实验指导书

实验4 ROS简单应用：激光SLAM与导航

1、实验目的

（1）了解机器人和 SLAM 之间关系
（2）了解激光 SLAM
（3）了解 gmapping 和导航

2、实验设备

硬件环境：PC
软件环境：Ubuntu18.04、ROS Melodic

3、实验内容

（1）SLAM 技术由来

近些年来，随着世界各国经济和科技的迅猛发展，目前已经形成了全球信息化和自动化的生产和生活格局。伴随其发展的机器人、无人机和自动驾驶等人工智能技术受到了科技界和企业界的越发青睐。智能机器人技术作为现代人工智能新兴产业之一，不仅改变了企业的生产方式，而且还提高了普通民众的生活质量水平。

机器人实现智能化的重要基础是能够自主移动，其不仅需要机器人获取位于环境中的方位信息，而且还需要对环境进行感知和建模。如家庭服务扫地机器人、智能无人机、无人驾驶汽车等需要对周围环境进行建图，并定位自身在地图中的方位，同时执行路径规划和碰撞检测等操作。因此，机器人要实现自主移动，需解决定位、建图和路径规划等关键问题，其中定位和建图问题主要由同时定位与地图构建（Simultaneous Localization and Mapping，SLAM）技术解决，如图 1 所示。SLAM 技术最早是在机器人研究领域被专家们提出，近年来随着人工智能技术的快速发展，该项技术受到了研究者们的广泛关注。

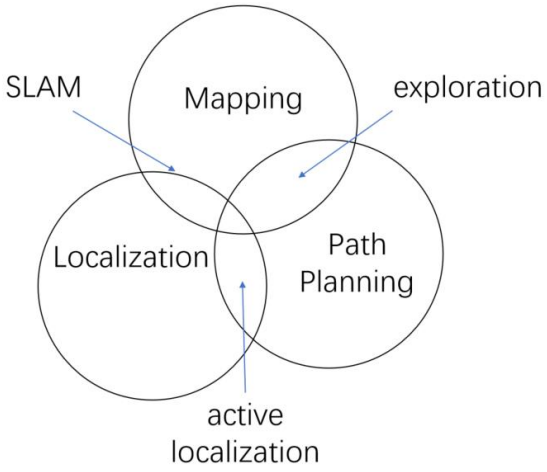

图 1 机器人自主移动相关技术

（2）SLAM 分类

一般地，SLAM 可根据传感器的种类分为激光 SLAM 和视觉 SLAM 两种典型方法。基于激光的 SLAM 理论研究时间较早，技术成熟，因此它的应用产品丰富，如利用多维激光进行 3D 点云建图等。视觉 SLAM 搭载的传感器（单目、双目、RGB-D 等等）可以从环境中采集并提取海量丰富的纹理信息，这些信息可以提供给计算机视觉相关任务处理（如目标检测和语义分割），进而使系统拥有较强的语义场景辨识能力；同时可以利用视觉信息跟踪环境中的动态物体，如行人、车辆等，对于在复杂动态环境中的应用至关重要。

（3）gmapping

- **介绍**

gmapping 通过粒子滤波将激光距离数据转化为珊格地图。优点：在长廊及低特征场景中建图效果好；构建小场景地图所需的计算量较小且精度较高。缺点：依赖里程计，无法适用无人机及地面小车不平坦区域；无回环；大的场景，粒子较多的情况下，特别消耗资源

- **安装依赖**

在开始该部分设计之前，我们先安装相关支持的功能包（**本部分在虚拟机中会出现较多问题，建议使用完整的 Ubuntu18 的系统**），包括建图、导航以及机器人仿真的功能包（克隆下载后仅保留 turtlebot3_gazebo 文件夹），相关命令如下：

```
sudo apt-get update
sudo apt-get install ros-melodic-turtlebot3-*
sudo apt-get install ros-melodic-navigation
sudo apt-get install ros-melodic-slam-gmapping
git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
sudo apt-get install ros-melodic-dwa-local-planner
```

- **建立文件夹**

新建 ROS 工作空间 test_ws：

```
mkdir –p test_ws/src
cd test_ws
catkin_make
cd src
catkin_create_pkg my_navi std_msgs roscpp rospy sensor_msgs message_generation
```

将下载的 turtlebot3_gazebo 文件夹复制到到 my_navi 文件夹下。在 my_navi 文件夹下建立四个文件夹 launch，map，params，scripts。

在使用仿真机器人之前先利用以下命令确定机器人类型，在 test_ws 目录下运行：

```
echo “export TURTLEBOT3_MODEL=burger”   >>~/.bashrc
```

- **添加建图文件内容**

在建图部分的设计我们采用开源的 Gmapping 算法，在 launch 文件中新建文件，建图文件名为 build_map.launch，算法主要参数如下，该文件主要是启动机器人以及加载 gmapping 节点。

```
<launch>
    <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
    <arg name="set_base_frame" default="base_footprint"/>
    <arg name="set_odom_frame" default="odom"/>
    <arg name="set_map_frame"   default="map"/>
    <!-- TurtleBot3 机器人启动-->
    <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
        <arg name="model" value="$(arg model)" />
    </include>
    <!--   ********** Gmapping   **********   -->
    <node name="gmapping" pkg="gmapping" type="slam_gmapping">
        <param name="base_frame" value="$(arg set_base_frame)"/>
        <param name="odom_frame" value="$(arg set_odom_frame)"/>
        <param name="map_frame"   value="$(arg set_map_frame)"/>
        <param name="scan" value="/scan"/>
        <param name="map_update_interval" value="3.0"/>
        <param name="xmin" value="-25"/>
        <param name="xmax" value="25"/>
        <param name="ymin" value="-25"/>
        <param name="ymax" value="25"/>
    </node>
</launch>
```

完成文件配置之后，回到 test _ws 目录下，先 catkin_make 编译然后刷新工作空间，最后在运行 launch 文件即可完成整体功能的启动，建图流程：

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
roslaunch my_navi build_map.launch
rosrun turtlebot3_teleop turtlebot3_teleop_key
rosrun map_server map_saver –f map2
```

运行完上述前两条命令之后会出现一个仿真的场景，然后 rviz 中会显示当前建图结果，如图 2 所示，然后运行第三条命令会出现键盘控制界面，可以利用“w”，“a”，“s”，“d”来控制机器人移动，这四个按键分别控制向前移动、向左自旋、停止和向右自旋，合理的控制机器人移动来建立环境地图，结果如图 3 所示，最后利用第四条命令保存建图结果，产生图片名为“map2.pgm”和名为“map2.yaml”的地图参数，最终建图结果如图 4 所示，地图参数则如下所示：

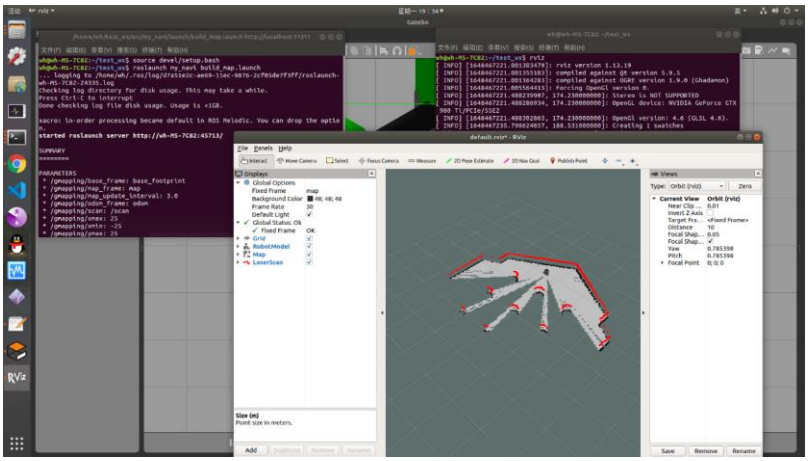

图 2 开始时建图结果

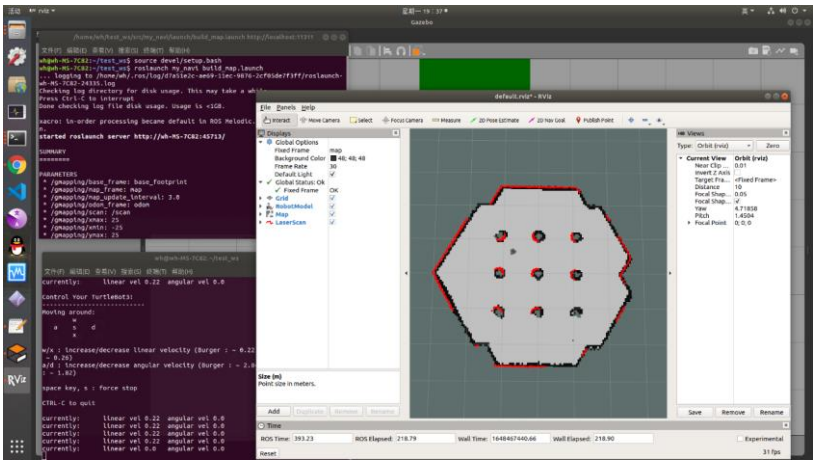

图 3 建图完成

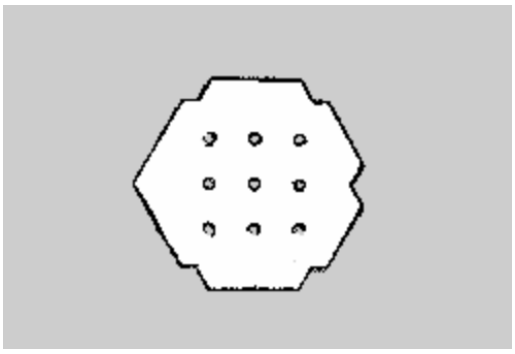

图 4 map2.pgm 建图结果

而 map2.yaml 中的参数如下：

```
image: map2.pgm
resolution: 0.050000
origin: [-25.000000, -25.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

- **添加导航功能**

接下来是导航功能，在 params 文件夹下新建一个机器人定位启动的文件 amcl.launch，参数如下：

```
<launch>
  <!-- Arguments -->
  <arg name="scan_topic"         default="scan"/>
  <arg name="initial_pose_x" default="-1.968"/>
  <arg name="initial_pose_y" default="-0.046"/>
  <arg name="initial_pose_a" default="0.015"/>

  <!-- AMCL -->
  <node pkg="amcl" type="amcl" name="amcl">

    <param name="min_particles"                 value="500"/>
    <param name="max_particles"                 value="3000"/>
    <param name="kld_err"                       value="0.02"/>
    <param name="update_min_d"                  value="0.20"/>
    <param name="update_min_a"                  value="0.20"/>
    <param name="resample_interval"             value="1"/>
    <param name="transform_tolerance"           value="0.5"/>
    <param name="recovery_alpha_slow"           value="0.00"/>
    <param name="recovery_alpha_fast"           value="0.00"/>
    <param name="initial_pose_x"                value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"                value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"                value="$(arg initial_pose_a)"/>
    <param name="gui_publish_rate"              value="50.0"/>

    <remap from="scan"                          to="$(arg scan_topic)"/>
    <param name="laser_max_range"               value="3.5"/>
    <param name="laser_max_beams"               value="180"/>
    <param name="laser_z_hit"                   value="0.5"/>
    <param name="laser_z_short"                 value="0.05"/>
    <param name="laser_z_max"                   value="0.05"/>
    <param name="laser_z_rand"                  value="0.5"/>
    <param name="laser_sigma_hit"               value="0.2"/>
    <param name="laser_lambda_short"            value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type"              value="likelihood_field"/>

    <param name="odom_model_type"               value="diff"/>
    <param name="odom_alpha1"                   value="0.1"/>
    <param name="odom_alpha2"                   value="0.1"/>
    <param name="odom_alpha3"                   value="0.1"/>
    <param name="odom_alpha4"                   value="0.1"/>
    <param name="odom_frame_id"                 value="odom"/>
    <param name="base_frame_id"                 value="base_footprint"/>

  </node>
</launch>
```

在 params 文件夹下新建一个文件，用于存放代价地图的通用参数，名为 costmap_common_params.yaml，参数如下：

```
obstacle_range: 3.0
raytrace_range: 3.5
footprint: [[-0.105, -0.105], [-0.105, 0.105], [0.041, 0.105], [0.041, -0.105]]
#robot_radius: 0.105
inflation_radius: 1.0
cost_scaling_factor: 3.0
map_type: costmap
observation_sources: scan
scan: {sensor_frame: base_scan, data_type: LaserScan, topic: scan, marking: true, clearing: true}
```

在 params 文件夹下新建一个存放驱动底座的控制器参数的文件，文件名为 dwa_local_planner_params.yaml ，相关参数配置如下：

```
DWAPlannerROS:
# Robot Configuration Parameters
   max_vel_x: 0.26
   min_vel_x: -0.26
   max_vel_y: 0.0
   min_vel_y: 0.0
# The velocity when robot is moving in a straight line
   max_vel_trans:  0.26
   min_vel_trans:  0.13
   max_vel_theta: 1.82
   min_vel_theta: 0.9
   acc_lim_x: 2.5
   acc_lim_y: 0.0
   acc_lim_theta: 3.2
# Goal Tolerance Parametes
   xy_goal_tolerance: 0.05
   yaw_goal_tolerance: 0.17
   latch_xy_goal_tolerance: false
# Forward Simulation Parameters
   sim_time: 2.0
   vx_samples: 20
   vy_samples: 0
   vth_samples: 40
   controller_frequency: 10.0
# Trajectory Scoring Parameters
   path_distance_bias: 32.0
   goal_distance_bias: 20.0
   occdist_scale: 0.02
   forward_point_distance: 0.325
   stop_time_buffer: 0.2
   scaling_speed: 0.25
   max_scaling_factor: 0.2
# Oscillation Prevention Parameters
   oscillation_reset_dist: 0.05
# Debugging
   publish_traj_pc : true
   publish_cost_grid_pc: true
```

在 params 文件夹下新建一个文件，存放全局代价地图参数，名为global_costmap_params.yaml，相关参数如下：

```
global_costmap:
    global_frame: map
    robot_base_frame: base_footprint
    update_frequency: 5.0
    publish_frequency: 5.0
    static_map: true
    transform_tolerance: 0.5
    plugins:
      - {name: static_layer,            type: "costmap_2d::StaticLayer"}
      - {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
      - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

在 params 文件夹下新建一个文件，存放局部代价地图参数，名为local_costmap_params.yaml，相关参数如下：

```
local_costmap:
    global_frame: map
    robot_base_frame: base_footprint
    update_frequency: 5.0
    publish_frequency: 2.0
    static_map: false
    rolling_window: true
    width: 3.0
    height: 3.0
    resolution: 0.05
    transform_tolerance: 0.5
    plugins:- {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
      - {name: inflation_layer,       type: "costmap_2d::InflationLayer"}
```

在 params 文件夹下新建一个文件，存放全局路径规划参数，名为global_planner_params.yaml，相关参数如下：

```
GlobalPlanner:          # Also see: http://wiki.ros.org/global_planner
  old_navfn_behavior: false
  use_quadratic: true
  use_dijkstra: true
  use_grid_path: false
  visualize_potential: false
  allow_unknown: true
  planner_window_x: 0.0                                  # default 0.0
  planner_window_y: 0.0                                  # default 0.0
  default_tolerance: 0.0
  publish_scale: 100
  planner_costmap_publish_frequency: 0.0          # default 0.0
  lethal_cost: 253                                       # default 253
  neutral_cost: 50 #50                                   # default 50
  cost_factor: 3.0 #3.0
  publish_potential: true
```

在 params 文件夹下新建一个文件，名为 move_base.launch，用来加载导航参数，相关内容如下：

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="cmd_vel_topic" default="/cmd_vel" />
  <arg name="odom_topic" default="odom" />
  <arg name="move_forward_only" default="false"/>

  <!-- move_base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    <rosparam file="$(find my_navi)/params/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find my_navi)/params/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find my_navi)/params/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find my_navi)/params/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find my_navi)/params/global_planner_params.yaml" command="load" />
    <rosparam file="$(find my_navi)/params/dwa_local_planner_params.yaml" command="load" />
    <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
    <remap from="odom" to="$(arg odom_topic)"/>
    <param name="DWAPlannerROS/min_vel_x" value="0.0" if="$(arg move_forward_only)" />
  </node>
</launch>
```

在 launch 文件夹下新建一个文件，用于启动自主导航整体配置文件，名称为 turtlebot_navigation.launch，具体文件内容如下：

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="open_rviz" default="true"/>
  <arg name="move_forward_only" default="false"/>
  <!-- Turtlebot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>
  <!-- Map server -->
  <arg name="map_file"            default="$(find my_navi)/map/map2.yaml"/>
  <node pkg="map_server" type="map_server" name="map_server" args="$(arg map_file)">
    <param name="frame_id" value="/map" />
  </node>
  <!-- AMCL -->
  <include file="$(find my_navi)/params/amcl.launch"/>
  <!-- move_base -->
  <include file="$(find my_navi)/params/move_base.launch">
    <arg name="model" value="$(arg model)" />
    <arg name="move_forward_only" value="$(arg move_forward_only)"/>
  </include>
  <node pkg="rviz" type="rviz" name="rviz" required="true"
    args="-d $(find turtlebot3_navigation)/rviz/turtlebot3_navigation.rviz" output="screen"/>
</launch>
```

最后相关文件结构如图所示：

```
wh@wh-MS-7C82:~/test_ws/src/my_navi$ tree -L 2
.
├── CMakeLists.txt
├── launch
│   ├── build_map.launch
│   └── turtlebot_navigation.launch
├── map
│   ├── map2.pgm
│   └── map2.yaml
├── package.xml
├── params
│   ├── amcl.launch
│   ├── costmap_common_params.yaml
│   ├── dwa_local_planner_params.yaml
│   ├── global_costmap_params.yaml
│   ├── global_planner_params.yaml
│   ├── local_costmap_params.yaml
│   └── move_base.launch
├── scripts
│   └── main.py
└── turtlebot3_gazebo
    ├── CHANGELOG.rst
    ├── CMakeLists.txt
    ├── include
    ├── launch
    ├── models
    ├── package.xml
    ├── rviz
    ├── src
    └── worlds
```

图 5 文件结构

- **运行导航功能**

导航相关命令如下，在 test_ws 目录下执行命令之后会出现仿真场景和 rviz，点击 rviz 导航栏的“2D Nav Goal”，然后点击地图上的任意空白地点之后机器人会自动导航至目标点，并且 rviz 和仿真地图中机器人的动作在没有系统延时的时候会保持一致，如图 6 所示。

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
roslaunch my_navi turtlebot_navigation.launch
```

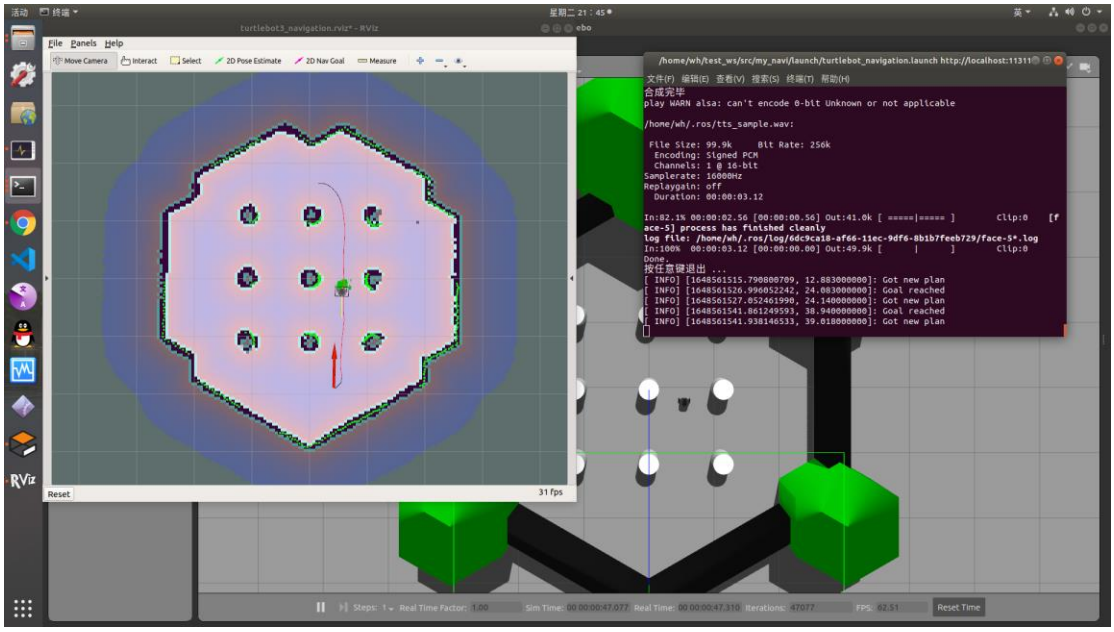

图 6 导航结果

动手实现 1：

在之前的导航参数上，尝试修改导航配置文件，查看导航效果，同时复习之前学习的命令知识，例如查看话题信息，查看 tf 坐标等。