《机器人编程基础》

实验报告

实验题目：**Robot OS 实验**

姓名、学号：成子谦，201730681303

学院、班级：软件学院 2017 级 1 班

华南理工大学软件学院

二〇一九年十一月

目　录

1. 任务一：ROS 基础实验

1.1. 安装 ROS

1.1.1. 安装 Ubuntu 18.04 LTS 并设置软件源（与实验无关，此处不赘述）

1.1.2. 设置 ROS 软件源

打开终端，执行以下代码。

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

1.1.3. 设置 ROS 软件源验证 key

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

1.1.4. 更新软件包列表并更新系统

```
sudo apt update
sudo apt upgrade
```

1.1.5. 安装 ROS

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

1.1.6. 初始化 ROS

```
sudo rosdep init
rosdep update
```

1.1.7. 设置 ROS 环境变量

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

1.1.8. 安装其他所需软件包

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

这样，我们就完成了 ROS 系统的安装。我们可以通过检查环境变量的方式来确定是否安装成功。

```
printenv | grep ROS
```

可以看到终端输出以下信息：

```
ROS_ROOT=/opt/ros/kinetic/share/ros
ROS_PACKAGE_PATH=/opt/ros/kinetic/share
ROS_MASTER_URI=http://localhost:11311
ROSLISP_PACKAGE_DIRECTORIES=
ROS_DISTRO=kinetic
ROS_ETC_DIR=/opt/ros/kinetic/etc/ros
```

1.2. 运行 ROS

1.2.1. 运行 roscore

```
roscore&
```

会打印如下信息。

```
PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.7
NODES
auto-starting new master
process[master]: started with pid [3829]
ROS_MASTER_URI=http://youwu-OptiPlex-7040:11311/
setting /run_id to 416e84ee-3ef6-11e7-96ab-1866da35b5cb
process[rosout-1]: started with pid [3842]
started core service [/rosout]
```

1.2.2. 运行 turtlesim_node

```
rosrun turtlesim turtlesim_node
```

会弹出如下窗口

每次运行该程序，小乌龟形状并不一样。

1.2.3. 运行 turtle_teleop_key

```
rosrun turtlesim turtle_teleop_key
```

鼠标点击运行 turtle_teleop_key 的终端，将其作为当前活动窗口，从而正确接收键盘按键。通过方向键控制小乌龟运动：上，小乌龟超前运动；下，小乌龟向后

运动；左，小乌龟左转；右，小乌龟右转。小乌龟运动后在界面上会留下运动轨迹，如下图所示。

1.2.4. 控制小乌龟画圆

新建 terminal，执行如下命令。

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

消息的 topic 为"/turtle1/cmd_vel"，消息的类型为"geometry_msgs/Twist"，消息的内容为" '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' "，"-1"表示"rostopic pub"节点发送一次信息之后就停止，"--"表示分隔符，分隔符之后的参数均为消息的值。发布消息之后，可以在 turtlesim_node 的窗口观察到小乌龟的运动轨迹，如下图所示。

输入以下命令，以 1Hz 的频率循环发布消息。

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

其中，"-r"表示循环发布，"1"表示以 1Hz 的频率周期发送。

1.2.5. 清除小乌龟轨迹

使用 rosservice call 可以从终端调用 node 提供的服务。命令格式如下：

```
rosservice call [service] [args]
```

例如，调用/clear，清除小乌龟运动轨迹服务。

```
rosservice call /clear
```

1.2.6. 修改 turtlesim 背景

使用 rosparam set 可以修改 parameter server 中参数的值。输入以下命令：

```
rosparam set /background_b 200
rosparam set /background_g 200
rosparam set /background_r 200
rosservice call /clear
```

可以把背景颜色设为浅灰色。

1.2.7. paramater server 参数导出与导入

导出参数命令格式如下：

```
rosparam dump [file_name] [namespace]
```

若 namespace 为空，默认存到根目录。

使用以下命令把 parameter server 所有参数存到 params.yaml 文件中，并查看 params.yaml 文件内容：

```
rosparam dump params.yaml
cat params.yaml
```

使用 rosparam load 可以将文件中的参数导入 parameter server：

```
rosparam load [file_name] [namespace]
```

若 namespace 为空，默认存到根目录。

修改之前保存参数的文件 params.yaml，使参数值与之前不同，可以将未修改的参数删除，也可添加新的参数。修改之后的 params.yaml 文件内容如下：

```
background_b: 100
background_g: 100
background_r: 100
hello：  1
```

将 params.yaml 文件中的参数导入 parameter server，修改根目录树下的参数值。并查看当前系统的参数。

```
rosparam load params.yaml
rosparam get /
```

1.2.8. 关于 ROS 工作空间

ROS 采用 catkin workspace 作为其工作空间。catkin 是 ROS 的编译系统，将源代码编译为终端可执行的目标程序。在 catkin workspace 中编写代码，有利于提高项目开发的效率、提高项目的可管理性。

首先创建工作区根目录、源代码目录：

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

切换到工作区根目录并使用 catkin_make 编译。

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

创建工作区之后，需要配置环境变量让 ROS 知道工作区的位置。可以执行工作区编译之后生成的脚本文件方便的修改环境变量。

```
echo “source /home/[user_name]/catkin_ws/devel/setup.bash” >> ~/.bashrc
source ~/.bashrc
```

其中/home/[user_name]/catkin_ws 为工作区所在路径。若需对当前终端有效，可直接修改环境变量并查看环境变量检查是否配置成功。

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

1.2.9. 关于 ROS 包

与 ROS workspace 一致，ROS package 也采用 catkin 格式的包。

catkin package 有三个格式要求：

Package 中必须含有一个遵循 catkin 编译格式的 package.xml 文件，用来描述 package 的元信息。Package 中必须含有一个遵循 catkin 编译格式的 CMakelist.txt 文件。同一个 package 目录下只允许存在一个 catkin package。一个最简单的 catkin package 包含一个 package.xml 文件和一个 CMakelist.txt 文件。编写的 catkin packages 存放于 catkin workspace 中。

ROS 提供了一个命令行工具 catkin_create_pkg，可以方便的创建 catkin package。常用的 catkin_create_pkg 命令格式如下：

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

package_name 是即将创建的 package 名字，depend1、depend2 等等是 package 需要的依赖包。输入以下命令创建 catkin package：beginner_tutorials，其直接依赖包为 std_msgs、rospy、roscpp。std_msgs 是标准信息包，rospy、roscpp 是客户库包，允许用不同语言编写的 package 之间可以相互传递数据。

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

创建 package 之后需要进行编译操作，切换到 catkin workspace 根目录使用 catkin_make 进行编译。

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

编译生成的文件会储存在 catkin_ws/build 目录下对应 package 名字的目录中。

1.2.10. 小乌龟模仿动作

通过编写 launch 引导文件，可以让 ROS 按照文件中描述的方式运行节点。功能文件通常存放于功能目录中。

在 beginner_tutorials 中新建 launch 目录，在 launch 目录中新建 turtlemimic.launch

文件。

```
roscd beginner_tutorials
mkdir launch
cd launch
touch turtlemimic.launch
```

填写 turtlemimic.launch 文件如下：

```
<launch>
  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>
  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>
  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>
</launch>
```

文件中定义了"turtlesim1"、"turtlesim2"两个 group 的命名空间，防止出现命名冲突。每个 group 中定义了一个节点，节点执行 package "turtlesim"中的可执行程序"turtlesim_node"，节点命名为"sim"。group 之外定义了一个节点，节点执行 package "turtlesim"中的可执行程序"mimic"，节点命名为"mimic"。重新映射节点的输入、输出，将"turtlesim1/turtle1"作为输入，"turtlesim2/turtle1"作为输出。在终端中 roslaunch 执行 launch 文件。

```
roslaunch beginner_tutorials turtlemimic.launch
```

此时会弹出两个仿真小乌龟的窗口。给小乌龟 1 发送运动指令，让小乌龟 1 作匀速圆周运动。新建终端，输入以下命令：

```
rostopic pub /turtlesim1/turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

此时小乌龟 1 作圆周运动，小乌龟 2 会模仿小乌龟 1 作相同的动作。

1.2.11. 关于 ROS 发布者和订阅者

切换到 beginner_tutorials 目录，建立 src 目录。发布者 Publisher 也称为 talker。在 src 目录中建立 talker.cpp 文件，并编写 talker.cpp 程序。

```
roscd beginner_tutorials
mkdir -p src
cd src
touch talker.cpp
```

订阅者 Subscriber 也称为 listener。在 src 目录中建立 listener.cpp 文件，并编写 listener.cpp 程序。

```
roscd beginner_tutorials
cd src
touch listener.cpp
```

接下来需要编译两份代码。打开 CMakeLists.txt 文件。

```
rosed beginner_tutorials CMakeLists.txt
```

在 CMakeLists.txt 文件中添加以下语句，以编译 talker.cpp 和 listener.cpp。

```
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker beginner_tutorials_generate_messages_cpp)
add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```

语句可添加在任意位置，可以添加在 CMakeLists.txt 文件底部，也可加在 CMakeLists.txt 文件中 build 部分。在工作区根目录使用 catkin_make 编译。

```
cd ~/catkin_ws
catkin_make
```

新建两个终端并分别运行 talker 和 listener 即可。

1.2.12. ROS Server 与 Client

在 beginner_tutorials/src 目录下新建 add_two_ints_server.cpp 文件，编写程序使节点提供求两数之和的服务。

```
roscd beginner_yutorials
```

```
cd src
touch add_two_ints_server.cpp
```

在 beginner_tutorials/src 目录下新建 add_two_ints_client.cpp 文件， 编写程序使节点使用别的节点提供的求和服务。

```
roscd beginner_tutorials
cd src
touch add_two_ints_client.cpp
```

打开并修改 CMakeLists.txt 文件。

```
rosed beginner_tutorials CMakeLists.txt
```

在 CMakeLists.txt 文件中添加以下语句，以编译 add_two_ints_server.cpp 和 add_two_ints_client.cpp。

```
add_executable(add_two_ints_server src/add_two_ints_server.cpp)
target_link_libraries(add_two_ints_server ${catkin_LIBRARIES})
add_dependencies(add_two_ints_server beginner_tutorials_gencpp)
add_executable(add_two_ints_client src/add_two_ints_client.cpp)
target_link_libraries(add_two_ints_client ${catkin_LIBRARIES})
add_dependencies(add_two_ints_client beginner_tutorials_gencpp)
```

语句可添加在任意位置，可以添加在 CMakeLists.txt 文件底部，也可加在 CMakeLists.txt 文件中 build 部分。在工作区根目录使用 catkin_make 编译。

```
cd ~/catkin_ws
catkin_make
```

新建终端并运行 add_two_ints_server 服务即可。

1.2.13. 设计任务

设计控制乌龟运动的 ROS 节点，具体要求如下： （1）必须有发布 topic （2）必须有订阅 topic （3）必须调用 service （4）实现 python 和 C++两种版本 （4）自由发挥创造，设计节点的功能

例子： （1）向小乌龟发布命令。小乌龟先走一个矩形，再走一个圆形 （2）订阅小乌龟的位置信息。当小乌龟走完矩形并开始走圆形时，随机改变背景的颜色

具体实现见附件。

2. 任务二：ROS 仿真实验

2.1. Build robot

Build a mobile robot with a laser sensor.

2.1.1. Create ROS packages

```
cd ~/catkin_ws/src
mkdir mmbot_demo
catkin_create_pkg mmbot_description
catkin_create_pkg mmbot_gazebo
catkin_create_pkg mmbot_control rospy roscpp
```

2.1.2. Create key files

核心文件包括 mmbot.xacro, wheel.xacro, mmbot_gazebo.launch，详见附件。

2.2. Write robot controllers

Directory: mmbot_control

2.2.1. Create key files

核心文件包括 key, where, avoid(自行编写), goto(自行编写), hybrid(自行编写)，详见附件。

附录 A：ROS 设计任务源代码

main.cpp:

```
// c++ basic header
#include <iostream>
// ros basic header
#include <ros/ros.h>
#include <std_srvs/Empty.h>
#include <geometry_msgs/Twist.h>
// turtlesim node header
#include <turtlesim/Pose.h>
#include <turtlesim/Color.h>

geometry_msgs::Twist msg;

// print message information to stdIO
void printInfo() {
    ROS_INFO_STREAM(std::setprecision(2) << std::fixed << "msg.linear.x = " << msg.linear.x <<
" , msg.angular.z = " << msg.angular.z);
}

// let turtle go straight
void goForward(ros::Publisher &move_publisher) {
    for (int i = 1; i <= 50000; i++) {
        msg.linear.x = 1;
        msg.angular.z = 0;
        move_publisher.publish(msg);
        printInfo();
    }
}
```

```
// let turtle draw a ractangle
void drawRectangle(ros::Publisher &move_publisher) {
    goForward(move_publisher);
    // the limit of i is not equal
    for (int i = 1; i <= 42325; i++) {
        msg.linear.x = 0;
        msg.angular.z = 1;
        move_publisher.publish(msg);
        printInfo();
    }
    goForward(move_publisher);
    for (int i = 1; i <= 42310; i++) {
        msg.linear.x = 0;
        msg.angular.z = 1;
        move_publisher.publish(msg);
        printInfo();
    }
    goForward(move_publisher);
    for (int i = 1; i <= 42200; i++) {
        msg.linear.x = 0;
        msg.angular.z = 1;
        move_publisher.publish(msg);
        printInfo();
    }
    goForward(move_publisher);
    for (int i = 1; i <= 42190; i++) {
        msg.linear.x = 0;
        msg.angular.z = 1;
        move_publisher.publish(msg);
```

```
        printInfo();
    }
    for (int i = 1; i <= 20000; i++) {
        msg.linear.x = 1;
        msg.angular.z = 0;
        move_publisher.publish(msg);
        printInfo();
    }
}

// change background color of the program
void changeBackgroundColor(ros::NodeHandle &nodeHandle) {
    // rand a color
    int Red = 255 * double(rand()) / double(RAND_MAX);
    int Green = 255 * double(rand()) / double(RAND_MAX);
    int Bule = 255 * double(rand()) / double(RAND_MAX);
    // set param of background
    ros::param::set("background_r", Red);
    ros::param::set("background_g", Green);
    ros::param::set("background_b", Bule);
    // make a service client
    ros::ServiceClient clearClient = nodeHandle.serviceClient<std_srvs::Empty>("/clear");
    std_srvs::Empty srv;
    clearClient.call(srv);
}

// let turtle draw a circle and change background color within some time
void drawCircleAndChangeColor(ros::NodeHandle &nodeHandle, ros::Publisher &move_publisher) {
    for (int i = 1; i <= 150000; i++) {
        // judge change background color
```

```
        if (i % 30000 == 0) changeBackgroundColor(nodeHandle);
        msg.linear.x = 1;
        msg.angular.z = 1;
        move_publisher.publish(msg);
        printInfo();
    }
}

// let turtle draw a circle
void drawCircle(ros::Publisher &move_publisher) {
    for (int i = 1; i <= 150000; i++) {
        msg.linear.x = 1;
        msg.angular.z = 1;
        move_publisher.publish(msg);
        printInfo();
    }
}

// main
int main(int argc, char **argv) {
    // ros init
    ros::init(argc, argv, "main");
    ros::NodeHandle nodeHandle;
    srand(time(0));

    // publish signal to topic
    ros::Publisher                    move_publisher                    =
nodeHandle.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1000);

    // draw ractangle
```

```
    drawRectangle(move_publisher);


    // draw a circle
    drawCircle(move_publisher);


    // draw a circle with background color changed
    while (ros::ok()) {
        drawCircleAndChangeColor(nodeHandle, move_publisher);
        rate.sleep();
    }
    return 0;
}
```

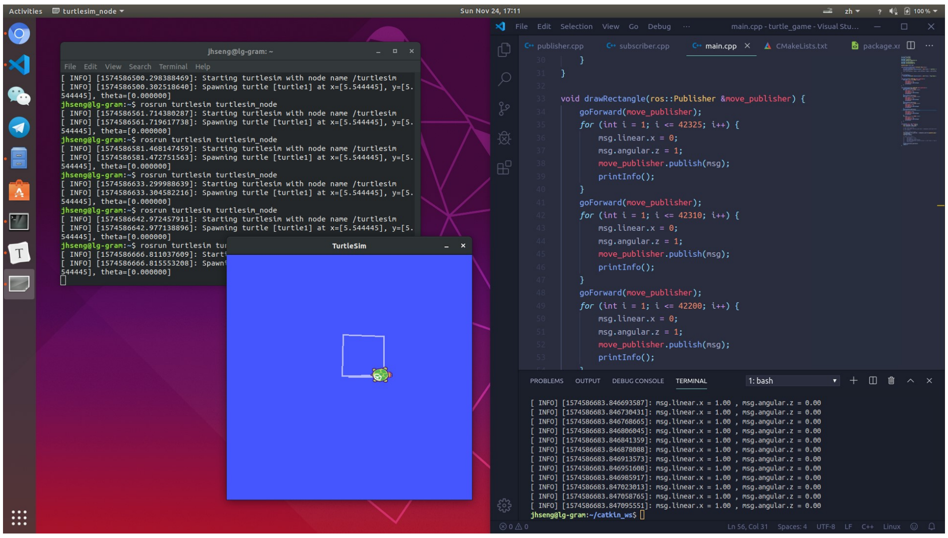
CMakeList.txt

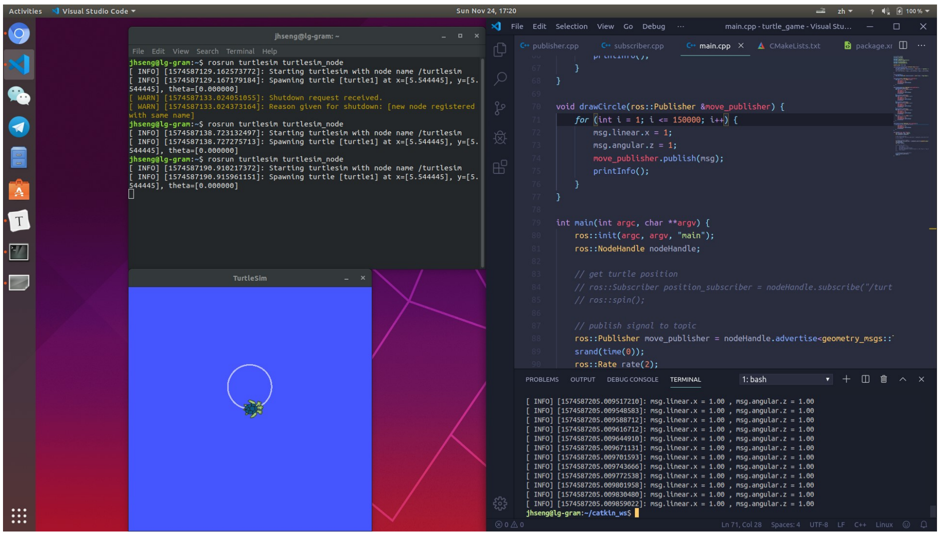
```
cmake_minimum_required(VERSION 2.8.3)
project(turtle_game)


find_package(catkin REQUIRED COMPONENTS
    roscpp
    turtlesim
)
add_executable(main main.cpp)


${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(main ${catkin_LIBRARIES})
```

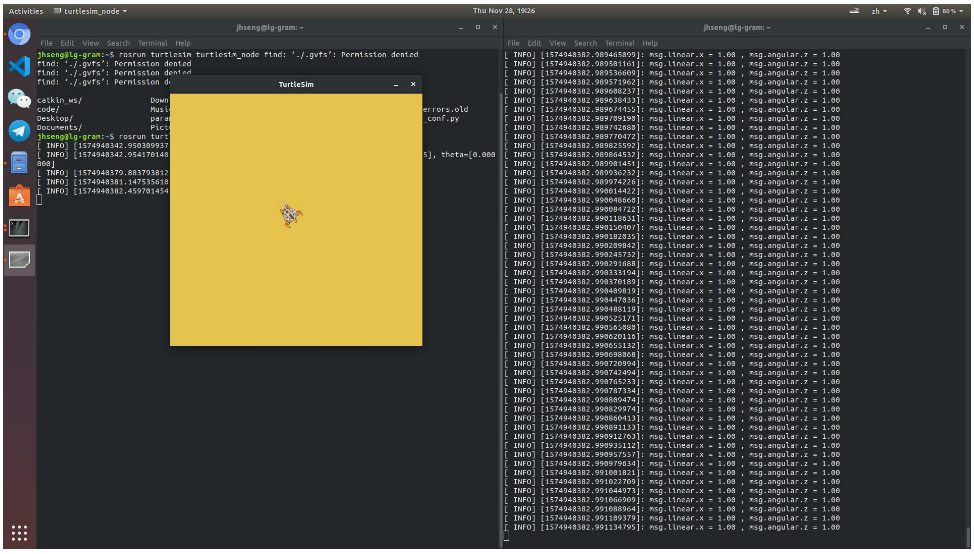
package.xml

```xml
<?xml version="1.0"?>
<package format="2">
  <name>turtle_game</name>
  <version>0.1.0</version>
  <description>a game for turtlesim node</description>

  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="zxc98567@gmail.com">jhseng</maintainer>
  <license>MIT</license>

  <author email="zxc98567@gmail.com">jhseng</author>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <exec_depend>roscpp</exec_depend>
  <build_depend>geometry_msgs</build_depend>
  <exec_depend>geometry_msgs</exec_depend>
  <build_depend>turtlesim</build_depend>
  <exec_depend>turtlesim</exec_depend>
  <build_depend>std_msgs</build_depend>
  <exec_depend>std_msgs</exec_depend>
  <export>

  </export>
</package>
```

附录 B：ROS 设计任务运行截图







附录 C：ROS 仿真实验源代码

mmbot.xacro

```
<?xml version="1.0"?>
<robot name="mmrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">
<xacro:include filename="$(find mmbot_description)/urdf/wheel.xacro" />
<!-- Defining the colors used in this robot -->
   <material name="Black">
      <color rgba="0.0 0.0 0.0 1.0"/>
   </material>
   <material name="Red">
      <color rgba="0.8 0.0 0.0 1.0"/>
   </material>
   <material name="White">
      <color rgba="1.0 1.0 1.0 1.0"/>
   </material>
   <material name="Blue">
      <color rgba="0.0 0.0 0.8 1.0"/>
   </material>
<!-- PROPERTY LIST -->
   <!--All units in m-kg-s-radians unit system -->
   <property name="M_PI" value="3.1415926535897931" />
   <property name="M_PI_2" value="1.570796327" />
   <property name="DEG_TO_RAD" value="0.017453293" />
   <!-- Main body radius and height -->
   <!-- Main Body Cylinder base      -->
   <property name="base_height" value="0.02" />
   <property name="base_radius" value="0.15" />
   <property name="base_mass" value="5" /> <!-- in kg-->
   <!-- caster wheel radius and height -->
   <!-- caster wheel mass -->
```

```
  <property name="caster_f_height" value="0.04" />
  <property name="caster_f_radius" value="0.025" />
  <property name="caster_f_mass" value="0.5" /> <!-- in kg-->
  <!-- caster wheel radius and height -->
  <!-- caster wheel mass -->
  <property name="caster_b_height" value="0.04" />
  <property name="caster_b_radius" value="0.025" />
  <property name="caster_b_mass" value="0.5" /> <!-- in kg-->
  <!-- Wheels -->
  <property name="wheel_mass" value="2.5" /> <!-- in kg-->
  <property name="base_x_origin_to_wheel_origin" value="0.25" />
  <property name="base_y_origin_to_wheel_origin" value="0.3" />
  <property name="base_z_origin_to_wheel_origin" value="0.0" />
  <!-- Hokuyo Laser scanner -->
  <property name="hokuyo_size" value="0.05" />
  <!-- Macro for calculating inertia of cylinder -->
  <macro name="cylinder_inertia" params="m r h">
    <inertia  ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
              iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
              izz="${m*r*r/2}" />
  </macro>
<!-- BASE-FOOTPRINT -->
<!-- base_footprint is a fictitious link(frame) that is on the ground right below base_link origin -->
  <link name="base_footprint">
    <inertial>
      <mass value="0.0001" />
      <origin xyz="0 0 0" />
      <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
          iyy="0.0001" iyz="0.0"
```

```
            izz="0.0001" />
        </inertial>
        <visual>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <box size="0.001 0.001 0.001" />
            </geometry>
        </visual>
    </link>
    <gazebo reference="base_footprint">
        <turnGravityOff>false</turnGravityOff>
    </gazebo>
    <joint name="base_footprint_joint" type="fixed">
        <origin xyz="0 0 ${wheel_radius - base_z_origin_to_wheel_origin}" rpy="0 0 0" />
        <parent link="base_footprint"/>
        <child link="base_link" />
    </joint>
<!-- BASE-LINK -->
<!--Actual body/chassis of the robot-->
    <link name="base_link">
        <inertial>
            <mass value="${base_mass}" />
            <origin xyz="0 0 0" />
            <!--The 3x3 rotational inertia matrix. -->
            <cylinder_inertia  m="${base_mass}" r="${base_radius}" h="${base_height}" />
        </inertial>
        <visual>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <cylinder length="${base_height}" radius="${base_radius}" />
```

```
        </geometry>
        <material name="White" />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0 " />
        <geometry>
            <cylinder length="${base_height}" radius="${base_radius}" />
        </geometry>
      </collision>
    </link>
    <gazebo reference="base_link">
      <material>Gazebo/White</material>
      <turnGravityOff>false</turnGravityOff>
    </gazebo>
<!--Caster front    -->
    <link name="caster_front_link">
      <visual>
        <origin xyz="0 0.02 0" rpy="${M_PI/2} 0 0" />
        <geometry>
          <sphere radius="${caster_f_radius}" />
        </geometry>
        <material name="Black" />
      </visual>
      <collision>
          <geometry>
            <sphere radius="${caster_f_radius}" />
          </geometry>
          <origin xyz="0 0.02 0" rpy="${M_PI/2} 0 0" />
      </collision>
      <inertial>
```

```
            <mass value="${caster_f_mass}" />
            <origin xyz="0 0 0" />
            <inertia ixx="0.001" ixy="0.0" ixz="0.0"
                     iyy="0.001" iyz="0.0"
                     izz="0.001" />
        </inertial>
    </link>
    <joint name="caster_front_joint" type="fixed">
          <parent link="base_link"/>
          <child link="caster_front_link"/>
          <origin xyz="0.115 0.0 0.007" rpy="${-M_PI/2} 0 0"/>
    </joint>
    <gazebo reference="caster_front_link">
        <turnGravityOff>false</turnGravityOff>
    </gazebo>
<!--Caster back    -->
    <link name="caster_back_link">
        <visual>
            <origin xyz="0.02 0.02 0 " rpy="${M_PI/2} 0 0" />
            <geometry>
                <sphere radius="${caster_b_radius}" />
            </geometry>
            <material name="Black" />
        </visual>
        <collision>
            <geometry>
                <sphere radius="${caster_b_radius}" />
            </geometry>
            <origin xyz="0 0.02 0 " rpy="${M_PI/2} 0 0" />
        </collision>
```

```
    <inertial>
        <mass value="${caster_b_mass}" />
        <origin xyz="0 0 0" />
        <inertia ixx="0.001" ixy="0.0" ixz="0.0"
                 iyy="0.001" iyz="0.0"
                 izz="0.001" />
    </inertial>
  </link>
  <joint name="caster_back_joint" type="fixed">
        <parent link="base_link"/>
        <child link="caster_back_link"/>
        <origin xyz="-0.135 0.0 0.009" rpy="${-M_PI/2} 0 0"/>
  </joint>
  <gazebo reference="caster_back_link">
    <turnGravityOff>false</turnGravityOff>
  </gazebo>
<!-- Wheel Definitions -->
  <wheel fb="front" lr="right" parent="base_link" translateX="0" translateY="0.5" flipY="1"/>
  <wheel fb="front" lr="left" parent="base_link" translateX="0" translateY="-0.5" flipY="1"/>
<!-- SENSORS -->
<!-- hokuyo -->
  <link name="hokuyo_link">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
          <box size="${hokuyo_size} ${hokuyo_size} ${hokuyo_size}"/>
        </geometry>
        <material name="Blue" />
```

```
    </visual>
  </link>
  <joint name="hokuyo_joint" type="fixed">
    <origin xyz="${base_radius - hokuyo_size/2} 0 ${base_height+hokuyo_size/4}" rpy="0 0 0" />
    <parent link="base_link"/>
    <child link="hokuyo_link" />
  </joint>
  <gazebo reference="hokuyo_link">
    <material>Gazebo/Blue</material>
    <turnGravityOff>false</turnGravityOff>
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>${hokuyo_size/2} 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>5</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>10</samples>
            <resolution>1</resolution>
            <min_angle>-1.570796</min_angle>
            <max_angle>1.570796</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.10</min>
          <max>10.0</max>
          <resolution>0.001</resolution>
        </range>
      </ray>
```

```
        <plugin                                     name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
            <topicName>/scan</topicName>
            <frameName>hokuyo_link</frameName>
        </plugin>
      </sensor>
    </gazebo>
<!-- Differential drive controller   -->
    <gazebo>
      <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
        <rosDebugLevel>Debug</rosDebugLevel>
        <publishWheelTF>false</publishWheelTF>
        <robotNamespace>/</robotNamespace>
        <publishTf>1</publishTf>
        <publishWheelJointState>false</publishWheelJointState>
        <alwaysOn>true</alwaysOn>
        <updateRate>10.0</updateRate>
        <leftJoint>front_left_wheel_joint</leftJoint>
        <rightJoint>front_right_wheel_joint</rightJoint>
        <wheelSeparation>${2*base_radius}</wheelSeparation>
        <wheelDiameter>${2*wheel_radius}</wheelDiameter>
        <broadcastTF>1</broadcastTF>
        <wheelTorque>30</wheelTorque>
        <wheelAcceleration>1.8</wheelAcceleration>
        <commandTopic>cmd_vel</commandTopic>
        <odometryFrame>odom</odometryFrame>
        <odometryTopic>odom</odometryTopic>
        <robotBaseFrame>base_footprint</robotBaseFrame>
        <legacyMode>true</legacyMode>
        <odometrySource>world</odometrySource>
```

```
    </plugin>
  </gazebo>
</robot>
```

wheel.xacro

```
<?xml version="1.0"?>
<robot name="wheel" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- Wheels -->
  <property name="wheel_radius" value="0.04" />
  <property name="wheel_height" value="0.02" />
  <property name="wheel_mass" value="2.5" /> <!-- in kg-->
  <property name="base_x_origin_to_wheel_origin" value="0.25" />
  <property name="base_y_origin_to_wheel_origin" value="0.3" />
  <property name="base_z_origin_to_wheel_origin" value="0.0" />
  <macro name="cylinder_inertia" params="m r h">
    <inertia  ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
              iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
              izz="${m*r*r/2}" />
  </macro>
  <xacro:macro name="wheel" params="fb lr parent translateX translateY flipY">
<!--fb : front, back ; lr: left, right -->
    <link name="${fb}_${lr}_wheel">
      <visual>
        <origin xyz="0 0 0" rpy="${flipY*M_PI/2} 0  0 " />
        <geometry>
          <cylinder length="${wheel_height}" radius="${wheel_radius}" />
        </geometry>
        <material name="DarkGray" />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="${flipY*M_PI/2} 0 0 " />
```

```
            <geometry>
                <cylinder length="${wheel_height}" radius="${wheel_radius}" />
            </geometry>
        </collision>
        <inertial>
            <mass value="${wheel_mass}" />
            <origin xyz="0 0 0" />
            <cylinder_inertia m="${wheel_mass}" r="${wheel_radius}"
h="${wheel_height}" />
        </inertial>
    </link>
    <gazebo reference="${fb}_${lr}_wheel">
        <mu1 value="1.0"/>
        <mu2 value="1.0"/>
        <kp  value="10000000.0" />
        <kd  value="1.0" />
        <fdir1 value="1 0 0"/>
        <material>Gazebo/Grey</material>
        <turnGravityOff>false</turnGravityOff>
    </gazebo>
    <joint name="${fb}_${lr}_wheel_joint" type="continuous">
        <parent link="${parent}"/>
        <child link="${fb}_${lr}_wheel"/>
        <origin xyz="${translateX * base_x_origin_to_wheel_origin} ${translateY *
base_y_origin_to_wheel_origin} ${base_z_origin_to_wheel_origin}" rpy="0 0 0" />
        <axis xyz="0 1 0" rpy="0  0" />
        <limit effort="100" velocity="100"/>
        <joint_properties damping="0.0" friction="0.0"/>
    </joint>
    <!-- Transmission is important to link the joints and the controller -->
```

```
    <transmission name="${fb}_${lr}_wheel_joint_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="${fb}_${lr}_wheel_joint" />
      <actuator name="${fb}_${lr}_wheel_joint_motor">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>
  </xacro:macro>
</robot>
```

avoid.py

```
#!/usr/bin/env python


import rospy
import math
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan



class Mmbot:
    def __init__(self):
        rospy.init_node('run_avoid', anonymous=True)
        self.velocity_publisher = rospy.Publisher(
            'cmd_vel', Twist, queue_size=5)
        self.pose_subscriber = rospy.Subscriber(
            '/scan', LaserScan, self.update_ranges)
        self.laser_scan = LaserScan()
        self.rate = rospy.Rate(100)

    def update_ranges(self, data):
```

```
            print("This in the callback function: ")
            self.laser_scan = data
            print(data.ranges[0])

    def run(self):
            __rate = rospy.Rate(1)
            __rate.sleep()
            currentMessage = Twist()
            currentMessage.linear.x = 0.2
            while not rospy.is_shutdown():
                    r = self.laser_scan.ranges
                    if (r[2] < 1 or r[3] < 1):
                            currentMessage.angular.z = 2
                    elif (r[7] < 1 or r[6] < 1):
                            currentMessage.angular.z = -2
                    else:
                            currentMessage.angular.z = 0
                    if (r[4] < 1):
                            currentMessage.angular.z = 2
                    elif (r[5] < 1):
                            currentMessage.angular.z = -2
                    else:
                            currentMessage.angular.z = 0
                    self.velocity_publisher.publish(currentMessage)
                    self.rate.sleep()


if __name__ == '__main__':
    try:
            mmbot = Mmbot()
```

```
        mmbot.run()
    except rospy.ROSInternalException:
        pass
```

goto.py

```
#!/usr/bin/env python
import rospy
import tf
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from math import pow, atan2, sqrt
from tf.transformations import euler_from_quaternion
class Mmbot:
    def __init__(self):
        rospy.init_node('goto', anonymous=True)
        self.velocity_publisher = rospy.Publisher(
            'cmd_vel', Twist, queue_size=5)
        self.pose_subscriber = rospy.Subscriber(
            '/odom', Odometry, self.update_pose)
        self.odometry = Odometry()
        self.rate = rospy.Rate(1)
    def update_pose(self, data):
        self.odometry = data
        self.odometry.pose.pose.position.x = round(
            self.odometry.pose.pose.position.x, 4)
        self.odometry.pose.pose.position.y = round(
            self.odometry.pose.pose.position.y, 4)
    def euclidean_distance(self, goal_pose):
        return sqrt(pow((goal_pose.pose.pose.position.x -
self.odometry.pose.pose.position.x), 2) +
                    pow((goal_pose. pose.pose.position.y -
```

```
self.odometry.pose.pose.position.y), 2))
    def linear_vel(self, goal_pose, constant=0.2):
        return constant * self.euclidean_distance(goal_pose)
    def steering_angle(self, goal_pose):
        return            atan2(goal_pose.pose.pose.position.y          -
self.odometry.pose.pose.position.y,         goal_pose.pose.pose.position.x        -
self.odometry.pose.pose.position.x)
    def angular_vel(self, goal_pose, constant=1):
        p = euler_from_quaternion((self.odometry.pose.pose.orientation.x,
                                   self.odometry.pose.pose.orientation.y,
                                   self.odometry.pose.pose.orientation.z,
                                   self.odometry.pose.pose.orientation.w))
        print constant * (self.steering_angle(goal_pose) - p[2])
        return constant * (self.steering_angle(goal_pose) - p[2])
    def move2goal(self):
        self.rate.sleep()
        goal_pose = Odometry()
        goal_pose.pose.pose.position.x = input("Set your x goal: ")
        goal_pose.pose.pose.position.y = input("Set your y goal: ")
        distance_tolerance = input("Set your tolerance: ")
        vel_msg = Twist()
        while self.euclidean_distance(goal_pose) >= distance_tolerance:
            vel_msg.linear.x = self.linear_vel(goal_pose)
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = -self.angular_vel(goal_pose)
            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()
```

```
            vel_msg.linear.x = 0
            vel_msg.angular.z = 0
            self.velocity_publisher.publish(vel_msg)
            rospy.spin()
if __name__ == '__main__':
    try:
        mmbot = Mmbot()
        mmbot.move2goal()
    except rospy.ROSInterruptException:
        pass
```

hybrid.py

```
#!/usr/bin/env python
import rospy
import tf
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from math import pow, atan2, sqrt
from tf.transformations import euler_from_quaternion
from sensor_msgs.msg import LaserScan
class Mmbot:
    def __init__(self):
        rospy.init_node('goto', anonymous=True)
        self.velocity_publisher = rospy.Publisher(
            'cmd_vel', Twist, queue_size=5)
        self.pose_subscriber = rospy.Subscriber(
            '/odom', Odometry, self.update_pose)
        self.laser_subscriber = rospy.Subscriber(
            '/scan', LaserScan, self.update_ranges)
        self.odometry = Odometry()
```

```
        self.laser_scan = LaserScan()
        self.rate = rospy.Rate(1)
    def update_ranges(self, data):
        self.laser_scan = data
    def update_pose(self, data):
        self.odometry = data
        self.odometry.pose.pose.position.x = round(
            self.odometry.pose.pose.position.x, 4)
        self.odometry.pose.pose.position.y = round(
            self.odometry.pose.pose.position.y, 4)
    def euclidean_distance(self, goal_pose):
        return sqrt(pow((goal_pose.pose.pose.position.x -
self.odometry.pose.pose.position.x), 2) +
                    pow((goal_pose. pose.pose.position.y -
self.odometry.pose.pose.position.y), 2))
    def linear_vel(self, goal_pose, constant=0.2):
        return constant * self.euclidean_distance(goal_pose)
    def steering_angle(self, goal_pose):
        return atan2(goal_pose.pose.pose.position.y -
self.odometry.pose.pose.position.y, goal_pose.pose.pose.position.x -
self.odometry.pose.pose.position.x)
    def angular_vel(self, goal_pose, constant=1):
        p = euler_from_quaternion((self.odometry.pose.pose.orientation.x,
                                   self.odometry.pose.pose.orientation.y,
                                   self.odometry.pose.pose.orientation.z,
                                   self.odometry.pose.pose.orientation.w))
        print constant * (self.steering_angle(goal_pose) - p[2])
        return constant * (self.steering_angle(goal_pose) - p[2])
    def move2goal(self):
        self.rate.sleep()
```

```
        goal_pose = Odometry()
        goal_pose.pose.pose.position.x = input("Set your x goal: ")
        goal_pose.pose.pose.position.y = input("Set your y goal: ")
        distance_tolerance = input("Set your tolerance: ")
        currentMessage = Twist()
        while self.euclidean_distance(goal_pose) >= distance_tolerance:
            __range = self.laser_scan.ranges
            currentMessage.linear.x = self.linear_vel(goal_pose)
            currentMessage.linear.y = 0
            currentMessage.linear.z = 0
            currentMessage.angular.x = 0
            currentMessage.angular.y = 0
            currentMessage.angular.z = -self.angular_vel(goal_pose)
            if (__range[4] < 1):
                currentMessage.angular.z = 2
            elif(__range[5] < 1):
                currentMessage.angular.z = -2
            self.velocity_publisher.publish(currentMessage)
            self.rate.sleep()
        currentMessage.linear.x = 0
        currentMessage.angular.z = 0
        self.velocity_publisher.publish(currentMessage)
        rospy.spin()
if __name__ == '__main__':
    try:
        mmbot = Mmbot()
        mmbot.move2goal()
    except rospy.ROSInterruptException:
        pass
```

key.py

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
import sys
import select
import termios
import tty
msg = """
Control Your Turtlebot!
---------------------------
Moving around:
    u    i    o
    j    k    l
    m    ,    .
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
CTRL-C to quit
"""
moveBindings = {
        'i': (1, 0),
        'o': (1, -1),
        'j': (0, 1),
        'l': (0, -1),
        'u': (1, 1),
        ',': (-1, 0),
        '.': (-1, 1),
```

```
    'm': (-1, -1),
}
speedBindings = {
    'q': (1.1, 1.1),
    'z': (.9, .9),
    'w': (1.1, 1),
    'x': (.9, 1),
    'e': (1, 1.1),
    'c': (1, .9),
}
def getKey():
    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key
speed = .2
turn = 1
def vels(speed, turn):
    return "currently:\tspeed %s\tturn %s " % (speed, turn)
if __name__ == "__main__":
    settings = termios.tcgetattr(sys.stdin)
    rospy.init_node('turtlebot_teleop')
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=5)
    x = 0
    th = 0
    status = 0
```

```
count = 0
acc = 0.1
target_speed = 0
target_turn = 0
control_speed = 0
control_turn = 0
try:
    print msg
    print vels(speed, turn)
    while(1):
        key = getKey()
        if key in moveBindings.keys():
            x = moveBindings[key][0]
            th = moveBindings[key][1]
            count = 0
        elif key in speedBindings.keys():
            speed = speed * speedBindings[key][0]
            turn = turn * speedBindings[key][1]
            count = 0
            print vels(speed, turn)
            if (status == 14):
                print msg
            status = (status + 1) % 15
        elif key == ' ' or key == 'k':
            x = 0
            th = 0
            control_speed = 0
            control_turn = 0
        else:
            count = count + 1
```

```
        if count > 4:
            x = 0
            th = 0
        if (key == '\x03'):
            break
    target_speed = speed * x
    target_turn = turn * th
    if target_speed > control_speed:
        control_speed = min(target_speed, control_speed + 0.02)
    elif target_speed < control_speed:
        control_speed = max(target_speed, control_speed - 0.02)
    else:
        control_speed = target_speed
    if target_turn > control_turn:
        control_turn = min(target_turn, control_turn + 0.1)
    elif target_turn < control_turn:
        control_turn = max(target_turn, control_turn - 0.1)
    else:
        control_turn = target_turn
    twist = Twist()
    twist.linear.x = control_speed
    twist.linear.y = 0
    twist.linear.z = 0
    twist.angular.x = 0
    twist.angular.y = 0
    twist.angular.z = control_turn
    pub.publish(twist)
    #print("loop: {0}".format(count))
    #print("target: vx: {0}, wz: {1}".format(target_speed, target_turn))
    #print("publihsed: vx: {0}, wz: {1}".format(twist.linear.x,
```

```
twist.angular.z))
    except:
        print e
    finally:
        twist = Twist()
        twist.linear.x = 0
        twist.linear.y = 0
        twist.linear.z = 0
        twist.angular.x = 0
        twist.angular.y = 0
        twist.angular.z = 0
        pub.publish(twist)
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

test.py

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
def solve():
    rospy.init_node('test')
    currentMessage = Twist()
    messagePublisher = rospy.Publisher("/cmd_vel", Twist, queue_size=1000)
    currentMessage.linear.x = 0.2
    rate = rospy.Rate(100)
    while not rospy.is_shutdown():
        messagePublisher.publish(currentMessage)
        rospy.loginfo("published")
        rate.sleep()
if __name__ == '__main__':
    try:
        solve()
```

```
    except rospy.ROSInterruptException:
        pass
```

where.py

```
#!/usr/bin/env python
# Copyright(c) 2017, SCUT RIS
# author : Jinhui Zhu
import rospy
from nav_msgs.msg import Odometry
def odom_callback(data):
    rx = data.pose.pose.position.x
    ry = data.pose.pose.position.y
    rospy.loginfo(rospy.get_caller_id()+": %.2f %.2f", rx, ry)
if __name__ == "__main__":
    rospy.init_node("where")
    sub = rospy.Subscriber('/odom', Odometry, odom_callback)
    rospy.spin()
```

提交文件说明

```
|-report.doc
|-report.pdf
|-src
    |-lab1
        | - main.cpp
        | - CMakeLists.txt
        | - package.xml
    |-lab2
            | - mmbot_control
                | - CMakeLists.txt
                | - package.xml
                | - scripts
                    | - avoid.py
                    | - goto.py
                    | - hybrid.py
                    | - key.py
                    | - test.py
                    | - where.py
            | - mmbot_description
                | - CMakeLists.txt
                | - package.xml
                | - launch
                    | - mmbot_gazebo.launch
                | - urdf
                    | - mmbot.xacro
                    | - wheel.xacro
            | - mmbot_gazebo
                | - CMakeLists.txt
```

| - package.xml

| - launch

| - mmbot_gazebo.launch

| - CMakeLists.txt

| - package.xml

|-image

|- lab1

| -小乌龟画矩形.png

| -小乌龟画圆形.png

| -小乌龟画圆形并变色.png