publisher.publish(vel_msg)

        # If we press control + C, the node will stop.
        rospy.spin()

if __name__ == '__main__':
    try:
        x = TurtleBot()
```

```
        x.move2goal()
    except rospy.ROSInterruptException:
        pass
```

First, we import the libraries that will be needed. The `rospy` and `geometry_msgs` were discussed in the previous tutorials. The `math` library contains the function that will be used, such as `atan`, `sqrt` and `round`. The `turtlesim.msg` contains the `Pose` message type, which is the one published to the topic `'/turtle1/pose'`. You can check with the following command:

```
$ rostopic info /turtle1/pose
```

You should see the following screen:

```
~$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://birnuc1:45049/)

Subscribers: None
```

The pose message is composed by the x and y coordinates, the theta angle, linear velocity and angular velocity. You can see this info with the following command:

```
$ rosmsg show turtlesim/Pose
```

You should see the following screen:

```
~$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

Then we create our class. In the `__init__` method, we initiate the node, publisher, subscriber and the pose object. It's necessary to set a "publishing rate" in this case too:

Toggle line numbers

```
    def __init__(self):
        # Creates a node with name 'turtlebot_controller' and make sure it is a
        # unique node (using anonymous=True).
        rospy.init_node('turtlebot_controller', anonymous=True)

        # Publisher which will publish to the topic '/turtle1/cmd_vel'.
        self.velocity_publisher = rospy.Publisher('/turtle1/cmd_vel',
                                                  Twist, queue_size=10)

        # A subscriber to the topic '/turtle1/pose'. self.update_pose is called
        # when a message of type Pose is received.
        self.pose_subscriber = rospy.Subscriber('/turtle1/pose',
                                                Pose, self.update_pose)

        self.pose = Pose()
        self.rate = rospy.Rate(10)
```

The `update_pose` method is a callback function which will be used by the subscriber: it will get the turtle current pose and save it in the `self.pose` attribute:

Toggle line numbers

```
    def update_pose(self, data):
        """Callback function which is called when a new message of type Pose is
        received by the subscriber."""
        self.pose = data
        self.pose.x = round(self.pose.x, 4)
        self.pose.y = round(self.pose.y, 4)
```

The `euclidean_distance` method will be used to calculate the distance between the previously saved turtle position and the goal position:

Toggle line numbers

```
    def euclidean_distance(self, goal_pose):
        """Euclidean distance between current pose and the goal."""
        return sqrt(pow((goal_pose.x - self.pose.x), 2) +
                    pow((goal_pose.y - self.pose.y), 2))
```

The `move2goal` method will be the one which moves the turtle. First, we create the `goal_pose` object, which will receive the user's input, and it has the same type as the `self.pose` object. Then, we declare the `vel_msg` object, which will be published in `'/turtle1/cmd_vel'`:

Toggle line numbers

```
    def move2goal(self):
        """Moves the turtle to the goal."""
        goal_pose = Pose()

        # Get the input from the user.
        goal_pose.x = input("Set your x goal: ")
        goal_pose.y = input("Set your y goal: ")

        # Please, insert a number slightly greater than 0 (e.g. 0.01).
        distance_tolerance = input("Set your tolerance: ")

        vel_msg = Twist()
```

In the `while` loop, we will keep publishing until the distance of the turtle to the goal is less than the `distance_tolerance`:

Toggle line numbers

```
        while self.euclidean_distance(goal_pose) >= distance_tolerance:
```

At the end of the loop, we order the turtle to stop:

Toggle line numbers

```
        # Stopping our robot after the movement is over.
        vel_msg.linear.x = 0
        vel_msg.angular.z = 0
        self.velocity_publisher.publish(vel_msg)
```

The following statement causes the node to publish at the desired rate:

Toggle line numbers

```
            # Publish at the desired rate.
            self.rate.sleep()
```

The following statement guarantees that if we press *CTRL + C* our code will stop:

Toggle line numbers

```
        # If we press control + C, the node will stop.
        rospy.spin()
```

Finally, we call our function `move2goal` after having created an object `x` of type `TurtleBot`:

Toggle line numbers

```
if __name__ == '__main__':
    try:
        x = TurtleBot()
        x.move2goal()
    except rospy.ROSInterruptException:
        pass
```

4. Testing the code

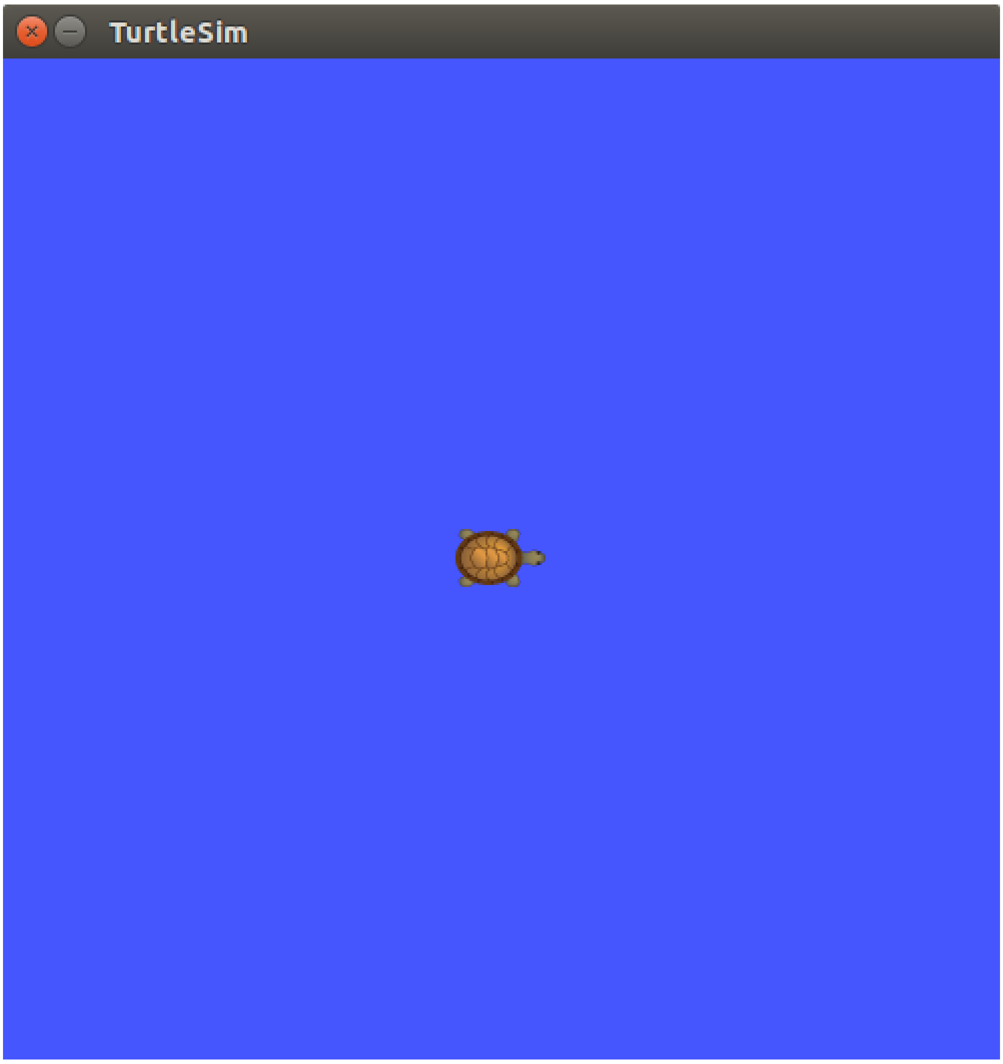
In a **new terminal**, run:

```
$ roscore
```

In a **new terminal**, run:

```
$ rosrun turtlesim turtlesim_node
```

The `turtlesim` window will open:



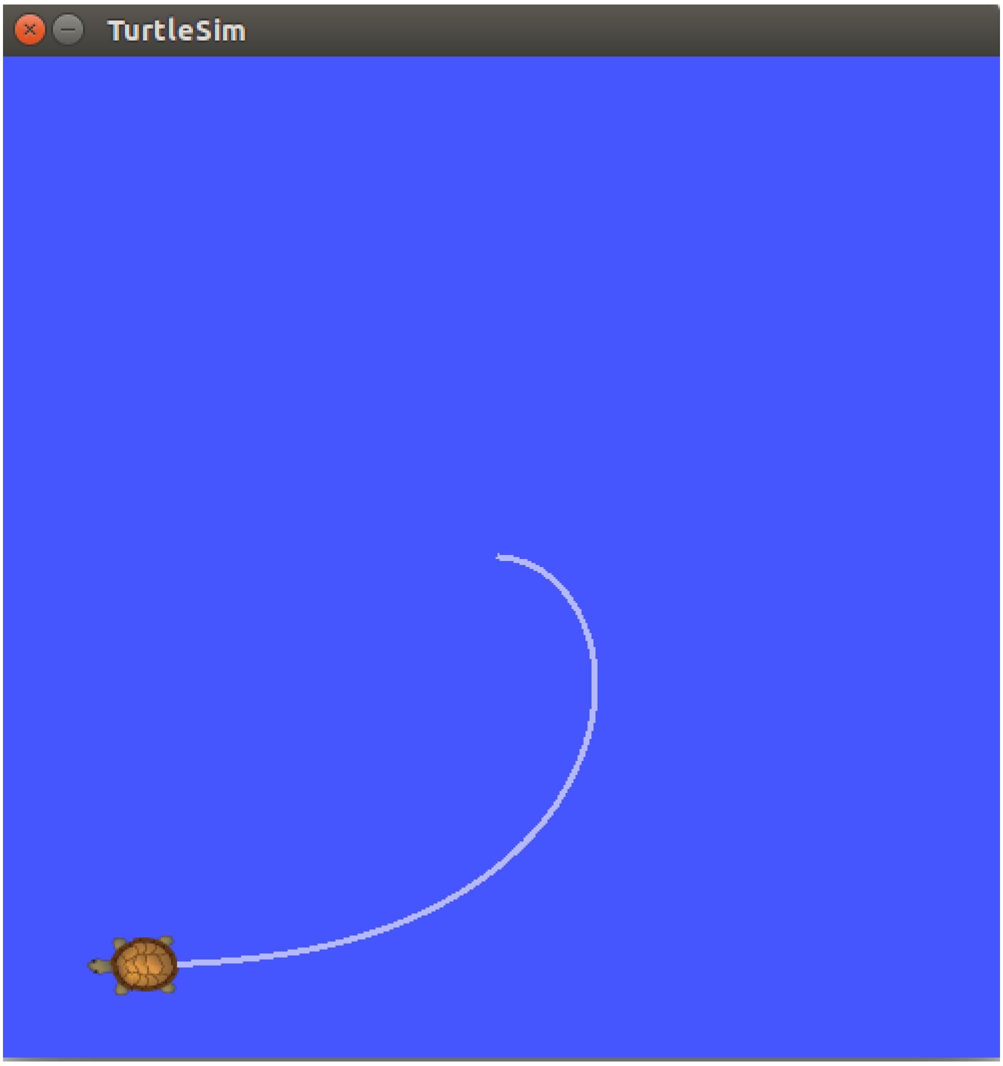
Now, in a **new terminal**, run our code:

```
$ rosrun turtlesim_cleaner gotogoal.py
```

Just type your inputs and the turtle will move! Here we have an example:

```
rosrun turtlesim_cleaner gotogoal.py
Set your x goal: 1
Set your y goal: 1
Set your tolerance: 0.5
```

The turtle will move like this:



Congratulations! You have finished the tutorials!