



ROS

Part I

Workspace

Before you start writing any ROS code, you need to set up a workspace for this code to live in. A workspace is simply a set of directories in which a related set of ROS code lives.

You can have multiple ROS workspaces, but you can only work in one of them at any one time

Getting set up

We are going to create a workspace called teleop. Within mobile robot navigation, tele-operated refers to a robot under the control of a joystick or other controlled device. To create a workspace Open a Terminal and type in the following:

```
$ source /opt/ros/indigo/setup.bash
```

```
$ mkdir -p ~/teleop_ws/src
```

```
$ cd ~/teleop_ws/src
```

```
$ catkin_init_workspace
```

```
$ cd ~/teleop_ws
```

```
$ catkin_make
```

Running `catkin_make` will generate a lot of output as it does its work. When it's done, you'll end up with two new directories: `build` and `devel`.

```
ui70457@HML-26:~/Test_ws$ catkin_init_workspace
Creating symlink "/home/ui70457/Test_ws/src/CMakeLists.txt" pointing to "/opt/ro
s/indigo/share/catkin/cmake/toplevel.cmake"
ui70457@HML-26:~/Test_ws/src$ cd ~/Test_ws
ui70457@HML-26:~/Test_ws$ catkin_make
Base path: /home/ui70457/Test_ws
Source space: /home/ui70457/Test_ws/src
Build space: /home/ui70457/Test_ws/build
Devel space: /home/ui70457/Test_ws/devel
Install space: /home/ui70457/Test_ws/install

####
#### Running command: "cmake /home/ui70457/Test_ws/src -DCATKIN_DEVEL_PREFIX=/ho
me/ui70457/Test_ws/devel -DCMAKE_INSTALL_PREFIX=/home/ui70457/Test_ws/install -G
Unix Makefiles" in "/home/ui70457/Test_ws/build"
####
-- The C compiler identification is GNU 4.8.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Using CATKIN_DEVEL_PREFIX: /home/ui70457/Test_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/indigo
-- This workspace overlays: /opt/ros/indigo
-- Found PythonInterp: /usr/bin/python (found version "2.7.6")
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/ui70457/Test_ws/build/test_results
-- Looking for include file pthread.h
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Found Test sources under /usr/src/gtest: gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.6.18
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Build files have been written to: /home/ui70457/Test_ws/build
####
#### Running command: "make -j4 -l4" in "/home/ui70457/Test_ws/build"
####
ui70457@HML-26:~/Test_ws$
```



ROS

Package

ROS software is organized into packages, each of which contains some combination of code, data, and documentation. Packages sit inside workspaces, in the src directory.

Each package directory must include a CmakeLists.txt file and a package.xml file that describes the contents of the package and how catkin should interact with it.

Getting set up

To create a package is easy once we are in the src directory of our workspace.

The first argument, Teleop, is the name of the new package we want to create. The following arguments are the names of packages that the new package depends on.

```
$ source devel/setup.bash
```

```
$ cd ~/teleop_ws/src
```

```
$ catkin_create_pkg teleop rospy geometry_msgs sensor_msgs
```

Part II

Lets Get Started

We are now going show how we can implement a number of features in ROS . All of these features will culminate with a sophisticated two dimensional navigation system that can display video of its surroundings as well as map out its environment. You are not expected to fully understand whats happening this serves as more of an introduction into what ROS can do.

We are going to be using a lot of different Terminals in this demo, so remember Ctrl-Alt-t is a shortcut to open a new Terminal.



ROS



Roscore and Simulation

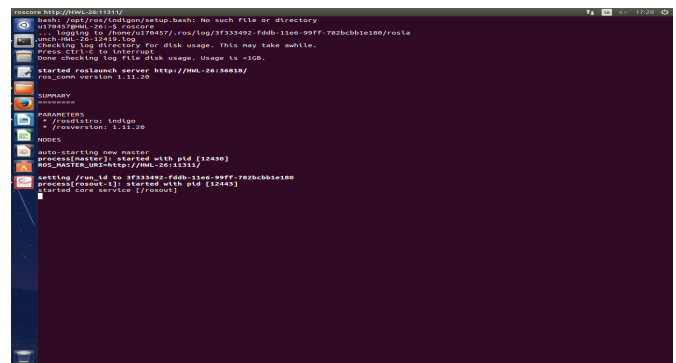
First thing we are going to do is launch Roscore. You'll remember that roscore is the network that all the nodes communicate on and it is how we will send and receive data from our robot

Since we don't all have a mobile robot such as Turtlebot to ourselves we are going to simulate one in Gazebo

Open New Terminal T1

\$ roscore

// "Minimize Terminal"

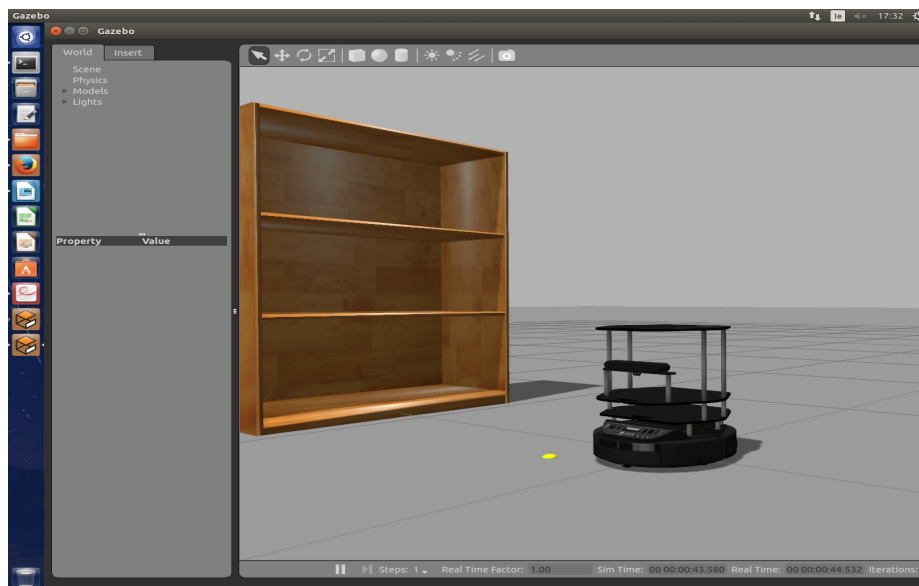


```
bash: /opt/ros/indigo/setup.bash: No such file or directory
vstool@vm: /$ roscore
... logging to /home/vstool/.ros/log/5f333492-fddb-11e0-99ff-782bcbb1c106/rosc
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is 100%.
started roslaunch server http://vm:203618/
ros_comm version 1.11.20
Summary
=====
PARAMETERS
 * /roscpp: indigo
 * /rosversion: 1.11.20
Notes
-----
auto-starting new master
process[master]: started with pid [12450]
ROS_MASTER_URI=http://vm:203618/
setting /run_id to 5f333492-fddb-11e0-99ff-782bcbb1c106
process[roscpp]: started with pid [12451]
started core service [/roscpp]
```

Open New Terminal T2

\$ roslaunch turtlebot_gazebo turtlebot_world.launch

// "Minimize Terminal"



This will open up a pre-set simulation of a turtlebot environment that we can pass all of our messages to over Roscore

Take some time to familiarize your self with the camera controls



Program and Nodes

To get the robot we need to create and program two nodes.

1. A program that will publish our keyboard key strokes and broadcast them over Roscore
2. A program that will subscribe to the first programs message and apply the correct motion to it

In simple terms one Program to say *“I pressed the arrow up key ”* and one program to say *“the up arrow key means go forward”*

Save the following program as `key_publisher.py` in the file we created `teleop_ws/`

```
#!/usr/bin/env python
import sys, select, tty, termios
import rospy
from std_msgs.msg import String

if __name__ == '__main__':
    key_pub = rospy.Publisher('keys', String, queue_size=1)
    rospy.init_node("keyboard_driver")
    rate = rospy.Rate(100)

    old_attr = termios.tcgetattr(sys.stdin)
    tty.setcbreak(sys.stdin.fileno())

    print    "Publishing keystrokes. Press Ctrl-C to exit..."

    while    not    rospy.is_shutdown():
        if select.select([sys.stdin], [], [], 0)[0] == [sys.stdin]:
            key_pub.publish(sys.stdin.read(1))
            rate.sleep()
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_attr)
```

This program uses the `termios` library to capture raw keystrokes, as soon as they are pressed. However it dose not support exteneded keys such as arrows keys ,these will result in `std_msgs/String` messages that are either weird symbols or multiple messages.



Save the following as `keys_to_twist_with_ramps.py` in the file we created `teleop_ws/`

```
#!/usr/bin/env python
import rospy
import math
from std_msgs.msg import String
from geometry_msgs.msg import Twist

key_mapping = { 'w':[ 0,1], 'x':[ 0, -1], 'a':[ 1, 0], 'd':[ -1, 0], 's':[ 0, 0] }
g_twist_pub = None
g_target_twist = None
g_last_twist = None
g_last_send_time = None
g_vel_scales = [0.1, 0.1] # default to very slow
g_vel_ramps = [1, 1] # units: meters per second^2

def ramped_vel(v_prev, v_target, t_prev, t_now, ramp_rate):
    # compute maximum velocity step
    step = ramp_rate*(t_now - t_prev).to_sec()
    sign = 1.0 if (v_target > v_prev) else -1.0
    error = math.fabs(v_target - v_prev)
    if error < step: # we can get there within this timestep-we're done.
        return v_target
    else:
        return v_prev + sign*step # take a step toward the target

def ramped_twist(prev, target, t_prev, t_now, ramps):
    tw = Twist()
    tw.angular.z = ramped_vel(prev.angular.z, target.angular.z, t_prev, t_now, ramps[0])
    tw.linear.x = ramped_vel(prev.linear.x, target.linear.x, t_prev, t_now, ramps[1])
    return tw

def send_twist():
    global g_last_twist_send_time, g_target_twist, g_last_twist, g_vel_scales, g_vel_ramps, g_twist_pub
    t_now = rospy.Time.now()
    g_last_twist = ramped_twist(g_last_twist, g_target_twist, g_last_twist_send_time, t_now, g_vel_ramps)
    g_last_twist_send_time = t_now
    g_twist_pub.publish(g_last_twist)

def keys_cb(msg):
    global g_target_twist, g_last_twist, g_vel_scales
    if len(msg.data) == 0 or not key_mapping.has_key(msg.data[0]):
        return # unknown key
    vels = key_mapping[msg.data[0]]
    g_target_twist.angular.z = vels[0]*g_vel_scales[0]
    g_target_twist.linear.x = vels[1]*g_vel_scales[1]

def fetch_param(name, default):
    if rospy.has_param(name):
        return rospy.get_param(name)
    else:
        print "parameter [%s] not defined. Defaulting to %.3f" % (name, default)
        return default

if __name__ == '__main__':
    rospy.init_node('keys_to_twist')
    g_last_twist_send_time = rospy.Time.now()
    g_twist_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.Subscriber('keys', String, keys_cb)
    g_target_twist = Twist() # initializes to zero
    g_last_twist = Twist()
    g_vel_scales[0] = fetch_param('~angular_scale', 0.5)
    g_vel_scales[1] = fetch_param('~linear_scale', 0.5)
    g_vel_ramps[0] = fetch_param('~angular_accel', 1.0)
    g_vel_ramps[1] = fetch_param('~linear_accel', 1.0)
    rate = rospy.Rate(20)
    while not rospy.is_shutdown():
        send_twist()
        rate.sleep()
```



Lets get moving

Open New Terminal T3

```
$ cd teleopbot_ws  
$ ./key_publisher.py
```

```
///  
// "Right Click on the top of this terminal and click Always on top "  
// "This terminal is important and we will be coming back to this "
```

Open New Terminal T4

```
$ cd teleopbot_ws  
$ ./keys_to_twist_with_ramps.py cmd_vel:=cmd_vel_mux/input/teleop
```

```
///  
// " Minimize"
```

Open Terminal 3 again

```
///  
// " Using w,x the robot will move forward and back"  
// " Using a,d the robot will rotate"  
// " Using s the robot will stop"
```