# 机器人操作系统（**ROS**）及仿真应用

**第 2 章、第 3 章、第 4 章源代码**

# 第 2 章　ROS 安装与系统架构

P22：

http://wiki.ros.org/noetic/Installation/Ubuntu

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

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

```
http://wiki.ros.org/noetic/Installation/Ubuntu
```

```
sudo apt-get update
```

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

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

P26：

```
echo $ROS_PACKAGE_PATH
```

```
mkdir -p ~/catkin_ws/src        #在主文件夹下创建 catkin_ws/src 空文件夹
cd ~/catkin_ws/src               #进入 src 文件夹
catkin_init_workspace           #初始化工作空间
```

```
cd ~/catkin_ws/                 #回到 catkin_ws 文件夹
catkin_make                      #进行编译
```

P27：

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

```
cd ~/catkin_ws/src
catkin_create_pkg test std_msgs rospy roscpp
```

```
 cd ~/catkin_ws/
 catkin_make
```

P28：

```
rosstack find ros_tutorials
```

P29：

```
roscore
```

```
 rosrun turtlesim turtlesim_node
```

```
 rosrun turtlesim turtle_teleop_key
```

```
rosrun rqt_graph rqt_graph
```

P30：

http://wiki.ros.org/IDEs

下载地址：https://code.visualstudio.com/Download

```
sudo dpkg -i code_xxxx_amd64.deb
```

# 第 3 章  ROS 通信方式

P37：

```
	cd catkin_ws/src/
```

```
	catkin_create_pkg turtle_vel_ctrl_pkg roscpp geometry_msgs
```

P38：

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>

int main(int argc, char** argv)
{
	ros::init(argc, argv, "turtle_vel_ctrl_node");
	ros::NodeHandle n;
	ros::Publisher vel_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 20);

while(ros::ok())
	{
		geometry_msgs::Twist vel_cmd;
		vel_cmd.linear.x = 0.1;
		vel_cmd.linear.y = 0;
		vel_cmd.linear.z = 0;
		vel_cmd.angular.x = 0;
		vel_cmd.angular.y = 0;
		vel_cmd.angular.z = 0;
		vel_pub.publish(vel_cmd);
		ros::spinOnce();
	}
	return 0;
}
```

P40：

```
add_executable(turtle_vel_ctrl_node src/turtle_vel_ctrl_node.cpp)
add_dependencies(turtle_vel_ctrl_node    ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(turtle_vel_ctrl_node ${catkin_LIBRARIES})
```

```
	cd
	cd catkin_ws/
```

P41：

```
	catkin_make
```

```
cd
roscore
```

P42：

```
rosrun turtlesim turtlesim_node
```

```
rosrun turtle_vel_ctrl_pkg turtle_vel_ctrl_node
```

```
vel_cmd.linear.x = 2;
vel_cmd.linear.y = 0;
vel_cmd.linear.z = 0;
vel_cmd.angular.x = 0;
vel_cmd.angular.y = 0;
vel_cmd.angular.z = 1.8;
```

P43：

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
int main(int argc, char** argv)
{
ros::init(argc, argv, "turtle_vel_ctrl_node");
ros::NodeHandle n;
ros::Publisher vel_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 20);
ros::Rate loopRate(2);          //与 Rate::sleep()配合指定自循环频率
int count= 0;
while(ros::ok())
{
    geometry_msgs::Twist vel_cmd;
    vel_cmd.linear.x   = 1;
    vel_cmd.linear.y   = 0;
    vel_cmd.linear.z   = 0;
    vel_cmd.angular.x = 0;
    vel_cmd.angular.y = 0;
    vel_cmd.angular.z = 0;
    count++;
    while(count==5)
    {
```

```
                count=0;
                vel_cmd.angular.z = 3.1415926;
            }
            vel_pub.publish(vel_cmd);
            ros::spinOnce();
            loopRate.sleep();       //按 loopRate(2)设置的 2HZ 将程序挂起
        }
        return 0;
}
```

P44：

```
        rqt_graph
```

P45：

```
        #include <ros/ros.h>
        #include <geometry_msgs/Twist.h>

        void callback(const geometry_msgs::Twist& cmd_vel)
        {
            ROS_INFO("Received a /cmd_vel message!");
            ROS_INFO("Linear Velocity:[%f,%f,%f]",
                    cmd_vel.linear.x,cmd_vel.linear.y,cmd_vel.linear.z);
            ROS_INFO("Angular Velocity:[%f,%f,%f]",
                    cmd_vel.angular.x,cmd_vel.angular.y,cmd_vel.angular.z);
        }

        int main(int argc, char** argv)
        {
            ros::init(argc, argv, "turtle_vel_rece_node");
            ros::NodeHandle n;
            ros::Subscriber sub = n.subscribe("/turtle1/cmd_vel", 1000, callback);
            ros::spin();
            return 1;
        }
```

P47：

```
add_executable(turtle_vel_rece_node src/turtle_vel_rece_node.cpp)
add_dependencies(turtle_vel_rece_node    ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(turtle_vel_rece_node ${catkin_LIBRARIES})
```

P48：

```
        cd
```

```
cd catkin_ws/
```

```
catkin_make
```

```
cd
roscore
```

P49：

```
rosrun turtlesim turtlesim_node
```

```
rosrun turtle_vel_ctrl_pkg turtle_vel_rece_node
```

```
rosrun turtle_vel_ctrl_pkg turtle_vel_ctrl_node
```

```
rqt_graph
```

P51：

```
cd catkin_ws/src/
```

```
catkin_create_pkg service_client_pkg roscpp std_msgs
```

P52：

```
cd    service_client_pkg
mkdir srv
```

P53：

```
string name
---
bool in_class
bool boy
int32 age
string personality
```

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

P54：

```
#find_package(catkin REQUIRED COMPONENTS
  #roscpp
  #std_msgs
#)
```

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  message_generation
  std_msgs
  std_srvs
)
```

```
#add_service_files(
   #FILES
   #Service1.srv
   #Service2.srv
 #)
```

```
add_service_files(
    FILES
    ServiceClientExMsg.srv
)
```

```
# generate_messages(
#     DEPENDENCIES
#     std_msgs
# )
```

```
generate_messages(
    DEPENDENCIES
    std_msgs
)
```

P55：

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

```
catkin_package(
   INCLUDE_DIRS include
   LIBRARIES service_client_pkg
   CATKIN_DEPENDS roscpp std_msgs
   DEPENDS system_lib
   CATKIN_DEPENDS message_runtime
)
```

```
cd
cd catkin_ws/
catkin_make
```

```
#include <service_client_pkg/ServiceClientExMsg.h>
```

P57：

```
#include <ros/ros.h>
#include <service_client_pkg/ServiceClientExMsg.h>
#include <iostream>
#include <string>
using namespace std;

bool infoinquiry (service_client_pkg::ServiceClientExMsgRequest& request,
               service_client_pkg::ServiceClientExMsgResponse& response)
{
ROS_INFO("callback activated");
string input_name(request.name);
response.in_class=false;

            if (input_name.compare("Tom")==0)
            {
                  ROS_INFO("Student infomation about Tom");
                  response.in_class=true;
                  response.boy=true;
                  response.age = 20;
                  response.personality="outgoing";
}
if (input_name.compare("Mary")==0)
            {
                  ROS_INFO("Student infomation about Mary");
                  response.in_class=true;
                  response.boy=false;
```

```
                        response.age = 21;
                        response.personality="introverted";
                }
                return true;
        }

        int main(int argc, char **argv)
        {
                ros::init(argc, argv, "service_example_node");
                ros::NodeHandle n;
                ros::ServiceServer service = n.advertiseService("info_inquiry_byname", infoinquiry);
            ROS_INFO("Ready to inquiry names.");
                ros::spin();
                return 0;
        }
```

P58：

```
        add_executable(service_example_node src/service_example_node.cpp)
        add_dependencies(service_example_node
${${PROJECT_NAME}_EXPORTED_TARGETS}
            ${catkin_EXPORTED_TARGETS})
        target_link_libraries(service_example_node ${catkin_LIBRARIES})
```

P59：

```
        cd
        cd catkin_ws/
```

```
        catkin_make
```

```
        cd
        roscore
```

P60：

```
        rosrun service_client_pkg service_example_node
```

```
        rosservice call info_inquiry_byname 'Tom'
```

P61：

```
        #include <ros/ros.h>
        #include <service_client_pkg/ServiceClientExMsg.h>
```

```
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char **argv)
{
  ros::init(argc, argv, "client_example_node");
  ros::NodeHandle n;
ros::ServiceClient client = n.serviceClient<service_client_pkg::ServiceClientExMsg>
                                    ("info_inquiry_byname");
                service_client_pkg::ServiceClientExMsg srv;
        string input_name;

while (ros::ok())
              {
                              cout<<endl;
                              cout << "enter a name (q to quit): ";
                              cin>>input_name;
                              if (input_name.compare("q")==0)
                             {        return 0;       }
                              srv.request.name = input_name;
                              if (client.call(srv))
                             {
                                          if (srv.response.in_class)
                                         {
                                                      if (srv.response.boy)
                                                     {   cout << srv.request.name << " is boy;" << endl;   }
                                                      else
                                                     {   cout << srv.request.name << " is girl;" << endl; }
                                                      cout << srv.request.name << " is " << srv.response.age
<< " years old;" << endl;
                                                      cout    <<    srv.request.name    <<    "    is    "    <<
srv.response.personality <<"."<< endl;
                                          }
                                          else
                                         {             cout  <<  srv.request.name  <<  "  is  not  in  class"  <<
endl;     }
```

```
            }
                              else
                              {
                                           ROS_ERROR("Failed to call service info_inquiry_byname");
                                           return 1;
                               }
                  }
                  return 0;
      }
```

P63：

```
      add_executable(client_example_node src/client_example_node.cpp)
      add_dependencies(client_example_node
${${PROJECT_NAME}_EXPORTED_TARGETS}
         ${catkin_EXPORTED_TARGETS})
      target_link_libraries(client_example_node ${catkin_LIBRARIES})
```

P64：

```
      cd
      cd catkin_ws/
```

```
      catkin_make
```

```
      cd
      roscore
```

```
      rosrun service_client_pkg service_example_node
```

```
      rosrun service_client_pkg client_example_node
```

P66：

```
      cd catkin_ws/src/
```

```
      catkin_create_pkg actionlib_example_pkg roscpp actionlib actionlib_msgs
```

P67：

```
      cd    actionlib_example_pkg
      mkdir action
```

P68：

```
#goal definition
int32 whole_distance
```

```
---
#result definition
bool is_finish
---
#feedback
int32 moving_meter
```

P69：

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

```
## Generate actions in the 'action' folder
# add_action_files(
#     FILES
#     Action1.action
#     Action2.action
# )
```

```
## Generate actions in the 'action' folder
  add_action_files(
    FILES
    ActionlibExMsg.action
  )
```

```
## Generate added messages and services with any dependencies listed here
# generate_messages(
#     DEPENDENCIES
#     std_msgs    # Or other packages containing msgs
# )
```

P70：

```
## Generate added messages and services with any dependencies listed here
  generate_messages(
    DEPENDENCIES
    actionlib_msgs                                   # Or other packages containing msgs
  )
```

```
cd
cd catkin_ws/
catkin_make
```

P71：

```
    #include <actionlib_example_pkg/ActionlibExMsgaction.h>
```

P72：

```
    #include <actionlib/client/simple_action_client.h>
    #include <actionlib_example_pkg/ActionlibExMsgAction.h>

    //action 完成后调用此函数
    void doneCb(const actionlib::SimpleClientGoalState& state,const actionlib_example_pkg::
                                          ActionlibExMsgResultConstPtr& result)
    {
        ROS_INFO("Task completed!");
        ros::shutdown();      //任务完成之后关闭节点
    }
    void activeCb()         //action 的目标任务发送给 server 且开始执行时，调用此函数
    {
       ROS_INFO("Goal is active! The robot begin to move forward.");
    }
    //action 任务在执行过程中，server 对过程有反馈则调用此函数
    void     feedbackCb(const     actionlib_example_pkg::ActionlibExMsgFeedbackConstPtr&
feedback)
    {
        //将服务器的反馈输出（机器人向前行进到第几米）
        ROS_INFO("The robot has moved forward %d meter:", feedback->moving_meter);
    }

    int main(int argc, char** argv)
    {
        ros::init(argc, argv, "actionlib_client_node");

        //创建一个 action 的 client，指定 action 名称为”moving_forward”
    actionlib::SimpleActionClient< actionlib_example_pkg::ActionlibExMsgAction> client
                                                      ("moving_forward",true);
        ROS_INFO("Waiting for action server to start");
        client.waitForServer();//等待服务器响应
        ROS_INFO("Action server started");
        //创建一个目标：移动机器人前进 10 米
        actionlib_example_pkg::ActionlibExMsgGoal goal;
        goal.whole_distance = 10;

        //把 action 的任务目标发送给服务器，绑定上面定义的各种回调函数
        client.sendGoal(goal,&doneCb,&activeCb,&feedbackCb);
        ros::spin();
        return 0;
    }
```

P73：

```
    add_executable(actionlib_client_node src/actionlib_client_node.cpp)
    add_dependencies(actionlib_client_node
${${PROJECT_NAME}_EXPORTED_TARGETS}
      ${catkin_EXPORTED_TARGETS})
    target_link_libraries(actionlib_client_node ${catkin_LIBRARIES})
```

P74：

```
    cd
    cd catkin_ws/
```

```
    catkin_make
```

P75：

```
#include <ros/ros.h>
#include <actionlib/server/simple_action_server.h>
#include <actionlib_example_pkg/ActionlibExMsgAction.h>

//服务器接受任务目标后，调用该函数执行任务
void execute(const actionlib_example_pkg:: ActionlibExMsgGoalConstPtr& goal,actionlib::
            SimpleActionServer< actionlib_example_pkg:: ActionlibExMsgAction>* as)
{
    ros::Rate r(0.5);
    actionlib_example_pkg:: ActionlibExMsgFeedback feedback;
    ROS_INFO("Task: The robot moves forward %d meters.", goal->whole_distance);
    for(int i=1; i<=goal-> whole_distance; i++)
    {
        feedback.moving_meter = i;
        as->publishFeedback(feedback); //反馈任务执行的过程
        r.sleep();
    }

    ROS_INFO("Task completed!");
    as->setSucceeded();}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "actionlib_server_node");
    ros::NodeHandle n;
    //创建一个 action 的 server，指定 action 名称为”moving_forward”
    actionlib::SimpleActionServer<actionlib_example_pkg::ActionlibExMsgAction> server(n,
                                "moving_forward",boost::bind(&execute, _1, &server), false);
    //服务器启动
    server.start();
    ros::spin();
    return 0;
}
```

P76：

```
    add_executable(actionlib_server_node src/actionlib_server_node.cpp)
    add_dependencies(actionlib_server_node
${${PROJECT_NAME}_EXPORTED_TARGETS}
      ${catkin_EXPORTED_TARGETS})
    target_link_libraries(actionlib_server_node ${catkin_LIBRARIES})
```

P77：

```
    cd
    cd catkin_ws/
```

```
    catkin_make
```

P78：

```
    cd

    roscore
```

```
    rosrun actionlib_example_pkg actionlib_client_node
```

```
    rosrun actionlib_example_pkg actionlib_server_node
```

```
rqt_graph
```

P79：

```
cd catkin_ws/src/
```

```
catkin_create_pkg parameter_server_pkg roscpp std_msgs
```

P80：

```
kinect_height: 0.34
kinect_pitch: 1.54
```

P81：

```
roscore
```

P82：

```
cd catkin_ws/src/parameter_server_pkg/launch
```

```
rosparam load para_setting.yaml
```

```
rosparam dump para_setting.yaml
```

P83：

```
<launch>
<rosparam command="load" file="$(find parameter_server_pkg)/launch/para_setting.yaml" />
</launch>
```

```
roslaunch parameter_server_pkg para_load.launch
```

```
rosparam list
```

P86：

```
#include <ros/ros.h>

int main(int argc, char **argv)
{
ros::init(argc, argv, "get_parameter_node");
  ros::NodeHandle nh; //  节点句柄
double kinect_height_getting, kinect_pitch_getting;//定义变量

            if (nh.getParam("/kinect_height", kinect_height_getting))
{
                  ROS_INFO("kinect_height set to %f", kinect_height_getting);
```

```
                 }
                 else
                {
                      ROS_WARN("could not find parameter value / kinect_height on parameter
server");
                }
                  if (nh.getParam("/kinect_pitch", kinect_pitch_getting))
      {
                       ROS_INFO("kinect_pitch set to %f", kinect_pitch_getting);
                 }
                 else
                {
                      ROS_WARN("could not find parameter value / kinect_pitch on parameter
server");
                }
      }
```

P87：

```
      add_executable(get_parameter_node src/get_parameter_node.cpp)
      add_dependencies(get_parameter_node
${${PROJECT_NAME}_EXPORTED_TARGETS}
         ${catkin_EXPORTED_TARGETS})
      target_link_libraries(get_parameter_node ${catkin_LIBRARIES})
```

```
      cd

      cd catkin_ws/
```

```
      catkin_make
```

P88：

```
      roslaunch parameter_server_pkg para_load.launch
```

P89：

```
      rosrun parameter_server_pkg get_parameter_node
```

# 第 4 章  ROS 实用工具

P91：

```
    cd catkin_ws/src/
    catkin_create_pkg tf_test_pkg roscpp tf geometry_msgs
```

P92：

```
    #include <ros/ros.h>
    #include <tf/transform_broadcaster.h>

    int main(int argc, char** argv)
    {
    ros::init(argc, argv, "tf_broadcaster");
    ros::NodeHandle n;
        ros::Rate loop_rate(100);

        tf::TransformBroadcaster broadcaster;
        tf::Transform base_link2base_laser;
        base_link2base_laser.setOrigin(tf::Vector3(0.1, 0.0, 0.2));
        base_link2base_laser.setRotation(tf::Quaternion(0, 0, 0, 1));

  while(n.ok())
    {
            broadcaster.sendTransform(tf::StampedTransform(
                          base_link2base_laser,ros::Time::now(),"base_link","base_laser"))
;
        //broadcaster.sendTransform(tf::StampedTransform(tf::Transform(tf::Quaternion(0, 0, 0,
                        0), //tf::Vector3(1, 0.0, 0)),ros::Time::now(),"base_link",
                        "base_laser"));
            loop_rate.sleep();
        }
        return 0;
    }
```

P93：

```
add_executable(tf_broadcaster src/tf_broadcaster.cpp)
add_dependencies(tf_broadcaster   ${${PROJECT_NAME}_EXPORTED_TARGETS}
  ${catkin_EXPORTED_TARGETS})
target_link_libraries(tf_broadcaster ${catkin_LIBRARIES})
```

P95：

```
    cd
    cd catkin_ws/
```

```
    catkin_make
```

P96：

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/PointStamped.h>
#include <iostream>

int main(int argc,char** argv)
{
   ros::init(argc,argv,"tf_listener");
   ros::NodeHandle n;
      ros::Rate loop_rate(100);

      tf::TransformListener listener;
      geometry_msgs::PointStamped laser_pos;
      laser_pos.header.frame_id = "base_laser";
      laser_pos.header.stamp = ros::Time();

      laser_pos.point.x =0.3;
      laser_pos.point.y = 0;
      laser_pos.point.z = 0;

      geometry_msgs::PointStamped base_pos;

   while(n.ok())
      {
             if (listener.waitForTransform("base_link","base_laser",ros::Time(0),ros::Duration(3)))
            {
                  listener.transformPoint("base_link",laser_pos,base_pos);
                  ROS_INFO("pointpos in base_laser: (%.2f, %.2f. %.2f) ", laser_pos.point.x,
                                                                    laser_pos.point.y, laser_pos.point.z);
                  ROS_INFO("pointpos in base_link: (%.2f, %.2f, %.2f) ", base_pos.point.x,
                                                                    base_pos.point.y, base_pos.point.z);

                  tf::StampedTransform laserTransform;
                  listener.lookupTransform("base_link","base_laser", ros::Time(0), laserTransform);
```

```
                std::cout << "laserTransform.getOrigin().getX(): " <<
                                                                                        laserTransform.
                                                                                        getOrigin().get
                                                                                        X() <<
                                                                                        std::endl;
                std::cout << "laserTransform.getOrigin().getY(): " <<
                                                                                        laserTransform.
                                                                                        getOrigin().get
                                                                                        Y() <<
                                                                                        std::endl;
                std::cout << "laserTransform.getOrigin().getZ(): " <<
                                                                                        laserTransform.
                                                                                        getOrigin().get
                                                                                        Z() <<
                                                                                        std::endl;
            }
            loop_rate.sleep();
        }
}
```
P98：

```
add_executable(tf_listener src/tf_listener.cpp)
add_dependencies(tf_listener    ${${PROJECT_NAME}_EXPORTED_TARGETS}
   ${catkin_EXPORTED_TARGETS})
target_link_libraries(tf_listener ${catkin_LIBRARIES})
```

```
    cd
    cd catkin_ws/
```

```
    catkin_make
```
P99：

```
    cd
    roscore
```

```
     rosrun tf_test_pkg tf_broadcaster
```

```
     rosrun tf_test_pkg tf_listener
```

```
rqt_graph
```

P100：

```
rosrun rqt_tf_tree rqt_tf_tree
```

P101：

```
rosrun tf tf_echo /base_link base_laser
```

P102：

```
roscore
rosrun turtlesim turtlesim_node
rosrun turtlesim turtle_teleop_key
```

```
cd catkin_ws/src/
```

```
catkin_create_pkg launch_test_pkg
```

P103：

```
cd launch_test_pkg
mkdir launch
```

```
cd launch
touch turtle_key_control.launch
```

```
gedit turtle_key_control.launch
```

```
<launch>
    <node pkg="turtlesim" name="turtle1" type="turtlesim_node"/>
    <node pkg="turtlesim" name="turtle1_key" type="turtle_teleop_key"/>
</launch>
```

```
cd
cd catkin_ws/
```

```
catkin_make
```

```
cd
roslaunch launch_test_pkg turtle_key_control.launch
```

P105：

```
    <launch>

        <node pkg="mrobot_bringup" name="mrobot_bringup" type="mrobot_bringup" output="screen" />

        <arg name="urdf_file" default="$(find xacro)/xacro --inorder '$(find mrobot_description)/urdf/mrobot_with_rplidar.urdf.xacro'" />
        <param name="robot_description" command="$(arg urdf_file)" />

        <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />

        <node pkg="robot_state_publisher" name="state_publisher" type="robot_state_publisher">
            <param name="publish_frequency" type="double" value="5.0" />
        </node>
        <node pkg="tf" name="base2laser" type="static_transform_publisher" args="0 0 0 0 0 0 1 /base_link /laser 50"/>

        <node pkg="robot_pose_ekf" name="robot_pose_ekf" type="robot_pose_ekf">
            <remap from="robot_pose_ekf/odom_combined" to="odom_combined"/>
            <param name="freq" value="10.0"/>
            <param name="sensor_timeout" value="1.0"/>
            <param name="publish_tf" value="true"/>
            <param name="odom_used" value="true"/>
            <param name="imu_used" value="false"/>
            <param name="vo_used" value="false"/>
            <param name="output_frame" value="odom"/>
        </node>
        <include file="$(find mrobot_bringup)/launch/rplidar.launch" />
    </launch>
```

P106：

```
    <node pkg="pkg-name" name="node-name" type="executable-name" output="log|screen" />
```

```
    <param name="param-name" value="param-value"/>
```

```
<param name="freq" value="10.0"/>
```

P107：

```
<rosparam command="load" value="path-to-param-file"/>
```

```
<arg name="arg-name" default="arg-value"/>
```

```
<arg name="demo" value="123"/>
<arg name="demo" default="123"/>
```

```
roslaunch pkg-name launch-file-name demo:=456
```

```
<remap from=" orig-topic-name" to="new-topic-name"/>
```

```
<remap from="chatter" to="demo/chatter"/>
```

```
<include file="$(find pkg-name)/launch/launch-file-name" ns="namespace" />
```

P108：

```
sudo apt-get install git
```

```
cd catkin_ws/src
git clone https://github.com/6-robot/wpr_simulation.git
```

P109：

```
~/catkin_ws/src/wpr_simulation/scripts/install_for_noetic.sh
```

```
sudo apt-get install ros-noetic-navigation
```

```
cd ~/catkin_ws
catkin_make
```

P110：

```
roslaunch wpr_simulation wpb_simple.launch
```

P112：

```
rosrun wpr_simulation keyboard_vel_ctrl
```

P113：

```
cd catkin_ws/src
git clone https://github.com/6-robot/wpb_home.git
```

```
sudo apt-get install ros-noetic-joy
```

```
sudo apt-get install ros-noetic-sound-play
```

P114：

```
cd
cd    catkin_ws
catkin_make
```

```
roslaunch wpr_simulation wpb_simple.launch
```

```
roslaunch wpr_simulation wpb_rviz.launch
```