Robot Vision Systems

Lecture 14: ROS Nodes in Python

Michael Felsberg

michael.felsberg@liu.se

Bad News

- ROS Jade supports OpenCV3, but NOT Python3
  - –at some point, there is a warning that cv_bridge has been built with 2.4 (!), more on this later
- You have to use Python2.7
  - –you could try to install rospkg via pip3, but several manual steps have to be done on top
  - –do NOT install python3-rospkg via apt-get, this will basically uninstall ROS
- Recommendation: use ‘future’ trick from previous lecture

Publisher Node in Python

- “Talker”, continually broadcasting a message
- part of your package, e.g.
  `$ roscd beginner_tutorials`
- code is located in ‘scripts’
- type or download (+ make executable)
  `$ wget https://raw.github.com/ros/ros_tutorials/jade-devel/rospy_tutorials/001_talker_listener/talker.py`

Publisher Node in Python

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

def talker():
    pub = rospy.Publisher('chatter', String,
                          queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
```

Publisher Node in Python

```
    hello_str = “T %s" % rospy.get_time()
    rospy.loginfo(hello_str)
    pub.publish(hello_str)
    rate.sleep()
```

```
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Details on Specific Lines

- `queue_size`: limits the number of messages if *any* subscriber is not fast enough
- Why `anonymous`? If two nodes with the same name are launched, the first is kicked off
- `loginfo` goes to
  - –screen
  - –log file
  - –rosout
- `rate.sleep()` waits according to rate

Subscriber Node in Python

- type or download (and make executable)
  `$ wget https://raw.github.com/ros/ros_tutorials/jade-devel/rospy_tutorials/001_talker_listener/listener.py`

Subscriber Node in Python

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

def callback(data):
    rospy.loginfo(rospy.get_caller_id() +
                  "I heard %s", data.data)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("chatter", String, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

Details on Specific Lines

- `callback` is invoked with the message as first argument
- Callback functions have their own threads
- `rospy.spin()` keep node from exiting, no influence on callbacks

Testing the Nodes

- run in `catkin_ws` (suggestion: add first line to `.bashrc`)

```
$ source ./devel/setup.bash
$ catkin_make
$ roscore
$ rosrun beginner_tutorials
talker.py
$ rosrun beginner_tutorials
listener.py
```

Less Trivial Example

- Remember “image hello world” from OpenCV part: split into two nodes
  - “imtalker” acquiring images
  - “imlistener” showing images
  - optional: either of both can be replaced with ROS standard tools
- Working USB-cam required
  - VirtualBox: tick under “Devices/Webcams”
  - you might have to install ros-jade-usb-cam

Image Publisher (OpenCV)

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
```

```
def talker():
    pub = rospy.Publisher('imager', Image,
                          queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(2) # 2hz
    capture = cv2.VideoCapture(0)
    br = CvBridge()
    while not rospy.is_shutdown():
```

Image Publisher (OpenCV)

```
        [status,img] = capture.retrieve()
        if status == True:
            rospy.loginfo('publish image')
            pub.publish(br.cv2_to_imgmsg(
                                        img))
            rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass):
```

Image Subscriber (OpenCV)

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
```

```
def callback(data):
    br = CvBridge()
    rospy.loginfo('receiving image')
    cv2.imshow("camera",
                    br.imgmsg_to_cv2(data))
    cv2.waitKey(1)
```

Image Subscriber (OpenCV)

```
def listener():
    rospy.init_node('listener',
                    anonymous=True)
    rospy.Subscriber('imager',
                     Image, callback)
    rospy.spin()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    listener()
```

Image Publisher (ROS)

- Generate launch-file (see next slide)

```
$ rosed beginner_tutorials
usb_cam.launch
```

- Call roslaunch

```
$ roslaunch beginner_tutorials
usb_cam.launch
```

Launch File

```
<launch>
  <node name="usb_cam" pkg="usb_cam"
type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="mjpeg" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
    <remap from="/usb_cam/image_raw"
to="/camera/image_raw" />
  </node>
</launch>
```

Image Subscriber (ROS)

- Run viewer node

```
$ rosrun image_view image_view
image:=/camera/image_raw
```

- Note: R-B swap compared to OpenCV

```
cv2.imshow("camera",cv2.cvtColor(
        br.imgmsg_to_cv2(data),
        cv2.COLOR_BGR2RGB))
```

- Note2: use different topic name

```
rospy.Subscriber('/camera/image_raw',
                 Image, callback)
```

Pitfalls

- ROS comes with OpenCV without GTK2 support (for good reasons)
- OpenCV `highgui` stuff will fail
- Identify which `cv2.so` is used:

```
rospy.loginfo(cv2.__file__)
```

- Work around:

```
import sys
sys.path.remove('/opt/ros/jade/lib/
python2.7/dist-packages')
import cv2
```

Services

- Simple example: adding two numbers
- Makes use of `AddTwoInts.srv` (lecture 10)
- Code placed in `scripts/`
- Scripts must be excutable (`$ chmod +x`)
- Two scripts:
  - server `add_two_ints_server.py`
  - client `add_two_ints_client.py`

Defining Services

- By srv file in the srv directory
- A request and a response part, divided by `triple dash (---)`

```
int64 A
int64 B
---
int64 Sum
```

- Example: copy

```
$ roscp rospy_tutorials
AddTwoInts.srv srv/AddTwoInts.srv
```

Code Generation

- Despite name: message_generation for msg and srv
- Use same additional dependencies `CATKIN_DEPENDS, generate_messages()`
- Add explicitly service file

```
add_service_files(
  FILES
  AddTwoInts.srv
)
```

- Check service

```
$ rossrv show
[beginner_tutorials/]AddTwoInts
```

Server

```
#!/usr/bin/env python
from beginner_tutorials.srv import *
import rospy

def handle_add_two_ints(req):
    print "Returning [%s + %s =
           %s]"%(req.a, req.b,
           (req.a + req.b))
    return AddTwoIntsResponse(req.a +
           req.b)
```

Server

```
def add_two_ints_server():
    rospy.init_node(
                'add_two_ints_server')
    s = rospy.Service('add_two_ints',
      AddTwoInts, handle_add_two_ints)
    print "Ready to add two ints."
    rospy.spin()

if __name__ == "__main__":
    add_two_ints_server()
```

Details on Specific Lines

- `rospy.Service()` declares new service
  - –named `add_two_ints`
  - –with `AddTwoInts` service type
- Requests are passed to `handle_add_two_ints`
  - –called with instances of `AddTwoIntsRequest`
  - –returns instances of `AddTwoIntsRespons`

Client

```
#!/usr/bin/env python
import sys
import rospy
from beginner_tutorials.srv import *
```

```
def add_two_ints_client(x, y):
    rospy.wait_for_service(
                        'add_two_ints')
    try:
        add_two_ints =
                rospy.ServiceProxy(
                'add_two_ints',AddTwoInts)
```

Client

```
        resp1 = add_two_ints(x, y)
        return resp1.sum
    except rospy.ServiceException, e:
        print "Service call failed:
                                %s"%e

def usage():
    return "%s [x y]"%sys.argv[0]

if __name__ == "__main__":
    if len(sys.argv) == 3:
```

Client

```
        x = int(sys.argv[1])
        y = int(sys.argv[2])
else:
        print usage()
        sys.exit(1)
print "Requesting %s+%s"%(x, y)
print "%s + %s = %s"%(x, y,
       add_two_ints_client(x, y))
```

Details on Specific Lines

- `rospy.wait_for_service()` blocks until the service named `add_two_ints` is available
- `rospy.ServiceProxy()` creates a handle for calling the service
- declaration by service type
  - `Request` object automatically generated
  - `Response` object returned
  - failure raises `rospy.ServiceException`

Test the Service

- Autogenerate code for messages and services
  `$ catkin_make` (in the workspace)
- Run the two scripts

```
$ rosrun beginner_tutorials
        add_two_ints_server.py
$ rosrun beginner_tutorials
        add_two_ints_client.py 2 6
```