Introduction to ROS: Basics, Motion, and Vision

Geesara Kulathunga

ROS Installation

1. Option 01: Linux-based users http://wiki.ros.org/melodic/Installation/Ubuntu
2. Option 02: Non Linux-based users. First, you need to install vmware or virtualbox. Second, install Linux-based operating system, e.g., Ubuntu 18.x or Ubuntu 20.x
3. Install docker https://docs.docker.com/engine/install/ubuntu/, and set the user permission https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04
4. Clone the repository https://github.com/GPrathap/ros_intro.git

```
cd <some path>/intro_ros/section00
./run_ros_node_[gpu|cpu].sh run # run the container
./run_ros_node_[gpu|cpu].sh start # start the container
./run_ros_node_[gpu|cpu].sh enter # enter the container
```

ROS Installation

1. Ubuntu 18.0x http://wiki.ros.org/melodic/Installation/Ubuntu
2. Ubuntu 20.0x http://wiki.ros.org/noetic/Installation/Ubuntu

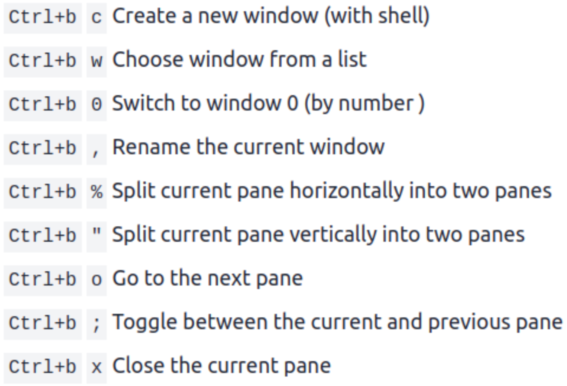
ROS Installation

1. Install building tool (https://catkin-tools.readthedocs.io/en/latest/installing.html)

```
sudo apt-get install python3-catkin-tools
```

Working with tmux

tmux, a program that runs in a terminal. It allows multiple other terminal programs to be run inside it. To install tmux: **sudo apt install tmux**



`Ctrl+b`	`c`	Create a new window (with shell)
`Ctrl+b`	`w`	Choose window from a list
`Ctrl+b`	`0`	Switch to window 0 (by number)
`Ctrl+b`	`,`	Rename the current window
`Ctrl+b`	`%`	Split current pane horizontally into two panes
`Ctrl+b`	`"`	Split current pane vertically into two panes
`Ctrl+b`	`o`	Go to the next pane
`Ctrl+b`	`;`	Toggle between the current and previous pane
`Ctrl+b`	`x`	Close the current pane

ROS Workspace

1. Default workspace is located at /some_path/ros/ros_version/setup.bash

You can create ros workspace in a location you prefer

```
mkdir -p  /catkin_ws/src
cd  /catkin_ws
catkin build
cd ./devel && pwd
echo source ‘pwd‘/setup.bash » ~/.bashrc
source devel/setup.bash
echo $ROS_PACKAGE_PATH$
```

ROS Build System

1. catkin build or catkin_make (older variant) is used to build the the ros packages and generate executable, libraries, and interfaces

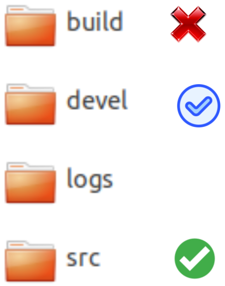
to navigate to workspace

```
>cd  /catkin_ws
```

to build your package

```
>catkin build package_name
Note:whenever package is built, it is required to
>source ./devel/setup.bash
```

ROS Build System



to see catkin workspace

```
>catkin config
```

Example

Hello world!

```
> cd  /catkin_ws/src
> git clone https://github.com/GPrathap/ros_intro.git
> cd ../ && catkin build hello_friend
> source devel/setup.bash
> roslaunch hello_friend pub_sub.launch
```

ROS Master (roscore)

Figure: https://www.youtube.com/watch?v=NmidmSS9Ylk

ROS Master (roscore)

1. The centralized controller or manager
2. Register nodes (sub-programs) when starts with the master
3. Handle communication between nodes (sub-programs (nodes))
4. Also, provide the Parameter server, which is shared among the Nodes that is used to retrieve parameters
5. rosout, which is /rosout, logging purpose
6. roscore = master + parameter_server + rosout

to start the master

```
>roscore
```

ROS Nodes

Figure: https://www.youtube.com/watch?v=NmidmSS9Ylk

ROS Nodes (Processors/pub-programs)

1. Do you know the different between threads and processors
2. Each nodes executes as a processor
3. Node APIs: roscpp, rospy

to run a node

```
>rosrun package_name node_name
```

to see active nodes

```
>rosnode list
```

to get information about a node

```
>rosnode info node_name
```

ROS Nodelets

1. Conceptually node and nodelets are same
2. These are designed to reduce to overhead, i.e., without copying the data, when running on the same machine
3. Quite complicated to implement

ROS Topics

Figure: https://www.youtube.com/watch?v=NmidmSS9YIk

ROS Topics

1. Topics can be used to communicate among the nodes
2. Nodes can publish, subscribe or both, typically 1 to n connection exist between a publisher and subscribers

to see active topics list

```
>rostopic list
```

to subscribe to a topic

```
>rostopic echo /topic
```

to get information about a topic

```
>rosnode info topic_name
```

ROS Launch

File Structure (ros_example_hello.launch)

```
<launch>
<node name ="friend_hello_pub" pkg="hello_friend" type="friend_hello_pub"/>
</launch>
```

1. **launch**: is the root element of launch file
2. **node**: launch file can be comprised with several nodes, each of which describes node information to provided before launching
3. **name**: as user wish
4. **pkg**: which package that the considered node belongs
5. **type**: it must be same as name
6. **output**: where to log the output: console or log file

ROS Launch

File Structure (pub_sub.launch)

```
<launch>
<node name ="friend_hello_pub" pkg="hello_friend" type="friend_hello_pub"/>
</launch>
```

1. **launch**: is the root element of launch file
2. **node**: launch file can be comprised with several nodes, each of which describes node information to provided before launching
3. **name**: as user wish
4. **pkg**: which package that the considered node belongs
5. **type**: it must be same as name
6. **output**: where to log the output: console or log file

ROS Package

1. **launch folder**: contains launch files each of which may have defined multiple nodes or includes another multiple launch files
2. **src folder**: source files
3. **package.xml**: or manifest file, contains the package meta data
4. **CMakeLists.txt**: dependencies, executable, and exporting all meta information

ROS Package Creation

dummy package with several dependencies

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

```
> catkin_create_pkg first_package std_msgs rospy roscpp
> source devel/setup.bash
```

ROS Package's Package.xml

1. **name**: name of the package
2. **version**: it should be defined with three integers separated by dots
3. **description**: objective of the package
4. **buildtool_depend**: dependencies that are required for the build tool
5. **build_depend**: dependencies of the package
6. **build_export_depend**: dependencies that are included in the headers
7. **exec_depend**: dependencies of shared libraries

ROS Messages

ROS Messages

ROS Messages

1. Message contains information to be transformed
2. Typically comprises of a nested structure of primitive data types, e.g., integer, double, float, boolean, and string.
3. Define as *.msg

to see type of a topic

```
>rostopic type /topic
```

to publish message over a topic

```
>rostopic pub /topic type <message>
```

ROS Messages

Odometry message example

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Header message example

```
uint32 seq
time stamp
string frame_id
```

More info: http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html

ROS Message Create

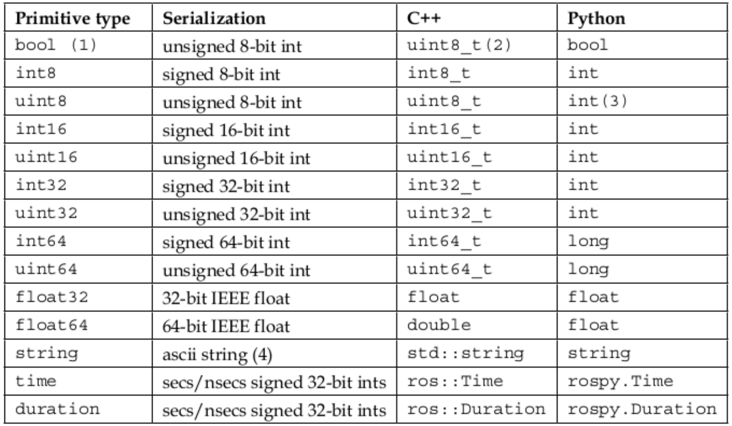
Friend’s message

```
mkdir -p catkin_ws/src/hello_friend/msg
cd catkin_ws/src/hello_friend/msg
touch friend_info.msg
```

Friend’s message content

```
string name
string id
```

ROS Message: Standard Types



Primitive type	Serialization	C++	Python
`bool (1)`	unsigned 8-bit int	`uint8_t(2)`	`bool`
`int8`	signed 8-bit int	`int8_t`	`int`
`uint8`	unsigned 8-bit int	`uint8_t`	`int(3)`
`int16`	signed 16-bit int	`int16_t`	`int`
`uint16`	unsigned 16-bit int	`uint16_t`	`int`
`int32`	signed 32-bit int	`int32_t`	`int`
`uint32`	unsigned 32-bit int	`uint32_t`	`int`
`int64`	signed 64-bit int	`int64_t`	`long`
`uint64`	unsigned 64-bit int	`uint64_t`	`long`
`float32`	32-bit IEEE float	`float`	`float`
`float64`	64-bit IEEE float	`double`	`float`
`string`	ascii string (4)	`std::string`	`string`
`time`	secs/nsecs signed 32-bit ints	`ros::Time`	`rospy.Time`
`duration`	secs/nsecs signed 32-bit ints	`ros::Duration`	`rospy.Duration`

ROS Message Create Cont.

Package Dependencies

buildtool_depend: catkin
build_depend: roscpp
rospy
std_msgs
message_generation
build_export_depend: roscpp
rospy
exec_depend: roscpp rospy
std_msgs
message_runtime

ROS Message Create Cont.

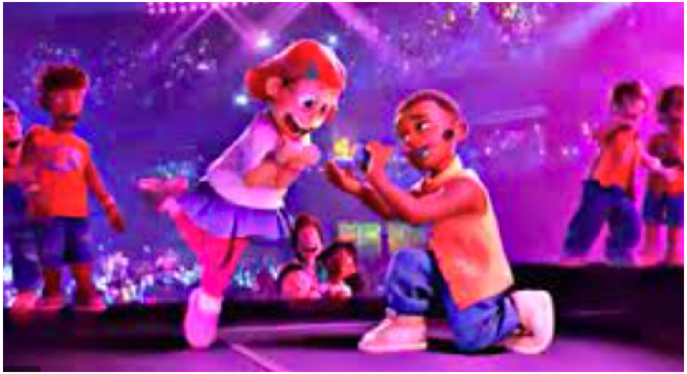
to find dependencies

```
find_package(catkin REQUIRED COMPONENTS roscpp
rospy
std_msgs
message_generation
)
catkin_package(
CATKIN_DEPENDS roscpp rospy std_msgs message_runtime
)
```

to generate messages

```
add_message_files(FILES friend_info.msg)
```

ROS Service



ROS Service

1. Peer-to-peer
2. Execute sequentially, i.e, request and then have to wait till the response
3. *.srv is the file type that defines the service that has two parts: a request and a response.When creating a service, request and response is separated by "—", which is given in the next slide
4. Similar analogy how topic works, yet services are two way transports. A service does one-to-one communication, topic does many to many communication

ROS Service Create

create a service

```
mkdir -p catkin_ws/src/first_package/srv
cd catkin_ws/src/first_package/srv
touch friend_info.srv
```

service

```
string name
string id
—
string heartbeat
```

ROS Service Create Cont.

to generate a service (define in CMakeLists.txt)

```
add_service_files(
FILES
friend_info.srv
)
```

ROS Service

to call a service

```
roslaunch hello_friend server_client.launch
rosservice call service_name message
```

ROS Publisher

ROS Publisher Create

```
import rospy
from std_msgs.msg import String
def send_hello():
    rospy.init_node('send_hello_node', anonymous=True)
    pub = rospy.Publisher('/send_hello_topic'
        , String, queue_size=10)
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
        hello_msg = "hello␣frield␣%s" % rospy.get_time()
        rospy.loginfo(hello_msg)
        pub.publish(hello_msg)
        rate.sleep()
if __name__ == '__main__':
    try:
        send_hello()
```

ROS Subscriber

ROS Subscriber

to write a subscriber

```
mkdir -p catkin_ws/src/first_package/scripts
cd catkin_ws/src/first_package/scripts
touch friend_hello_sub.py
chmod +x    friend_hello_sub.py )
```

ROS Subscriber Create

```
import rospy
from std_msgs.msg import String

def callback(msg):
    rospy.loginfo(rospy.get_caller_id()
            , 'message:%s', msg.data)
def receiver():
    rospy.init_node('receive_hello_node', anonymous=True)
    rospy.Subscriber('/send_hello_topic', String, callback)
    rospy.spin()

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

ROS Subscriber and Publisher Install

to install scripts (define in CMakeLists.txt)

```
catkin_install_python(PROGRAMS
scripts/friend_hello_pub.py
scripts/friend_hello_sub.py
DESTINATION $CATKIN_PACKAGE_BIN_DESTINATION
)
```

Let's try to say hello!

to run

```
roscore
rosrun first_package friend_hello_pub.py
rosrun first_package friend_hello_sub.py
```