Lesson 2 ROS Installation and Settings

http://wiki.ros.org/melodic/Installation

https://conceptbug.tistory.com/entry/ROS-Ubuntu-1804-LTS%EC%97%90-ROS-Melodic-%EC%84%A4%EC%B9%98%ED%95%98%EA%B8%B0-1

Step-by-step Installation (Routine Method)

1. Version Selection √

Different versions of Ubuntu have different corresponding versions of ROS .
The corresponding **ROS version** of Ubuntu 18.04 is **Melodic**.

2. Check Ubuntu Software and Update Source

3. Set the Download Source for ROS √

3.1 Set the Download Source

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

3.2 Set Public Key

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

3.3 Update Software Package

```
sudo apt update
```

4. Install ROS √

```
sudo apt install ros-melodic-desktop-full
```

5. Set the Environment Variables √

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

6. Install rosdep √

```
sudo apt-get install python3-rosdep
```

7. Initialization √

```
sudo rosdep init
rosdep update
```

Lesson 3 ROS Documents and Terms Instruction

1. The Composition of ROS File System √

ROS file consists of **Packages** and **Manifests (package.xml)**.

Packages: The function packages, the most basic unit of the Ros software, contains **node source code**, **configuration files**, **data definition**, etc.

Manifest (package xml): **The description file of function package** is used to define dependencies between the meta information ralated to the function packge, including version, maintainer, license agreement, etc.

2. Understand the Basic Terms of ROS √

The following table provides explanation on some of ROS basic terms.

Terms	Instruction
ROS Master	The connection and message communication between nodes.
Node	The smallest processing unit running in Ros is usually an executable file. Each node can use topic or service to communicate with other nodes.
Message	The variables of int, float, boolean and other data types.
Topic	A one-way asynchronous communication mechanism. The data can be transmitted between nodes through publishing message to topics or subscribing topics. The type of topics are determined by that of corresponding message.
Publish	Send data with the message type corresponding to the topic content.
Publisher	To execute publishing, the publisher nodes register their topics and other various information on the master node, and send messages to subscriber nodes that wish to subscribe.
Subscribe	Receive data with the massage type corresponding to the topic content.
Subscriber	To execute subscribing, the subscriber node registers its own topic and other various information on the master node and receives the information from the master node about all publisher nodes that have published the topic to which this node needs to subscribe.
Service	A two-way synchronous communication mechanism. The service client requests the service corresponding to the specific purpose task and the service server replies to the service.
Service Server	The node with request as input and respond as output
Service Client	The node with respond as input and request as output

3. Learn about ROS Common Files √

The following table describes some of the commonly used files in ROS:

Terms	Instruction
urdf file	It is a model file describing all elements of robot including link, joint, (kinematic parameter)axis, (kinetic parameter)dynamics, (visualization model)visual and (Collision Detection Model)collision.
srv file	It is stored in srv folder for ROS service message definition and contains two parts, request and reply. The symbol “--” is used to separate the request from the reply.
msg file	It is stored in msg folder and used for ROS topics message definitions.
package.xml	Describe the property of function package including its name, version, author, etc.
CmakeLists.txt	Compile configuration files by using Cmake.
Launch file	Launch file. It contains the node and services needed to systematically boot up the robot.

4. Learn About Common Commands √

ROS commands can be divided into five types: **ROS shell commands**, **ROS execution commands**, **ROS information commands**, **ROS catkin commands**, **ROS function package commands**. The followings are some of commonly used basic commands:

- ◆ **ROS shell commands**

Command	Explanation	Instruction
roscd	ros+cd(changes directory)	Come to the specified ROS package directory.
rosls	ros+ls(lists files)	Display the file and directory of ROS package
rosed	ros+ed(editor)	Edit the files of ROS package
roscp	ros+cp(copies files)	Copy the files of ROS package
rospd	ros+pushd	Add directory to ROS directory index
rosd	ros+directory	Display the directory in ROS directory index.

- ◆ **ROS Executing Command**

Command	Explanation	Instruction
roscore	ros+core	Boot ROS master node management
rosrun	ros+run	Run ROS node
roslaunch	ros+launch	Start launch file
rosclean	rosclean	Check or delete ROS log file

- ◆ **ROS Information command**

Command	Explanation	Instruction
rostopic	ros+topic	Confirm ROS topics information
rosservice	ros+service	Confirm ROS service information
rosnode	ros+node	Confirm ROS node information
rosparam	ros+param(parameter)	Confirm and modify ROS parameter information
rosbag	ros+bag	Record and replay ROS message
rosmsg	ros+msg	Display ROS message type
rossrv	ros+srv	Display ROS service type
rosversion	ros+version	Display ROS package and version information
roswtf	ros+wtf	Check ROS system

- ◆ **ROS catkin Command**

Command	Instruction
catkin_create_pkg	Automatically generate function package
catkin_make	Build the system based on catkin, build all functional packages in directory
catkin_eclipse	Modify the package generated with the catkin build system to work in Eclipse environment.
catkin_prepare_release	Update CHANGELOG.rst file generated by “catkin_generate_changelog”
catkin_generate_changelog	Generate or update CHANGELOG.rst file when publishing
catkin_init_workspace	Initialize the workspace directory of the catkin build system
catkin_find	Find all workspace directories in use

- ◆ **ROS Package Command**

Command	Explanation	Instruction
rospack	ros+pack(age)	View information related to ROS function package
rosinstall	ros+install	Install the ROS add-on package
rosdep	ros+dep(endencies)	Install the dependency files for the specified function package
roslocate	ros+locate	Display the information related to ROS function package
roscreate-pkg	ros+create-pkg	Automatically generate ROS function package (for old rosbuild system)
rosmake	ros+make	Build ROS function package (used for old rosbuild system)

Lesson 4 Create Workspace and Package

1. Create Workspace √

```
mkdir -p ~/catkin_ws/src
```

2. Create ROS Package √

```
cd ~/catkin_ws/src

catkin_create_pkg beginner_hiwonder std_msgs rospy roscpp

cd ~/catkin_ws

catkin_make

echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc

source ~/.bashrc

roscd beginner_hiwonder
```

Lesson 5 Write A Simple Publisher

1) `roscd beginner_hiwonder`

2) `mkdir scripts`

3) `cd scripts/`

4) Input "`vi velocity_publisher.py`" command to edit program, and then copy the following program. If need to modify, you can press "**i**" again. After modifying, press "**Esc**" and input "**:wq**" to save and exit the file.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#Publish this routine to the topic turtle1/cmd_vel， the type of message geometry_msgs::Twist
# The main purpose of this code is to control the movement of a small turtle in ROS so that it moves along the x-axis at a linear
velocity of 0.5m/s and rotates around the z-axis at an angular velocity of 0.2 rad/s, and outputs the relevant information.

import rospy
from geometry_msgs.msg import Twist #The Twist message type is used to represent the linear and angular velocity of the robot.

def velocity_publisher():
        # Initialize ROS node
    rospy.init_node('velocity_publisher', anonymous=True)
        # Initialize the ROS node with the node name 'velocity_publisher'
          and set it to be a name node (anonymous=True) so that the node name will be unique in the ROS network.

    turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
        # Create a Publisher object that will be used to publish messages of type Twist to
          a topic named /turtle1/cmd_vel with a queue length set to 10.
          This topic is typically used to control the baby turtles in a turtle simulator

        #Set the frequency of the loop to 10Hz
    rate = rospy.Rate(10)

    while not rospy.is_shutdown(): #Enter a loop until the ROS node is shut down.
            # Initialize messages of type geometry_msgs::Twist
        vel_msg = Twist() # Create a message object vel_msg of type Twist.
        vel_msg.linear.x = 0.5 # Set the linear velocity (x-axis direction) of vel_msg to 0.5m/s.
        vel_msg.angular.z = 0.2 # Set the angular velocity of vel_msg (in the direction around the z-axis) to 0.2 rad/s.
            # Publish message
        turtle_vel_pub.publish(vel_msg) # Post a vel_msg message to the /turtle1/cmd_vel topic.
        rospy.loginfo("Publsh turtle velocity command[%0.2f m/s, %0.2f rad/s]", vel_msg.linear.x, vel_msg.angular.z)
                                        # Output log messages for issued speed commands
        rate.sleep() # Delayed by cycle frequency
if __name__ == '__main__':
    try:
        velocity_publisher() # Call the velocity_publisher() function.
    except rospy.ROSInterruptException:  # If an ROS interrupt exception is encountered, the following code block is executed.
        pass
```

5） Input command “**chmod +x velocity_publisher.py**” to give executable permissions to the saved velocity_publisher.py

Lesson 6 Write A Simple Subscriber

1. Write Subscriber Node

1) Input “**cd catkin_ws/src/beginner_hiwonder/scripts/**” command and press “Enter”.

2) Enter “**vi pose_subscriber.py**” command to edit the program, and then copy the following program. If need to modify, press “**i**”. After modifying, press “**Esc**” and input “**:wq**” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#The purpose of this code is to subscribe to the turtle's location information and print out the x and y coordinates of the turtle
when a message is received.

import rospy
from turtlesim.msg import Pose #Import the Pose message type from the turtlesim package to represent the positional gesture
                               information of the turtle.

def poseCallback(msg): # Used to process incoming Pose messages.
    rospy.loginfo("Turtle pose: x:%0.6f, y:%0.6f", msg.x, msg.y) # Print the location information of the turtle, using the
rospy.loginfo function, formatted to output the x and y coordinates of the turtle to 6 decimal places.

def pose_subscriber():
    # ROS node initialization
    rospy.init_node('pose_subscriber', anonymous=True) # Initialize the ROS node with the node name 'pose_subscriber' and set
it to anonymous (anonymous=True) to ensure that the node name is unique in the ROS network.

    rospy.Subscriber("/turtle1/pose", Pose, poseCallback) # Create a Subscriber object that subscribes to a topic named
/turtle1/pose, with a message type of Pose, and registers the callback function poseCallback to be called when a message is
received.

    # Loop-waiting callback function
    rospy.spin() # Enter the ROS event loop, keeping the program running continuously, waiting to receive messages and calling
callback functions to process them.

if __name__ == '__main__':
    pose_subscriber() # Call the pose_subscriber() function to start the subscriber.
```

3) Input “**chmod +x pose_subscriber.py**” command and press “**Enter**” to give the executable permission to saved pose_subscriber.py.

2. Test Publisher and Subscriber

1) Input “**roscore**” command to start the node manager.

2) Input “**rosrun turtlesim turtlesim_node**” command and then press “**Enter**” to start TurtleSim.

3) Open a new terminal and enter “**rosrun beginner_hiwonder velocity_publisher.py**” command to run the publisher of velocity_publisher.py. Then press “**Ctrl+C”** to stop running the publisher node.

4) Open a new terminal and input “**rosrun beginner_hiwonder pose_subscriber.py**” command to run the subscriber of pose_subscriber.py Then press “**Ctrl+C**” to stop running the subscribe node.

Lesson 7 The Definition and Use Of Topics Message

1. Customize Topic Message

The specific operation steps for customizing topic massages are as following:

1) Open the terminal.

2) Enter “**roscd beginner_hiwonder**” command to locate to the package directory and press “**Enter**”.

3) Enter “**mkdir msg**” command and press “**Enter**”. Then create a new folder “msg” for storing text files.

4) Enter “**cd msg**” command and press “**Enter**”.

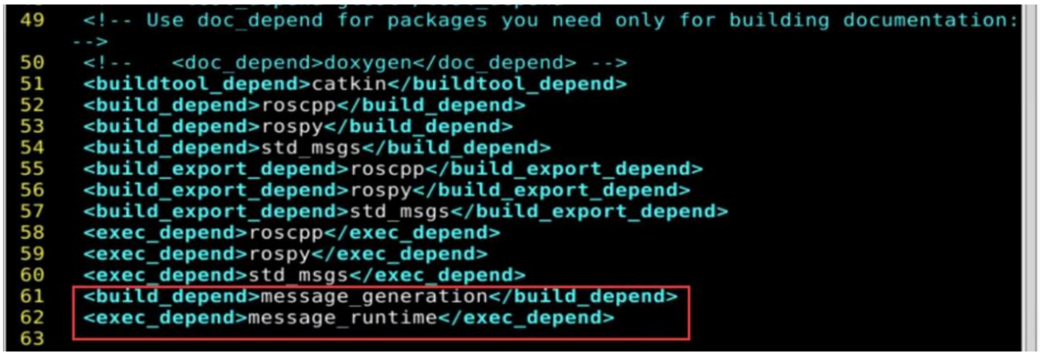
5) Enter “**vi Person.msg**” command to edit program and copy the following program. If want to modify, you can press “**i**” key. After modifying, press “**Esc**” and enter “**:wq**” to save and exit.

```
string name
uint8 age
uint8 sex

uint8 unknown = 0
uint8 male = 1
uint8 female = 2
```

6) Enter “**cd ..**”Enter “**vi package.xml**” command. Then copy the following program and add the package dependencies in the position shown in the figure below. If want to modify, you can press “**i**” again. After modifying, press “**Esc**” and enter “**:wq**” to save and exit.

7) Add the package dependencies in the position shown in the below figure:

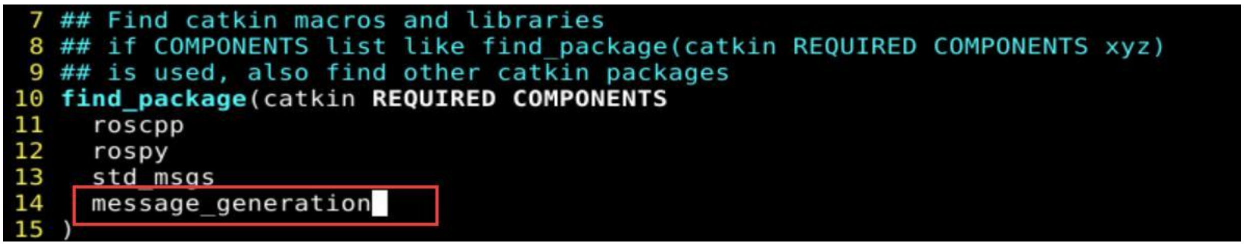
```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```



```
<!-- Use doc_depend for packages you need only for building documentation:
-->
<!--   <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>

```

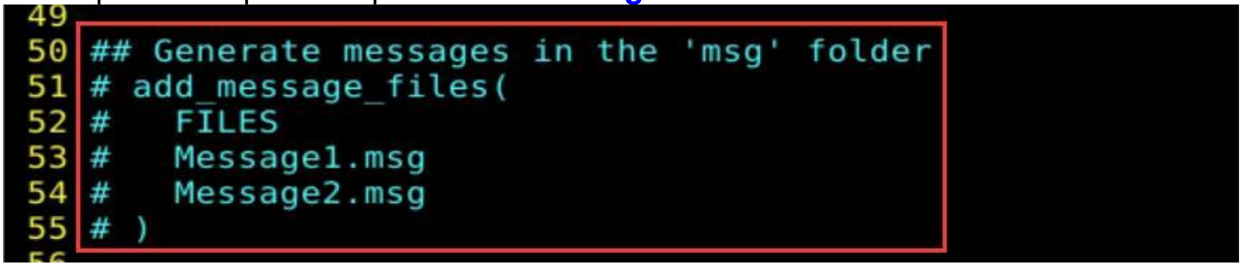
8) Enter “**vi CMakeLists.txt**” and press “**i**” to modify “**CMakeLists.txt**” file.

9) Add the required compilation option “**message_generation**” in the position shown in the figure below.

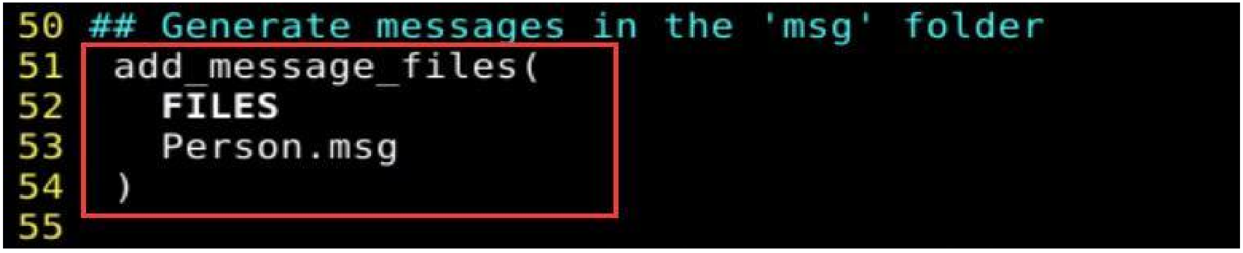


```
## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

10) Locate the code shown in the figure below. Then inverse comment the framed code and add the required compilation option “**Person.msg**”.

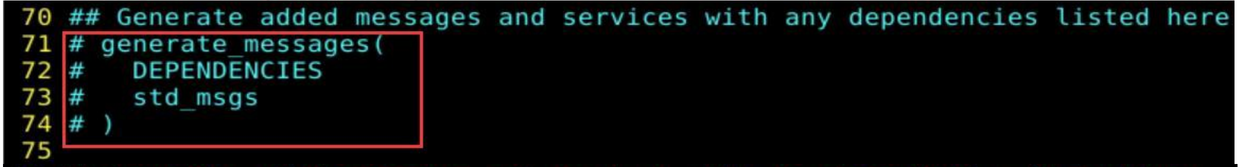


```
## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

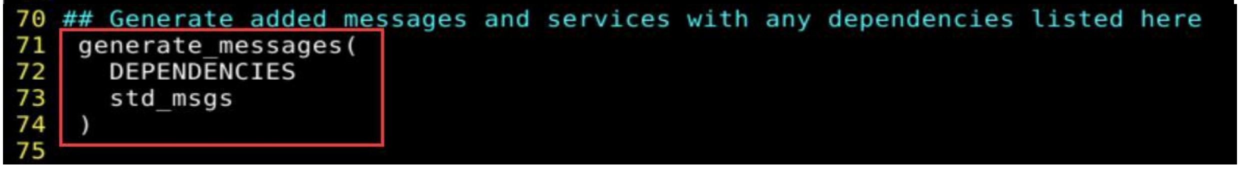


```
## Generate messages in the 'msg' folder
 add_message_files(
   FILES
   Person.msg
 )
```

11) Find the code shown in the figure below. Then inverse comment the code in red box and ensure that the required compilation options take effect.

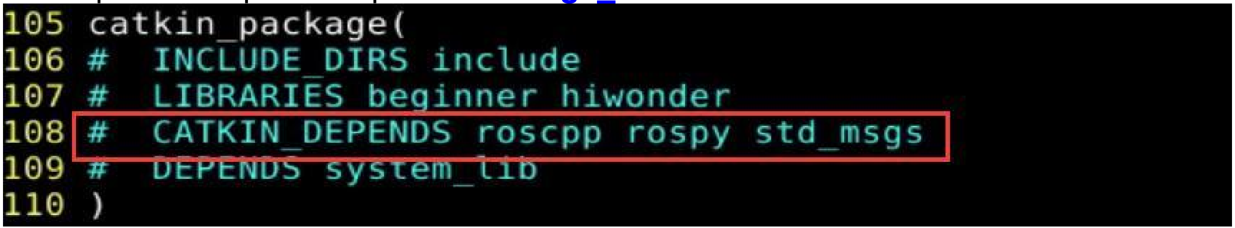


```
## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )
```

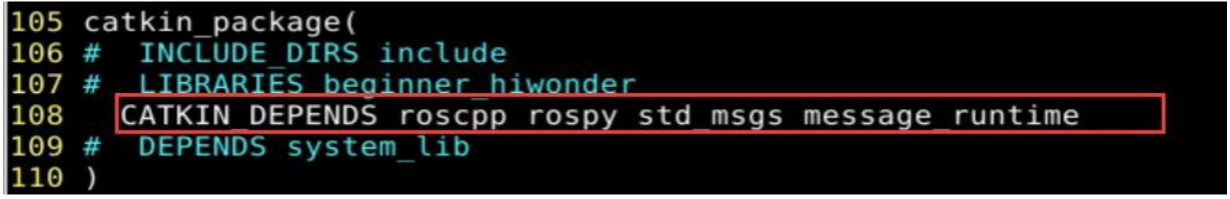


```
## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

12) Find the code shown in the figure below. Then inverse comment the code in red box and add the required compilation option “**message_runtime**”.



```
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES beginner_hiwonder
#  CATKIN_DEPENDS roscpp rospy std_msgs
#  DEPENDS system_lib
)
```



```
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES beginner_hiwonder
   CATKIN_DEPENDS roscpp rospy std_msgs message_runtime
#  DEPENDS system_lib
)
```

13) After modifying, press “**Esc**” and enter “**:wq**” to save and exit.

14) Enter the command “**rosmsg show beginner_hiwonder/Person**” and press “**Enter**” to check whether the message written can be recognized by system. When the words shown in red box appear, it means that they are recognized successfully.

Compiling

```
cd
cd catkin_ws/
catkin_make
```

2. The Use of Topic Message

2.1 Create Publisher and Subscriber Code

1) Open the terminal.

2) Enter “**roscd beginner_hiwonder**” command to locate to the package directory and press “**Enter**”.

3) Enter “**cd scripts**” command and press “**Enter**” to come to the folder “scripts” where Python scripts are stored.

4) Enter “**vi person_publisher.py**” command to edit program, and then copy the following program. If want to modify, you can press “**i**”. After modifying, press “**Esc**” and enter “**:wq**” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#This routine will pulish /person_info topic and customize message type beginner_hiwonder::Person.

import rospy
from beginner_hiwonder.msg import Person

def velocity_publisher():
    # Initialize ROS node
    rospy.init_node('person_publisher', anonymous=True)

    # Create a Publisher that publishes a topic named /person_info with a message type of beginner_hiwonder::Person and a queue length of 10.
    person_info_pub = rospy.Publisher('/person_info', Person, queue_size=10)

    # Set the frequency of the loop
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():

        # Initialize messages of type beginner_hiwonder::Person
        person_msg = Person()
        person_msg.name = "Tom";
        person_msg.age = 18;
        person_msg.sex = Person.male;

        # Post a message
        person_info_pub.publish(person_msg)
        rospy.loginfo("Publsh person message[%s, %d, %d]", person_msg.name, person_msg.age, person_msg.sex)

        # Delayed by cycle frequency
        rate.sleep()

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

5) Enter “**vi person_subscriber.py**” to edit program, and copy the following program. If want to modify, you can press “**i**”. After modifying, press “**Esc**” and enter “**:wq**” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will subscribe to the /person_info topic and customize the message type beginner_hiwonder::Person

import rospy
from beginner_hiwonder.msg import Person

def personInfoCallback(msg):

    rospy.loginfo("Subcribe Person Info: name:%s age:%d sex:%d", msg.name, msg.age, msg.sex)

def person_subscriber():

    # Initialize ROS node
    rospy.init_node('person_subscriber', anonymous=True)

    # Create a subscriber, subsribe the topic named /person_info and register the callback function personInfoCallback
    rospy.Subscriber("/person_info", Person, personInfoCallback)

    # loop and wait the callback function
    rospy.spin()

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

6) Enter “**chmod +x person_publisher.py**” command and press “**Enter**” to give the executable permission to the saved person_publisher.py

7) Enter “**chmod +x person_subscriber.py**” command and press “**Enter**” to give the executable permission to the saved person_subscriber.py

2.2 Run Publisher And Subscriber Nodes

1) Enter “**cd ~/catkin_ws**” command and press “**Enter**” to enter to catkin workspace.

2) Enter “**catkin_make**” command and press “**Enter”** to build all the packages in directory

3) Enter “**source ./devel/setup.bash**” command or Enter “**source ~/.bashrc**” command and press “**Enter**” to refresh the workspace environment.

4) Enter “**roscore**” command to start node manager.

5) Enter “**rosrun beginner_hiwonder person_publisher.py**” command and press “**Enter**” to run publisher node. If want to stop running node, you can press “**Ctrl+C**”.

6) Open a new terminal. Enter “**rosrun beginner_hiwonder person_subscriber.py**” command and press “**Enter**” to run the subscriber node. If want to stop running node, you can press “**Ctrl+C**”.

Note:
1) The publisher node needs to be started first, and then the subscriber node can subscribe message.

2) If need to receive the publisher messages completely, you can start the subscriber node first and then the publisher node.

Lesson 8 Write A Simple Client

This section takes the creation of a simple service (Client) node turtle_spawn.py as an example to explain and this node publishes a request for the client to spawn a new turtle by means of a program.

1. Configure Client Code Compilation Rule

1) Enter "**cd catkin_ws/src/beginner_hiwonder/scripts/**" command and press "Enter".

2) Enter "**vi turtle_spawn.py**" command to edit program and copy the following program. If want to program, you can press "i", and then press "Esc" to enter ":wq" to exit and save.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will request /spawn service and the service data type is
turtlesim::Spawn

import sys
import rospy
from turtlesim.srv import Spawn

def turtle_spawn():
        # Initialize ROS node
    rospy.init_node('turtle_spawn')
        # After finding the /spawn service, create a service client, and then connect the service named /spawn.
    rospy.wait_for_service('/spawn')
    try:
        add_turtle = rospy.ServiceProxy('/spawn', Spawn)

                # Request service call and input request data
        response = add_turtle(2.0, 2.0, 0.0, "turtle2")
        return response.name
    except rospy.ServiceException, e:
        print "Service call failed: %s"%e

if __name__ == "__main__":
        #The service calls and displays the result of call.
    print "Spwan turtle successfully [name:%s]" %(turtle_spawn())
```

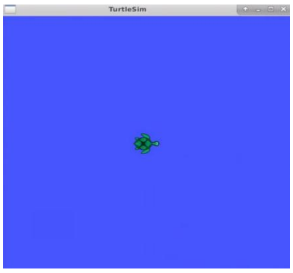
3) Enter "**chmod +x turtle_spawn.py**" command and press "Enter" to give the executable permission to the saved turtle_spawn.py

2. Run Client

1) Enter "**roscore**" command to start node manager

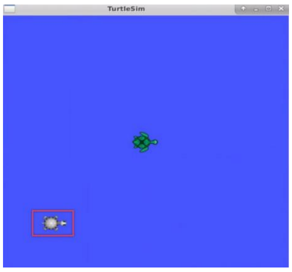
2) Enter "**rosrun turtlesim turtlesim_node**" command and press "enter" to run turtlesim.

At this time, the interface will pop up the turtlesim window, as the figure shown below:



3) Open a new terminal. Enter “**rosrun beginner_hiwonder turtle_spawn.py**” command and press “Enter” to run the client.

At this time, client will send the request to server and respond to start another turtle.



Lesson 9 Write A Simple Server

1. Create Service Code

Note: Before creating service code, you need to create workspace and package first. The specific operation steps can be viewed in file “ROS Basic Lessons->Lesson 3 Create Workspace and Package”.

1) Enter “**cd catkin_ws/src/beginner_hiwonder/scripts/**” command and press “Enter”.

2) Enter “**vi turtle_command_server.py**” command to edit program and copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import thread,time
from geometry_msgs.msg import Twist
from std_srvs.srv import Trigger, TriggerResponse

pubCommand = False;
turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

def command_thread():
    while True:
        if pubCommand:
            vel_msg = Twist()
            vel_msg.linear.x = 0.5
            vel_msg.angular.z = 0.2
            turtle_vel_pub.publish(vel_msg)
        time.sleep(0.1)

def commandCallback(req):
    global pubCommand
    pubCommand = bool(1-pubCommand)
    rospy.loginfo("Publish turtle velocity command![%d]", pubCommand)
    return TriggerResponse(1, "Change turtle command state!")

def turtle_command_server():
    rospy.init_node('turtle_command_server')
    s = rospy.Service('/turtle_command', Trigger, commandCallback)
    print "Ready to receive turtle command."
    thread.start_new_thread(command_thread, ())
    rospy.spin()

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

3) Enter “**chmod +x turtle_command_server.py**” command and press “Enter” to give the executable permission to the saved turtle_command_server.py.

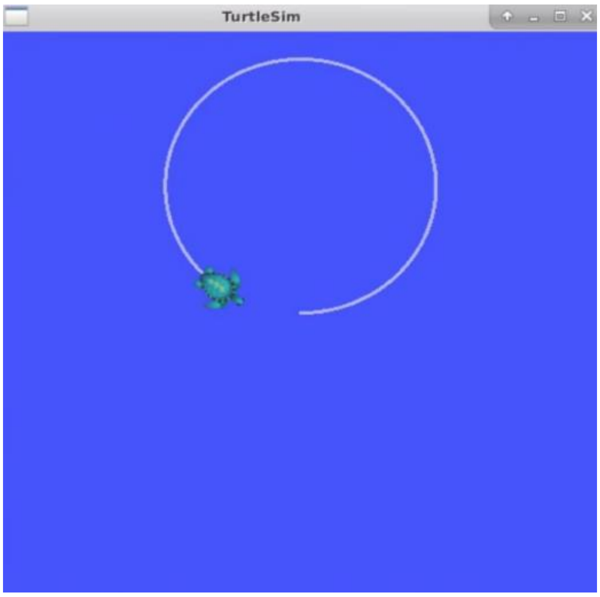
2. Run Server Node

1) Enter “**roscore**” command to start node manager.

2) Enter “**rosrun turtlesim turtlesim_node**” command and press “Enter” to start turtlesim simulator window.

3) Open a new terminal. Then enter “**rosrun beginner_hiwonder turtle_command_server.py**” and press “Enter” to run service node. If need to stop running the node, you can press “Ctrl+C”.

4) Open a new terminal again. Enter “**rosservice call /turtle_command "{}"**” command and press “Enter” to move the turtle along the circular path.



Lesson 10 The Definition and Use of Service Data

1. Customize Service Data

Note: Before customizing service data, the workspace and package need to be created first. The specific operation steps can be viewed in file “ROS Basic Lessons/Lesson 3 Create Workspace and Package”.

The specific operation steps to customize service data are as follow:

1) Open the terminal.

2) Enter “**roscd beginner_hiwonder**” command to go to the package directory and press “Enter”.

Note: If there is a prompt “**No such package/stack ‘beginner_hiwonder**” appears, it means that the package does not exist in the environment variable ROS_PACKAGE_PATH. The specific solution can be viewed in “ROS Basic Lessons/Lesson 3 Create Workspace and Package”. After the problem is solved, please repeat the current step.

3) Enter “**mkdir srv**” command and press “Enter”. Then create a new folder “srv” for storing text files.

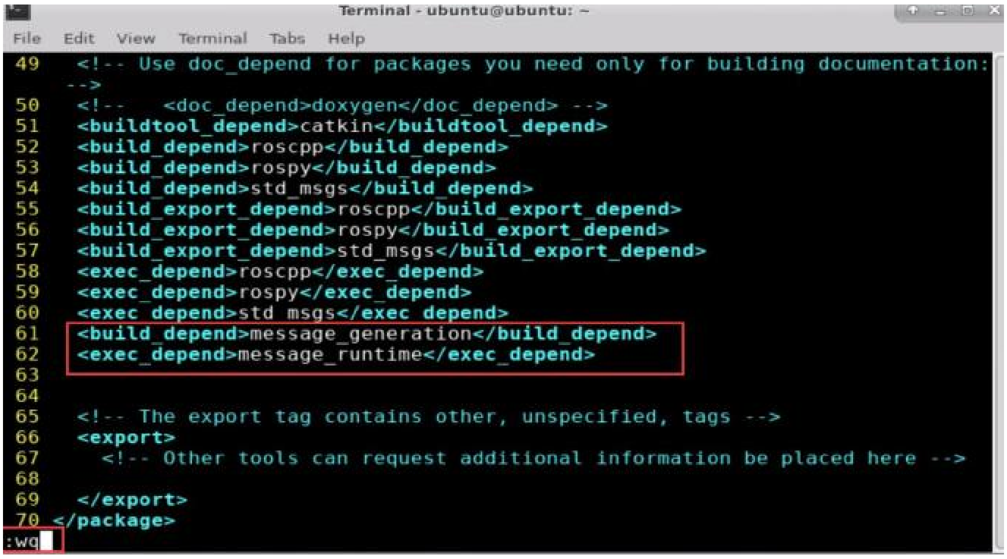
4) Enter “**vi Person.srv**” command to edit program, and then copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

```
string name
int8 age
int8 sex

int8 unknown = 0
int8 male = 1
int8 female = 2

---
string result
```

5) Enter “**cd ~/catkin_ws/src/beginner_hiwonder/**” command, and then press “Enter”.

6) Enter “**vi package.xml**” command. Then copy the following program and add the package dependencies in the position shown in the figure below. If want to modify, you can press “i” again. After modifying, press “Esc” and enter “:wq” to save and exit.

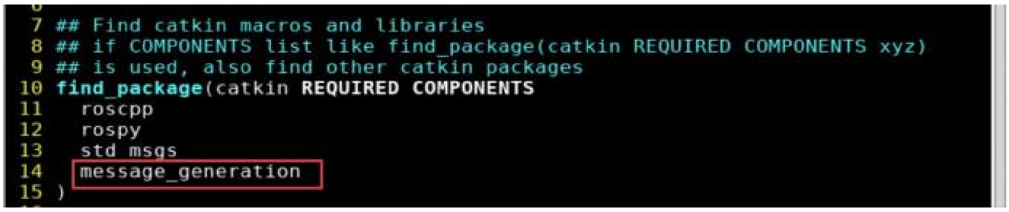
```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```


Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help

```
  <!-- Use doc_depend for packages you need only for building documentation:
 -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
:wq
```

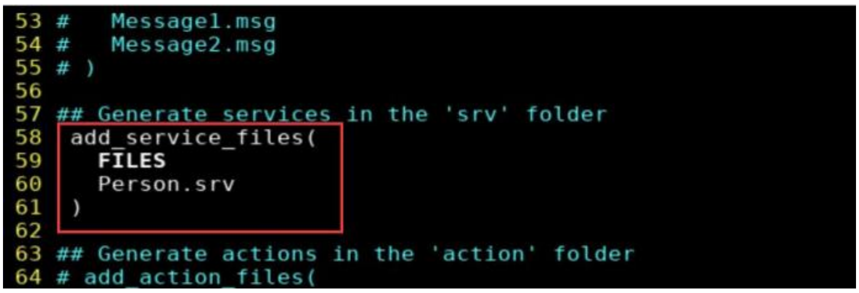
7) Enter “**vi CMakeLists.txt**” and press “i” to modify “CMakeLists.txt” file.

8) Add the required compilation option “message_generation” in the position shown in the figure below.



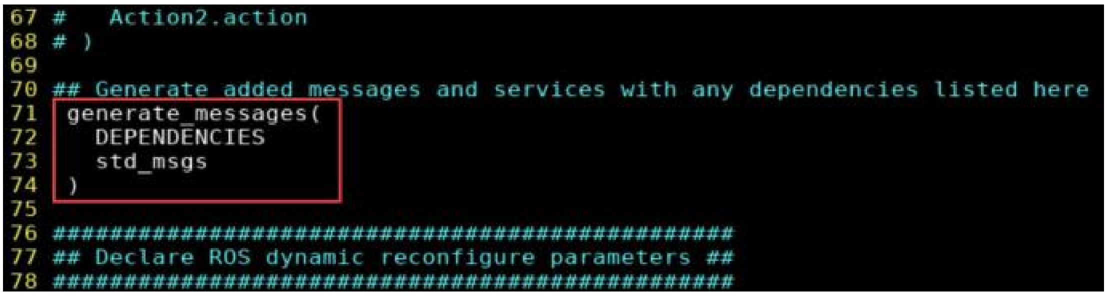
```
## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

9) Find the code shown in the figure below. Then uncomment the framed code and add the required compilation option “Person.srv”.



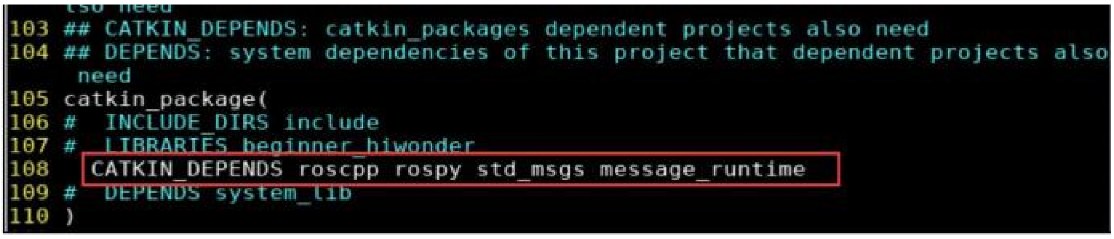
```
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
add_service_files(
  FILES
  Person.srv
)

## Generate actions in the 'action' folder
# add_action_files(
```

10) Find the code shown in the figure below. Then uncomment the code in red box and ensure that the required compilation options take effect.



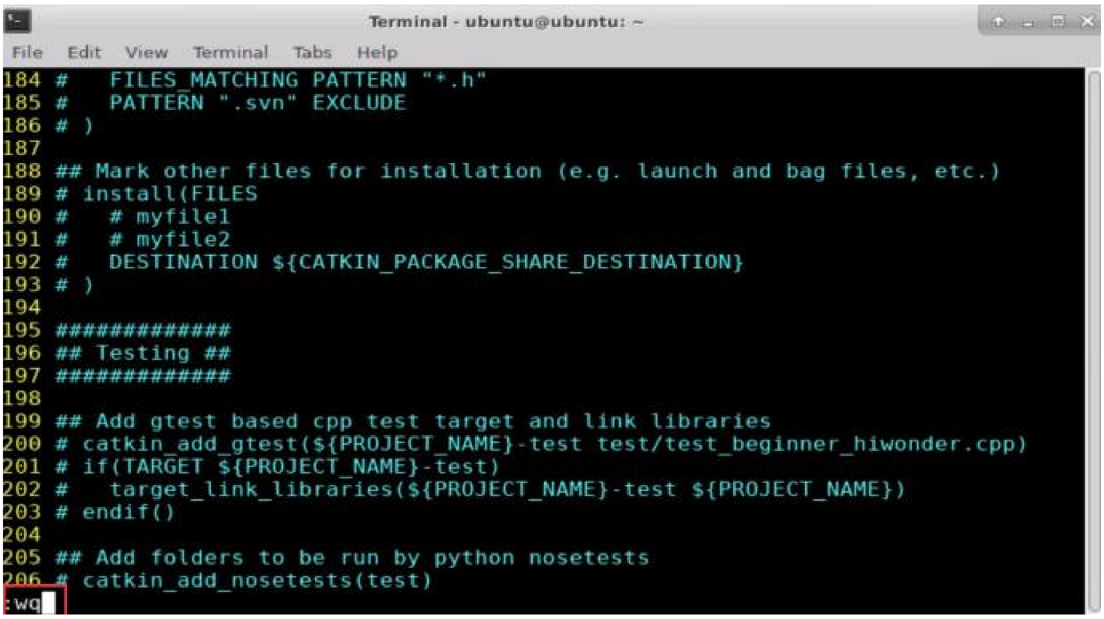
```
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  std_msgs
)

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################
```

11) Find the code shown in the figure below. Then uncomment the code in red box and add the required compilation option “message_runtime”.



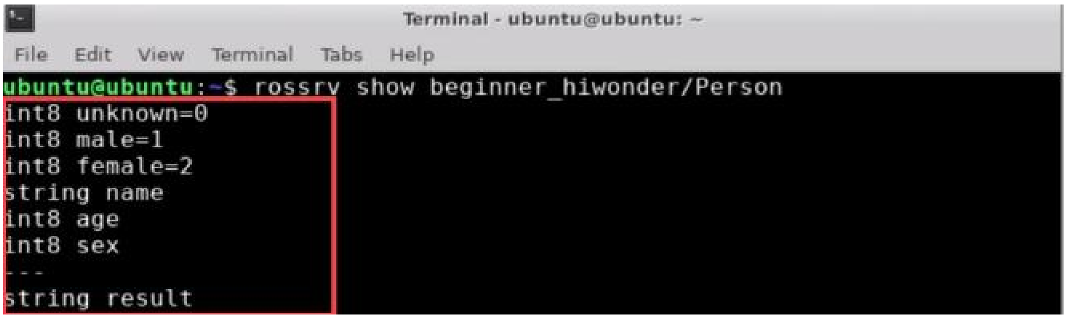
```
lso need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also
 need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES beginner_hiwonder
  CATKIN_DEPENDS roscpp rospy std_msgs message_runtime
#  DEPENDS system_lib
)
```

12) After modifying, press “Esc” and enter “:wq” to save and exit.



```
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_beginner_hiwonder.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
:wq
```

13) Enter the command “**rosmsg show beginner_hiwonder/Person**” and press “Enter” to check whether the massage written can be recognized by system. When the words shown in red box appear, it means that they are recognized successfully.



```
ubuntu@ubuntu:~$ rossrv show beginner_hiwonder/Person
int8 unknown=0
int8 male=1
int8 female=2
string name
int8 age
int8 sex
---
string result
```

2. The Use of Service Data

2.1 Create Server and Client Data

1) Open the terminal.

2) Enter “**cd catkin_ws/src/beginner_hiwonder/scripts/**” command and press “Enter” to come to the folder “**scripts**” where Python scripts are stored.

3) Enter “**vi person_server.py**” command to edit program, and then copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will execute the /show_perso service with service data type beginner_hiwonder::Person

import rospy
from beginner_hiwonder.srv import Person, PersonResponse

def personCallback(req):
            # Display request message
        rospy.loginfo("Person: name:%s age:%d sex:%d", req.name, req.age, req.sex)
            # Data feedback
        return PersonResponse("OK")

def person_server():
            # Initialize ROS node
        rospy.init_node('person_server')
            # Create a server named /show_person and register personCallback function
        s = rospy.Service('/show_person', Person, personCallback)
            # loop and wait callback function
        print "Ready to show person informtion."
        rospy.spin()

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

4) Enter “**vi person_subscriber.py**” to edit program, and copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will request /show_person service with the service data type beginner_hiwonder::Person

import sys
import rospy
from beginner_hiwonder.srv import Person, PersonRequest

def person_client():
            # Initialize ROS node
        rospy.init_node('person_client')
            # After finding the /spawn service, create a service client, and then connect service name
               /spawn
        rospy.wait_for_service('/show_person')
        try:
            person_client = rospy.ServiceProxy('/show_person', Person)
            # Request service call, input request data
            response = person_client("Tom", 20, PersonRequest.male)
            return response.result
        except rospy.ServiceException, e:
            print "Service call failed: %s"%e
if __name__ == "__main__":
            #Service call and display the call result
        print "Show person result : %s" %(person_client())
```

5) Enter “**chmod +x person_server.py**” and “**chmod +x person_client.py**” command, and then press “Enter” to give the executable permission to the saved person_publisher.py

2.2 Run Server and Client Nodes

1) Enter “**cd ~/catkin_ws**” command and press “Enter” to enter to catkin workspace.

2) Enter “**catkin_make**” command and press “Enter” to build all the packages in directory.

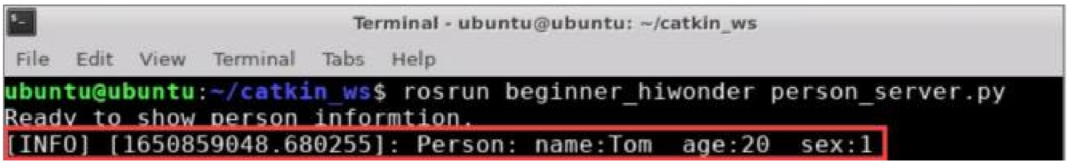
3) Enter “**source ./devel/setup.bash**” command and press “Enter” to refresh the workspace environment.

4) Enter “**roscore**” command to start node manager.

5) Enter “**rosrun beginner_hiwonder person_publisher.py**” command and press “Enter” to run publisher node. If want to stop running node, you can press “Ctrl+C”.

6) Open a new terminal. Enter “**rosrun beginner_hiwonder person_client.py**” command and press “Enter” to run the client node.

7) After running the client node, the terminal window for starting server node will print the content shown in the red box in the figure below.


Terminal - ubuntu@ubuntu: ~/catkin_ws
File Edit View Terminal Tabs Help

```
ubuntu@ubuntu:~/catkin_ws$ rosrun beginner_hiwonder person_server.py
Ready to show person informtion.
[INFO] [1650859048.680255]: Person: name:Tom  age:20  sex:1
```

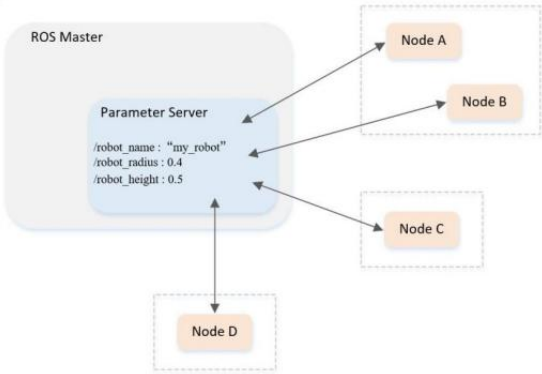
Lesson 11 Parameters Usage and Programming Method

1.Service Model

There is a parameter server in ROS master, which is a global dictionary to store configuration parameters among nodes. For example, parameter server for saving our name, radius and height can be globally accessible by each node.

If I access the robot name in Node A, I will get a value of “my_rot”. It only needs to send a query request to our ROS master, and then return the result of “my_rot”. The same goes for Node B, Node C, and Node D.

The parameter server model is shown in the following figure:

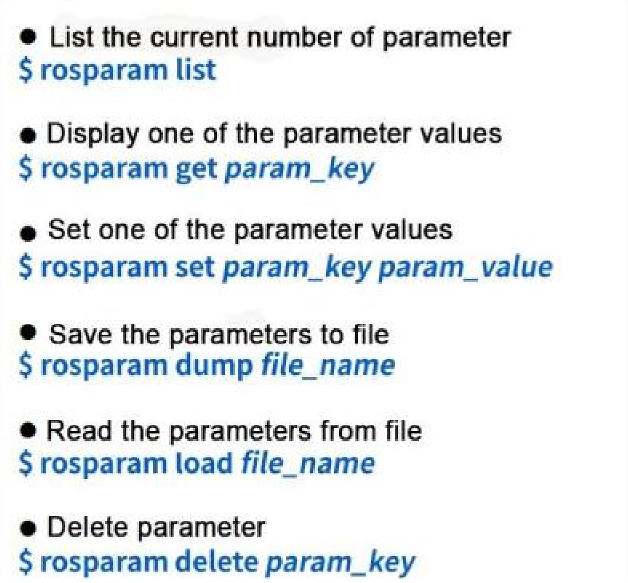


Parameter Model (Global dictionary)

2.rosparam Parameter

2.1 rosparam Detailed Parameter

Let’s get to know rosparam first and the detailed parameters are as follow:



- List the current number of parameter

```
$ rosparam list
```

- Display one of the parameter values

```
$ rosparam get param_key
```

- Set one of the parameter values

```
$ rosparam set param_key param_value
```

- Save the parameters to file

```
$ rosparam dump file_name
```

- Read the parameters from file

```
$ rosparam load file_name
```

- Delete parameter

```
$ rosparam delete param_key
```

2.2 Run Turtlesim Routine

Taking turtlesim project as an example, run the turtlesim routine first. The specific operation steps are as follows:

1) Enter “**roscore**” command and press “Enter”.

Note: If the prompt “**roscore cannot run as another roscore/master is already running**” appears, which means node manager has been started before and this step can be skipped.

2) Enter the command “**rosrun turtlesim turtlesim_node**”, and then press “Enter” to open the turtle simulator.

2.3 The Use of rosparam

The operation steps for the use of rosparam are as follow:

1) Open a new terminal.

2) Enter “**rosparam**” command and press “Enter”.

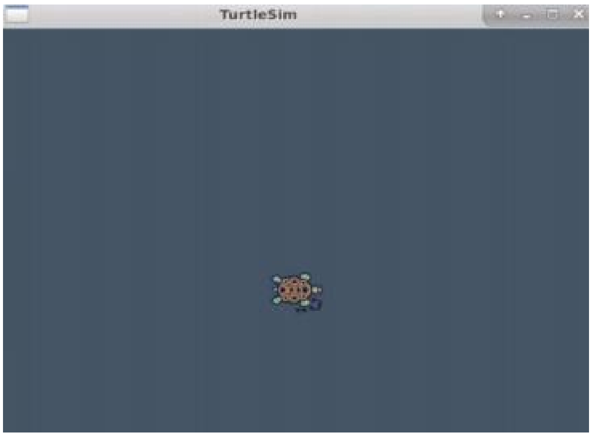
3) Enter the command "**rosparam list**" and press “Enter” to query the number of turtle parameters.

4) Enter “**rosparam get /turtlesim/background_b**” command and press “Enter” to get the value of “background_b”. The same method goes for getting other values.

5) Enter “**rosparam set /turtlesim/background_b 100**” command and press “Enter” to set “background_b” value. The same method goes for setting other values.

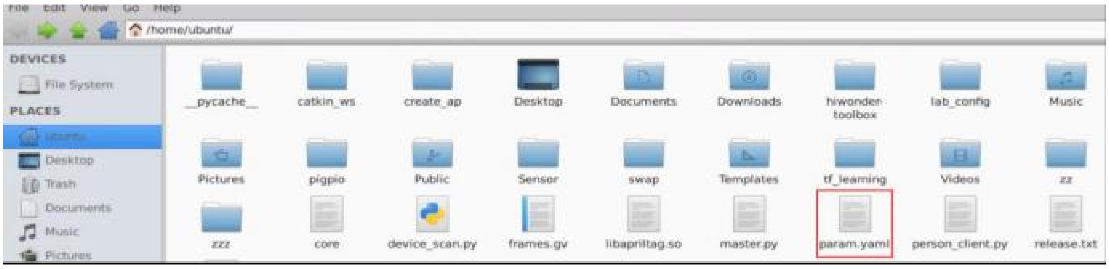
6) Enter “**rosparam get /turtlesim/background_b**” command and press “Enter”. Then you can find that the value has been modified to 100.

7) Enter “**rosservice call clear "{}"** ” command and press “Enter” to send the request to change color.

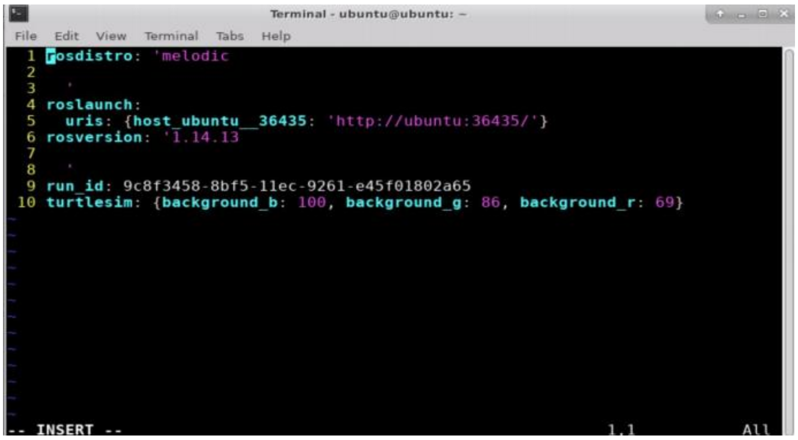


8) Enter “**rosparam dump param.yaml**” command and press “Enter” to save file.

9) The created file is saved in the following path and open it directly.

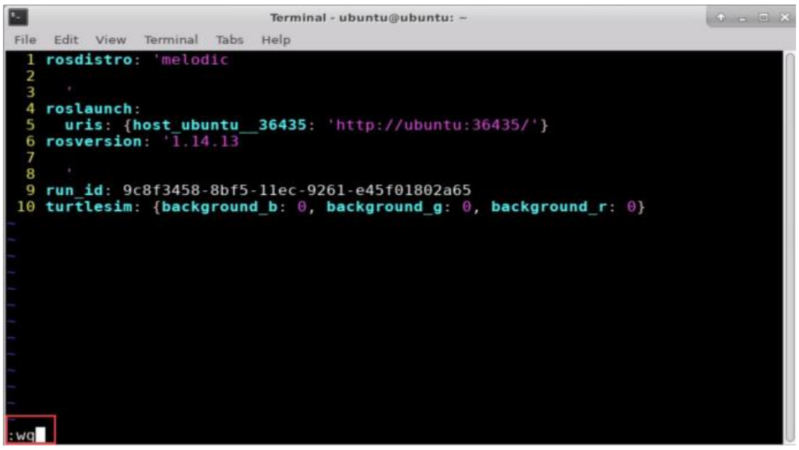


10) Enter “**vi param.yaml**” command and press “Enter”. Then press “i” to modify “param.yaml”.


Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help

```
rosdistro: 'melodic

  '
roslaunch:
  uris: {host_ubuntu__36435: 'http://ubuntu:36435/'}
rosversion: '1.14.13

  '
run_id: 9c8f3458-8bf5-11ec-9261-e45f01802a65
turtlesim: {background_b: 100, background_g: 86, background_r: 69}
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                          1,1           All
```

11) The color can be modified to 0 which is black. After modifying, press “Esc” and enter “:wq” to save and exit.

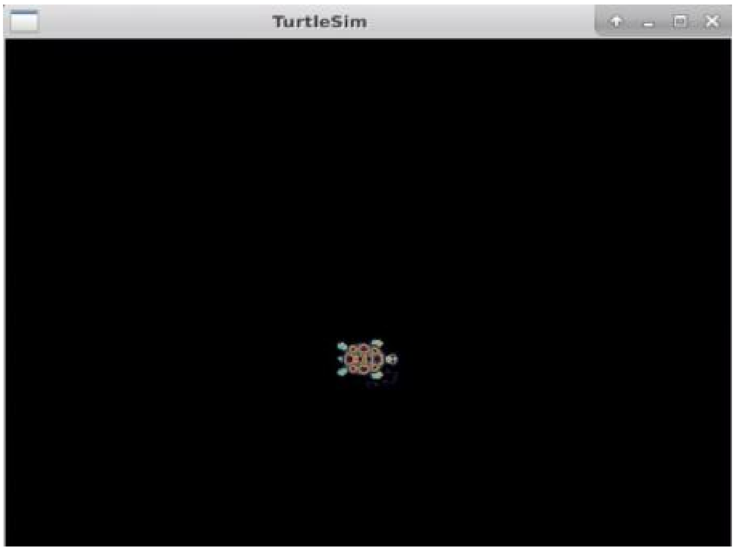

Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help

```
rosdistro: 'melodic

  '
roslaunch:
  uris: {host_ubuntu__36435: 'http://ubuntu:36435/'}
rosversion: '1.14.13

  '
run_id: 9c8f3458-8bf5-11ec-9261-e45f01802a65
turtlesim: {background_b: 0, background_g: 0, background_r: 0}
~
~
~
~
~
~
~
~
~
~
~
:wq
```

12) Enter “**rosparam load param.yaml**” command and press “Enter” to load the file.

13) Enter “**rosparam get /turtlesim/background_b**” command and press “Enter” to check the loading effect.

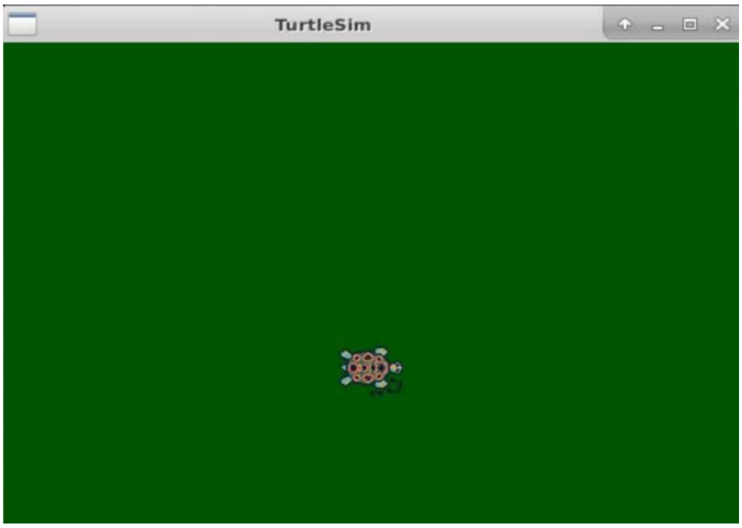
14) Enter “**rosservice call clear "{}"** ” command and press “Enter” to send a request that changes the background scolor to black.



15) Enter “**rosparam delete /turtlesim/background_g**” command and press “Enter” to delete the color of g.

16) Enter “**rosparam list**” command and press “Enter” to check effect.

17) Enter “**rosservice call clear "{}"** ” command and press “Enter” to refresh the blackground color to check effect.



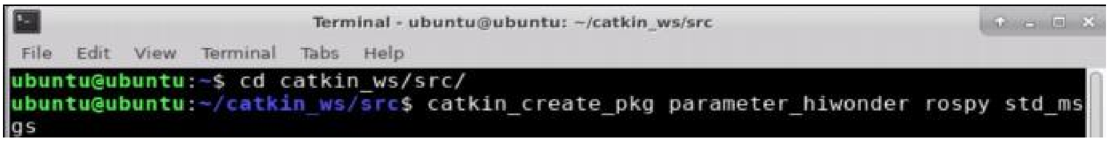
3. Programming Method

The operation steps for creating package is as follow:

1) Enter “**cd catkin_ws/src/**” command and press “Enter” to come to workspace.

2) Enter “**catkin_create_pkg parameter_hiwonder rospy std_msgs**” command and press “Enter” to create package.

3.1 Write Control Program


Terminal - ubuntu@ubuntu: ~/catkin_ws/src
File Edit View Terminal Tabs Help

```
ubuntu@ubuntu:~$ cd catkin_ws/src/
ubuntu@ubuntu:~/catkin_ws/src$ catkin_create_pkg parameter_hiwonder rospy std_ms
gs
```

1) Open the terminial.

2) Enter “**roscd parameter_hiwonder**” command and press “Enter” to come to the package directory.

3) Enter “**mkdir scripts**” command and press “Enter” to create a new folder “scripts” where Python scripts are stored.

4) Enter “**cd scripts/**” command and press “Enter” to come to the “scripts” folder.

5) Enter the command “**vi parameter_config.py**” to edit the program, and then copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine set and read the parameter in turtle routine

import sys
import rospy
from std_srvs.srv import Empty

def parameter_config():
        # Initialize ROS node
    rospy.init_node('parameter_config', anonymous=True)
        # read the parameter of blackground color
    red = rospy.get_param('/turtlesim/background_r')
    green = rospy.get_param('/turtlesim/background_g')
    blue = rospy.get_param('/turtlesim/background_b')
    rospy.loginfo("Get Backgroud Color[%d, %d, %d]", red, green, blue)
        # set the parameter of blackground color
    rospy.set_param("/turtlesim/background_r", 255);
    rospy.set_param("/turtlesim/background_g", 255);
    rospy.set_param("/turtlesim/background_b", 255);
    rospy.loginfo("Set Backgroud Color[255, 255, 255]");
        # read the parameter of blackground color
    red = rospy.get_param('/turtlesim/background_r')
    green = rospy.get_param('/turtlesim/background_g')
    blue = rospy.get_param('/turtlesim/background_b')
    rospy.loginfo("Get Backgroud Color[%d, %d, %d]", red, green, blue)
        # After finding /spawn, create a service client, and then connect service named /spawn.
    rospy.wait_for_service('/clear')

    try:
        clear_background = rospy.ServiceProxy('/clear', Empty)
        # Request service call, enter request data
        response = clear_background()
        return response
    except rospy.ServiceException, e:
        print "Service call failed: %s"%e

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

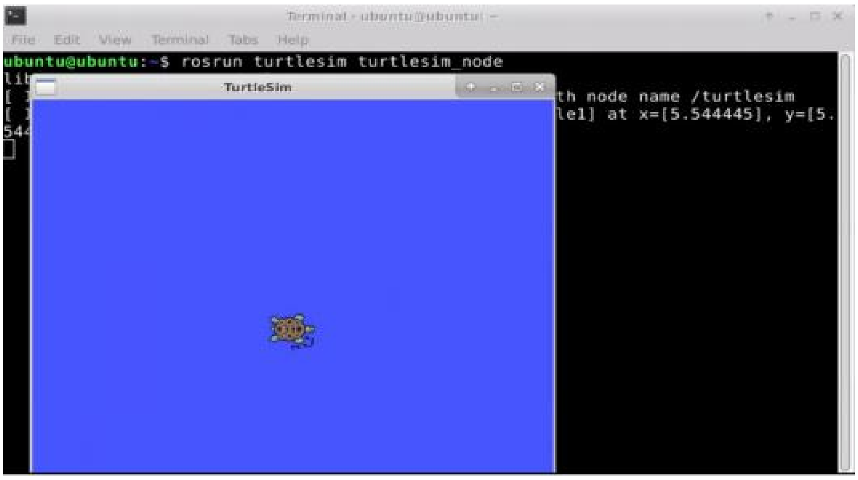
6) Enter “**chmod +x parameter_config.py**” command and press “Enter” to give the executable permission to the saved parameter_config.py.

4.2 Run Program

1) Enter “**roscore**” command and press “Enter” to start node manager.

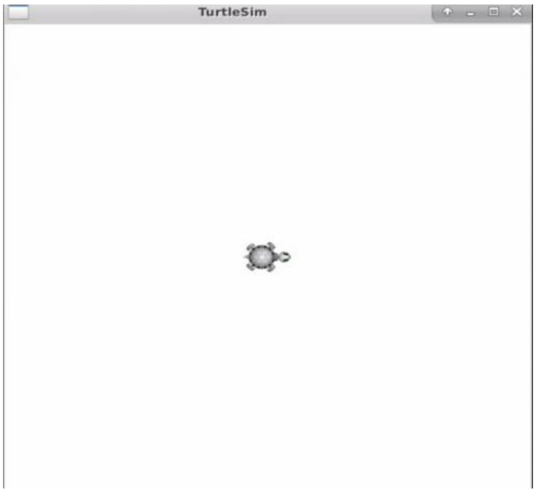
Note: If the prompt “roscore cannot run as another roscore/master is already running” appears, it means node manager has been started up so this step can be skipped directly.

2) Enter “**rosrun turtlesim turtlesim_node**” command and press “Enter” to open the turtle simulator.


Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help
TurtleSim

```
ubuntu@ubuntu:~$ rosrun turtlesim turtlesim_node
li
[                                                  th node name /turtlesim
[                                                  le1] at x=[5.544445], y=[5.
544
```

3) Enter “**rosrun parameter_hiwonder parameter_config.py**” command and press “Enter” to run the program, which changes the background color of turtle to white, as the figure shown below:



4) If want to stop running program, you can press “Ctrl+C”.

Lesson 12 The Use of Launch File

1.Preface

In the previous lessons, common line is tried to run a new node. However, as the more complex robotic systems are created, it becomes increasingly cumbersome to open multiple terminals and re-enter settings.

Therefore, through creating a launch file, we can start up and configure multiple ROS nodes to be executed as the same time, as well as start more functions. In addition, ROS Master can be started automatically

2.Basic Grammar of Launch File

2.1 Tag Instruction

Launch file: the configuration and startup of multiple nodes through XML file.

```
<launch>
    <node pkg=”turtlesim” name=”sim1” type=”turtlesim_node”/>
    <node pkg=”turtlesim” name=”sim2” type=”turtlesim_node”/>
</launch>
```

The above is an example of the simplest Lanuch file. The root element in the Launch file is defined by the <Lanuch> tag.

The node tag is one of the most commonly used tags in Launch files.

```
<node pkg=”package-name”type=”executable-name”name=”node-name”/>
```

1) <node>：The starting node.

2) pkg：The name of package in which the node is located.

3) type：The name of the node executable file. If it is written in python, the suffix “.py” needs to be added. If it is written in C++, the name of the executable file can be written directly without the suffix “.cpp”.

4) The name of the node when it runs. Each node needs its own unique name. If want to start up two identical files, you can write two different names, for example, start two turtles.

More parameters in <node> can be set in addition to pkg、type、name, as follow:

```
<launch>
    <node
        pkg=""
        type=""
        name=""
        respawn="true"
        required="true"
        launch-prefix="xterm -e"
        output="screen"
        ns="some_namespace"
        args=""
    />
</launch>
```

Parameter	Function
output	By default, information about launching node is stored in the following log file（/.ros/log/run_id/node_name-number-stdout.log） by which set the parameters, which can be set here to make the information appear on the screen. For example, output = “screen”.
required	Whether all the nodes started by node are shut down
respawn	Whether to automatically restart the node if it shuts down unexpectedly
ns	The node is classified into a different namespace, that is, the prefix given by ns is added in front of the node name

2.2 Parameter Settings

<param>/：Set the parameters running in ROS and store them in parameter server.

```
<param name=”output_frame”value=”odom”/>
```

1) name：parameter name

2) value：parameter value

<rosparam>：load multiple paramters in a parameter file

<rosparam file=”params.yaml”command=”load”ns=”params” />

<arg>：the local variable in launch file. It is restricted to Launch file.

<arg name=”arg-name” default=”arg-value” />

1) name：parameter name

2) value：parameter value

2.3 Remap nesting

<remap>：remap the name of ROS computation graph resource

<remap from=”/turtlebot/cmd_vel”to=”/cmd_vel”/”>

1) from：the original name

2) to：the name after mapping

<include>：Include other launch files which is similar to header in C programming language.

<include file=”$(dirname)/other.launch”>

file：The path to the other Launch files included.

3. Operation Steps

Take the startup of turtle sporting program with a launch file as an example, and the steps are as follow:

1) Open the terminal, and then enter “**cd catkin_ws/src/beginner_hiwonder**” command.

2) Enter “**mkdir lanuch**” command to create a launch folder.

3) Enter “**cd lanuch**” command to enter the launch folder.

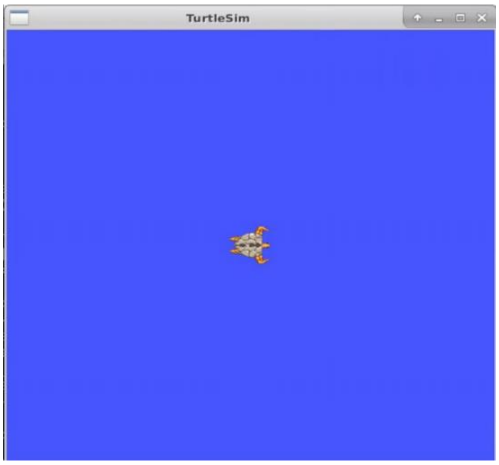
4) Enter “**vi turtlesim_launch_test.lanuch**” command to open the file created in step 1 through vi eiditor.

5) Copy the following content into the folder.

```
<launch>
<node pkg="turtlesim" type="turtlesim_node" name="turtlesim" />
<node pkg="turtlesim" type="turtle_teleop_key" name="turtle_teleop_key" output="screen" />
</launch>
```

6) Press “Esc” and enter “:wq” to save and exit.

7) Enter “**roslaunch beginner_hiwonder turtlesim_launch_test.lanuch**” command to start TurtleSim.



Lesson 13 The Programming Realization of TF Coordinates Broadcasting and Listening

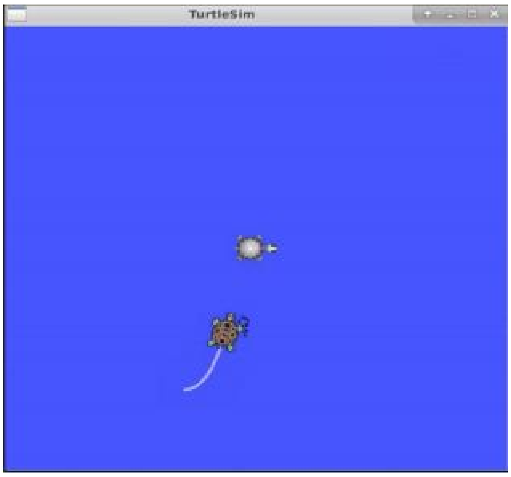
1.Coordinates Transformation

Before programming, the coordinates transformation of robot needs to be learned about first. Here takes running TurtleSim project as an example and the operation steps are as follow:

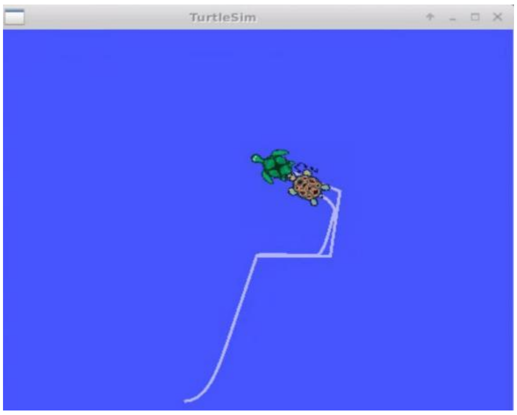
1) Enter “**sudo apt-get install ros-melodic-turtle-tf**” command to install the package.

Among them, “melodic” corresponds to ROS version.

2) Enter “**roslaunch turtle_tf turtle_tf_demo.launch**” command to run launch file.

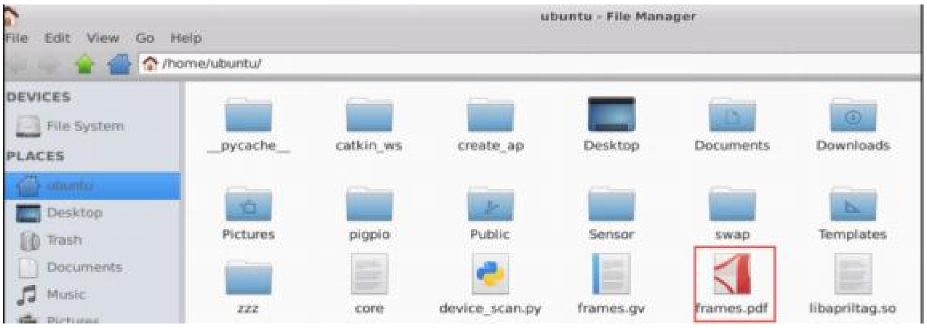


3) Enter “**rosrun turtlesim turtle_teleop_key**” command to run turtle keyboard control node.

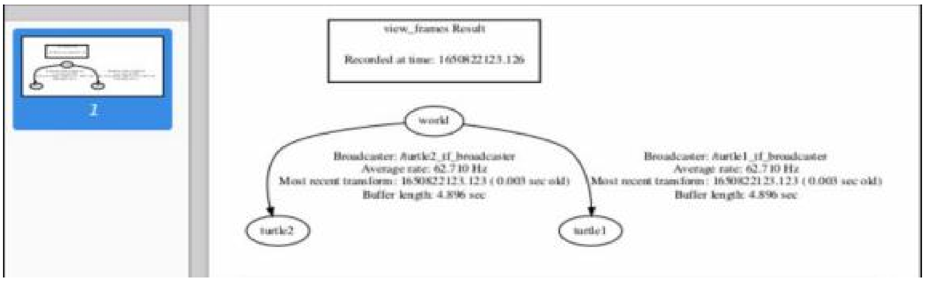


4) Enter “**rosrun tf view_frames**” command to view the frame.

5) Find the file “frames.pdf” under the main directory as the figure shown below:



6) Open the file “frames.pdf”, and then the position relationship between TF coordinates in current system can be viewed, as the figure shown below:


1
view_frames Result
Recorded at time: 1650822123.126
world
Broadcaster: /turtle2_tf_broadcaster
Average rate: 62.710 Hz
Most recent transform: 1650822123.123 (0.003 sec old)
Buffer length: 4.896 sec
Broadcaster: /turtle1_tf_broadcaster
Average rate: 62.710 Hz
Most recent transform: 1650822123.123 (0.003 sec old)
Buffer length: 4.896 sec
turtle2
turtle1

2.Create Package

The following operation are going to create the package:

1) Enter “**cd catkin_ws/src/**” command and press “Enter” to come to the workspace.

2) Enter “**catkin_create_pkg tf_hiwonder rospy std_msgs**” command and press “Enter” to create package.

3.Programming Method

3.1 Write Broadcasting and Listening Programs

1) Open the terminal.

2) Enter “**roscd tf_hiwonder**” command to enter the package directory and press “Enter”.

3) Enter “**mkdir scripts**” command and press “Enter” to create a new folder “scripts” where Python scripts are stored.

4) Enter “**cd scripts/**” command and press “Enter” to enter the folder “scripts” where Python scripts are stored.

5) Enter “**vi turtle_tf_broadcaster.py**” command to edit program and copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will request /show_person service and the service data type is learning_service::Person

import tf
import rospy
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()
    br.sendTransform((msg.x, msg.y, 0), tf.transformations.quaternion_from_euler(0, 0, msg.theta),
                     rospy.Time.now(), turtlename, "world")

if __name__ == '__main__':
    rospy.init_node('turtle_tf_broadcaster')
    turtlename = rospy.get_param('~turtle')
    rospy.Subscriber('/%s/pose' % turtlename, turtlesim.msg.Pose, handle_turtle_pose, turtlename)
    rospy.spin()
```

7) Enter “**vi turtle_tf_listener.py**” command to edit program and copy the following program. If need to modify, you can press “i”. After modifying, press “i” and enter “:wq” to save and exit.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will request /show_person service and the service data type is learning_service::Person

import roslib
import rospy
import math
import tf
import geometry_msgs.msg
import turtlesim.srv

if __name__ == '__main__':
    rospy.init_node('turtle_tf_listener')
    listener = tf.TransformListener()
    rospy.wait_for_service('spawn')
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    spawner(4, 2, 0, 'turtle2')
    turtle_vel = rospy.Publisher('turtle2/cmd_vel', geometry_msgs.msg.Twist,queue_size=1)
    rate = rospy.Rate(10.0)

    while not rospy.is_shutdown():
        try:
            (trans,rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))
        except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
            continue

        angular = 4 * math.atan2(trans[1], trans[0])
        linear = 0.5 * math.sqrt(trans[0] ** 2 + trans[1] ** 2)
        cmd = geometry_msgs.msg.Twist()
        cmd.linear.x = linear
        cmd.angular.z = angular
        turtle_vel.publish(cmd)

        rate.sleep()
```

8) Enter “**chmod +x turtle_tf_broadcaster.py**” and “**chmod +x turtle_tf_listener.py**” command, and then press “Enter” to give the executable permission to the files.

9) Enter “**cd ..**” and “**mkdir launch**” command to create a new folder “launch” where the launch scripts are stored.

10) Enter “**cd launch/**” command and press “Enter” to enter the folder “launch” where Python scripts are stored.

11) Enter “**vi start_tf_demo_py.launch**” command to edit program, and then copy the following program. If want to modify, you can press “i”. After modifying, press “Esc” and enter “:wq” to save and exit.

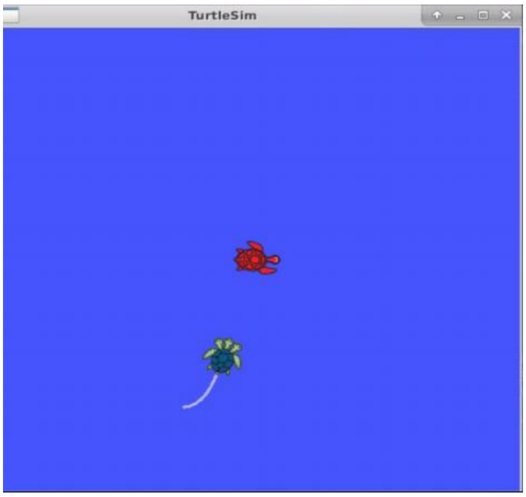
```
<launch>
            <!-- Turtlesim Node-->
            <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
            <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>

            <node name="turtle1_tf_broadcaster" pkg="tf_hiwonder" type="turtle_tf_broadcaster.py" respawn="false"
output="screen" >
              <param name="turtle" type="string" value="turtle1" />
            </node>

            <node name="turtle2_tf_broadcaster" pkg="tf_hiwonder" type="turtle_tf_broadcaster.py" respawn="false"
output="screen" >
              <param name="turtle" type="string" value="turtle2" />
            </node>

        <node pkg="tf_hiwonder" type="turtle_tf_listener.py" name="listener" />

</launch>
```

3.2 Run Program

1) Enter “**source ./devel/setup.bash**” command and press “Enter” to set the working environment.

2) Enter “**roslaunch tf_hiwonder start_tf_demo_py.launch**” and press “Enter” to run launch program.

A turtle automatically moves to the position of another turtle, as the figure shown below.



3) If want to stop program, you can press “Ctrl+C”.

Lesson 14 Common Visualization Tools

1. A Introduction to RQT Tool

1.1 Overview

RQT is a graphical user interface framework that implements various tools and interfaces in the form of plugins.

One can run all the existing GUI tools as dockable windows within RQT. When in use, RQT tools and plugins can be ran with command “rqt”. This GUI allows you to choose any available plugins on your system. In addition, you can also run plugins in standalone window.

1.2 RQT Component Structure

RQT consists of three metapackages：

1) Rqt: core infrastucture modules

2) rqt_common_plugins: back-end tool for building

3) rqt_robot_plugins: tool for interacting with robots

1.3 Advantage of RQt framework

Compared to building your own GUIs from scratch:

1) Standardized common procedures for GUI (start-shutdown hook, restore previous states).

2) Multiple widgets can be docked in a single window.

3) Easily turn your existing Qt widgets into RQt plugins.

4) Expect support at ROS Answers (ROS community website for the questions).

From system architecture's perspective:

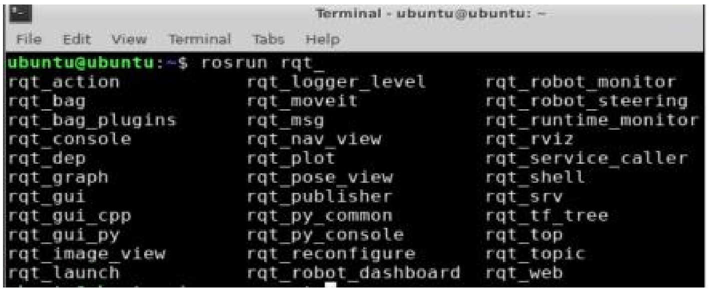
1) Support multi-platformand multi-language (Python, C++).

2) Manageable lifecycle: RQt plugins using common API makes maintainance and reuse easier.

2. RQT Running

Note: After ROS is installed successfully, it comes with RQT tool, no need to reinstall.

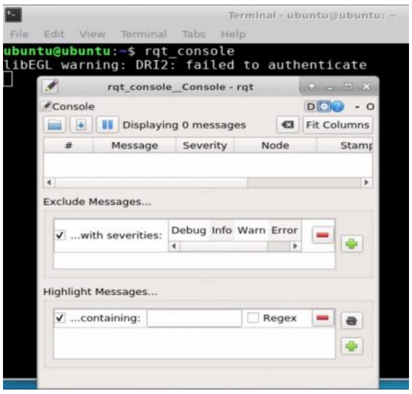
1) Open the terminal, and then enter “**rosrun rqt_**” and press “Tab” key to unlist the following command:


Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help

```
ubuntu@ubuntu:~$ rosrun rqt_
rqt_action            rqt_logger_level      rqt_robot_monitor
rqt_bag               rqt_moveit            rqt_robot_steering
rqt_bag_plugins       rqt_msg               rqt_runtime_monitor
rqt_console           rqt_nav_view          rqt_rviz
rqt_dep               rqt_plot              rqt_service_caller
rqt_graph             rqt_pose_view         rqt_shell
rqt_gui               rqt_publisher         rqt_srv
rqt_gui_cpp           rqt_py_common         rqt_tf_tree
rqt_gui_py            rqt_py_console        rqt_top
rqt_image_view        rqt_reconfigure       rqt_topic
rqt_launch            rqt_robot_dashboard   rqt_web
```

rqt_image_view is used to display the returned image.

1) rqt_console：Log output tool


Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help
rqt_console__Console - rqt
Console
D - O
Fit Columns

```
ubuntu@ubuntu:~$ rqt_console
libEGL warning: DRI2: failed to authenticate
```

Displaying 0 messages

#	Message	Severity	Node	Stamp

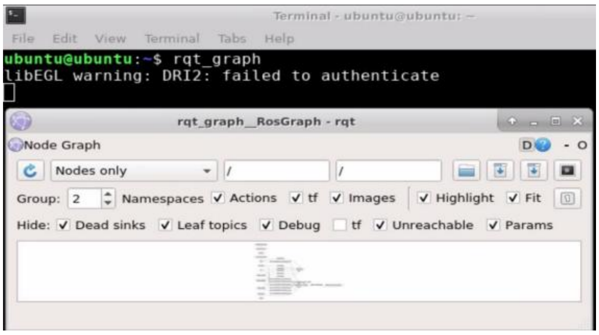
Exclude Messages...

☑ ...with severities: Debug Info Warn Error

Highlight Messages...

☑ ...containing: □ Regex

2) graph visualization tool



```
ubuntu@ubuntu:~$ rqt_graph
libEGL warning: DRI2: failed to authenticate
```

rqt_graph__RosGraph - rqt

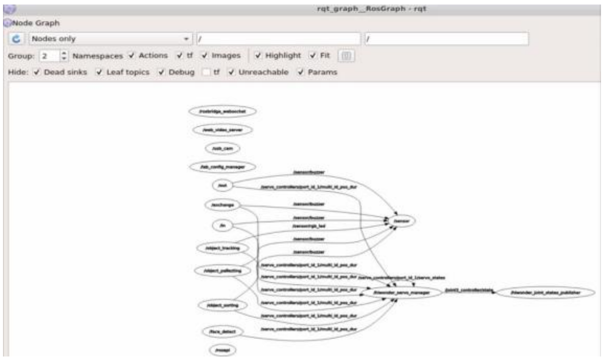
Node Graph

Nodes only

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug □ tf ☑ Unreachable ☑ Params

Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help



rqt_graph__RosGraph - rqt

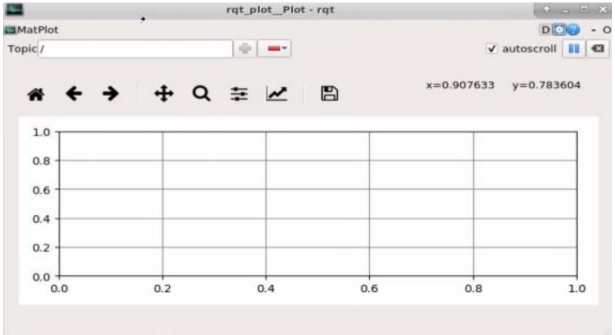
Node Graph

Nodes only

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug □ tf ☑ Unreachable ☑ Params

3) rqt_plot： graph visualization tool



rqt_plot__Plot - rqt

MatPlot

Topic /

☑ autoscroll

x=0.907633 y=0.783604

3. Plugins Function Introduction

Some useful plugins are enumerated in the following table:

Plugins	Function Instruction
topics monitor	Monitor the current transmisson data of a topic, bandwidth consumption, topic frequency, etc, which is equivalent to the original rostopic echo msg_name
message publisher	Publish a topic with a custom name as well as specify the message type, publishing data and publishing frequency of the topic
message type browser	View all currently defined message types including own defined msg, which is basically equivalent to the function of rosmsg show msg_name
robot steering	Publish a topic cmd_vel to publish Twist topic message, which can visually modify the speed, angle variables. It is convenient for testing some control commands conveniently
bag	Record a bag and arbitrarily choose and specify which topic to record. It can also open a bag, in which you can easily control the play or pause of the bag play, and specify the previous and next frames to be played.

The plugins in bag are shown in the following table:

Plugins	Function Instruction
node_graph	View all the nodes running in current node
process monitor	View all current nodes and PID, CPU and RAM usage of node
launch	Easily choose package and launch files in visualization interface, run and stop a node of lauch file

image view	Easily view the image messages delivered in ROS topic, which is convenient for us to observe the images that the robot is looking at.
plot	The data of a topic (all or part of the data) can be displayed on a graph, so that we can see the changes of the topic messages more visually, which is convenient for us to debug
tf tree	Display the structure of current tf tree
rviz	rivz is also integrated in rqt, which is convenient for us to open rviz tool in here