

Note: This tutorial assumes that you have completed the previous tutorials: Configuring and Using a Linux-Supported Joystick with ROS (/joy/Tutorials/ConfiguringALinuxJoystick).

💡 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.

Writing a Teleoperation Node for a Linux-Supported Joystick

Description: This tutorial covers how to write a teleoperation node and use it to drive the turtle in the turtlesim (/turtlesim).

Keywords: teleoperation, joystick

Tutorial Level: BEGINNER

Note: In Indigo, turtlesim was updated to use the defacto standard geometry_msgs/Twist (http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html) message on a cmd_vel topic. This tutorial requires an update to reflect this.

Contents

1. Create a Scratch Package
2. Writing a Simple Teleoperation Node
 1. The Code
 2. The Code Explained
3. Compiling and Running Turtle Teleop
4. Setting up the Joystick
5. Button and Axis mappings
6. Writing a Launch File to Start all of the Nodes

This tutorial is a getting started exercise. If you are simply looking for a generic teleop node to use with your Twist robot, consider teleop_twist_joy (/teleop_twist_joy).

0.1 Create a Scratch Package

Before starting this tutorial, take the time to create a scratch package to work in and manipulate the example code. See creating a ROS package (/ROS/Tutorials/CreatingPackage) to learn more about creating a package. Create a sandbox package with the following dependencies:

```
$ roscreate-pkg learning_joy roscpp turtlesim joy
```

Uncomment the genmsg() line in the CMakeLists.txt of the learning_joy package and run rosmake.

0.1 Writing a Simple Teleoperation Node

First, create `learning_joy/src/turtle_teleop_joy.cpp` in your favorite editor, and place the following code inside of it.

NOTE: If you installed joy as a diamondback stack, replace the references to "sensor_msgs" with "joy" throughout this code. See <http://answers.ros.org/question/10787/joy-node-build-bug-when-building-from-source> (<http://answers.ros.org/question/10787/joy-node-build-bug-when-building-from-source>) for more details.

NOTE: Example code using Python and ROS Hydro can be found here: <http://andrewdai.co/xbox-controller-ros.html#rosjoy> (<http://andrewdai.co/xbox-controller-ros.html#rosjoy>)

NOTE: In Hydro, turtlesim has deprecated its Velocity message and switched to twist. There is an ongoing pull request to resolve this. The changes needed can be seen at: https://github.com/ros-drivers/joystick_drivers_tutorials/pull/4/files (https://github.com/ros-drivers/joystick_drivers_tutorials/pull/4/files)

NOTE: In Hydro, these changes are needed to `teleop_turtle_joy.cpp`:

```
Replace: Velocity-->Twist
         Vel-->twist
         turtlesim-->geometry_msgs
         command_velocity-->cmd_vel
Add:     .x to the end of twist.linear
         .z to the end of twist.angular
```

0.0.1 The Code

https://raw.githubusercontent.com/ros-drivers/joystick_drivers_tutorials/master/turtle_teleop/src/teleop_turtle_joy.cpp
(https://raw.githubusercontent.com/ros-drivers/joystick_drivers_tutorials/master/turtle_teleop/src/teleop_turtle_joy.cpp)

Toggle line numbers

```
#include <ros/ros.h>
#include <turtlesim/Velocity.h>
#include <sensor_msgs/Joy.h>

class TeleopTurtle
{
public:
    TeleopTurtle();

private:
    void joyCallback(const sensor_msgs::Joy::ConstPtr& joy);

    ros::NodeHandle nh_;

    int linear_, angular_;
    double l_scale_, a_scale_;
    ros::Publisher vel_pub_;
    ros::Subscriber joy_sub_;

};

TeleopTurtle::TeleopTurtle():
    linear_(1),
    angular_(2)
{
    nh_.param("axis_linear", linear_, linear_);
    nh_.param("axis_angular", angular_, angular_);
    nh_.param("scale_angular", a_scale_, a_scale_);
    nh_.param("scale_linear", l_scale_, l_scale_);

    vel_pub_ = nh_.advertise<turtlesim::Velocity>("turtle1/command_velocity", 1);

    joy_sub_ = nh_.subscribe<sensor_msgs::Joy>("joy", 10, &TeleopTurtle::joyCallback, this);

}

void TeleopTurtle::joyCallback(const sensor_msgs::Joy::ConstPtr& joy)
{
    turtlesim::Velocity vel;
    vel.angular = a_scale_*joy->axes[angular_];
    vel.linear = l_scale_*joy->axes[linear_];
    vel_pub_.publish(vel);
}
```

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "teleop_turtle");
    TeleopTurtle teleop_turtle;

    ros::spin();
}
```

0.0.2 The Code Explained

Now, let's break down the code piece by piece.

Toggle line numbers

```
2 #include <ros/ros.h>
3 #include <turtlesim/Velocity.h>
4 #include <sensor_msgs/Joy.h>
5
```

- turtlesim/Velocity.h includes the turtle velocity msg so that we can publish velocity commands to the turtle
- joy/Joy.h includes the joystick msg so that we can listen to the joy topic

Toggle line numbers

```
7 class TeleopTurtle
8 {
9 public:
10     TeleopTurtle();
11
12 private:
13     void joyCallback(const sensor_msgs::Joy::ConstPtr& joy);
14
15     ros::NodeHandle nh_;
16
17     int linear_, angular_;
18     double l_scale_, a_scale_;
19     ros::Publisher vel_pub_;
20     ros::Subscriber joy_sub_;
21
22 };
```

Here we create the TeleopTurtle class and define the joyCallback function that will take a joy msg. We also create a node handle, publisher, and subscriber for later use.

Toggle line numbers

```

25 TeleopTurtle::TeleopTurtle():
26     linear_(1),
27     angular_(2)
28 {
29
30     nh_.param("axis_linear", linear_, linear_);
31     nh_.param("axis_angular", angular_, angular_);
32     nh_.param("scale_angular", a_scale_, a_scale_);
33     nh_.param("scale_linear", l_scale_, l_scale_);

```

Here we initialize some parameters: the `linear_` and `angular_` variables are used to define which axes of the joystick will control our turtle. We also check the parameter server for new scalar values for driving the turtle.

Toggle line numbers

```

36     vel_pub_ = nh_.advertise<turtlesim::Velocity>("turtle1/command_velocity", 1);

```

Here we create a publisher that will advertise on the `command_velocity` topic of the turtle.

Toggle line numbers

```

39     joy_sub_ = nh_.subscribe<sensor_msgs::Joy>("joy", 10,
    &TeleopTurtle::joyCallback, this);

```

Here we subscribe to the joystick topic for the input to drive the turtle. If our node is slow in processing incoming messages on the joystick topic, up to 10 messages will be buffered before any are lost.

Toggle line numbers

```

43 void TeleopTurtle::joyCallback(const sensor_msgs::Joy::ConstPtr& joy)
44 {
45     turtlesim::Velocity vel;
46     vel.angular = a_scale_*joy->axes[angular_];
47     vel.linear = l_scale_*joy->axes[linear_];
48     vel_pub_.publish(vel);
49 }

```

Here we take the data from the joystick and manipulate it by scaling it and using independent axes to control the linear and angular velocities of the turtle. Finally we publish the prepared message.

Toggle line numbers

```

52 int main(int argc, char** argv)
53 {
54     ros::init(argc, argv, "teleop_turtle");
55     TeleopTurtle teleop_turtle;
56
57     ros::spin();
58 }

```

Lastly we initialize our ROS node, create a teleop_turtle, and spin our node until Ctrl-C is pressed.

0.1 Compiling and Running Turtle Teleop

Add the following line to your CMakeLists.txt file:

```
roscpp_add_executable(turtle_teleop_joy src/turtle_teleop_joy.cpp)
```

NOTE: In Indigo, you will need to add:

```
add_executable(turtle_teleop_joy src/turtle_teleop_joy.cpp)
target_link_libraries(turtle_teleop_joy ${catkin_LIBRARIES})
```

and you will need to uncomment:

```
CATKIN_DEPENDS joy roscpp turtlesim
```

0.1 Setting up the Joystick

Before we can run the joystick and teleop together, we need to make the joystick accessible. Connect your joystick as we did in the previous tutorial ([/joy/Tutorials/ConfiguringALinuxJoystick](#)).

Now list the permissions of the joystick:

```
$ ls -l /dev/input/js0
```

You will see something similar to:

```
crw-rw-XX- 1 root dialout 188, 0 2009-08-14 12:04 /dev/input/js0
```

If XX is rw: the js device is configured properly.

If XX is --: the js device is not configured properly and you need to:

```
$ sudo chmod a+rw /dev/input/js0
```

0.1 Button and Axis mappings

In order to drive the turtle correctly, you'll need to configure the button and axis mappings of the joystick to the correct axes of the teleop node. If you're using a PS3 joystick, see the [PS3 Joystick \(/ps3joy\)](#) page for the axis mappings. If not, you'll have to check the button mappings manually.


To determine which button on the joystick publishes to each button in the ROS Joy message, view your joystick using

```
rostopic echo joy
```

after you launch the joystick. By pressing buttons and moving all the sticks, you'll be able to determine the correct mapping.

0.1 Writing a Launch File to Start all of the Nodes

Now let's make a launch file to start all of the nodes we need. Create the file `launch/turtle_joy.launch` and paste the following into it:

 https://raw.githubusercontent.com/ros-drivers/joystick_drivers_tutorials/master/turtle_teleop/launch/turtle_joy.launch
(https://raw.githubusercontent.com/ros-drivers/joystick_drivers_tutorials/master/turtle_teleop/launch/turtle_joy.launch)

Toggle line numbers

```
<launch>

<!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>

<!-- joy node -->
  <node respawn="true" pkg="joy"
    type="joy" name="turtle_joy" >
    <param name="dev" type="string" value="/dev/input/js0" />
    <param name="deadzone" value="0.12" />
  </node>

<!-- Axes -->
  <param name="axis_linear" value="1" type="int"/>
  <param name="axis_angular" value="0" type="int"/>
  <param name="scale_linear" value="2" type="double"/>
  <param name="scale_angular" value="2" type="double"/>

  <node pkg="turtle_teleop" type="turtle_teleop_joy" name="teleop"/>

</launch>
```

Since we built everything in package `learning_joy`, you may need to change the package in the last line from `turtle_teleop` to `learning_joy`.

NOTE: In Indigo, you will need to change

```
type="joy"-->type="joy_node"
pkg="turtle_teleop"-->pkg="learning_joy".
```

Now you can start the nodes to drive your turtle around:

```
roslaunch learning_joy turtle_joy.launch
```

CategoryHomepage (/CategoryHomepage)

Except where otherwise noted, the ROS wiki is licensed under Wiki: joy/Tutorials/WritingTeleopNode (last edited 2015-07-27 09:01:42 by FabioProietti (/FabioProietti))
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>)