

《移动机器人开发》实训

jeremy.dong@tonyrobotics.com
济南汤尼机器人科技有限公司

2018 年 4 月 25 日

目录

1 课程概述	3
1.1 课程目标	3
1.2 课程设备	3
1.2.1 Abel 移动机器人开发学习平台	3
1.2.2 RoboWare 机器人开发平台	4
1.3 基础知识	4
1.4 开发目标	4
2 基础开发	5
2.1 网络配置及测试	5
2.1.1 PC 及主控板卡 IP 获取	5
2.1.2 Abel 主控板卡配置	6
2.1.3 PC 配置	7
2.1.4 ROS 节点通讯测试	8
2.1.5 RoboWare 远程参数配置	10
2.2 创建 ROS 工作空间及远程部署	12
2.2.1 创建 ROS 工作空间	12
2.2.2 添加 Abel 开发相关 ROS 源码到工作空间	14
2.2.3 远程参数配置	15
2.2.4 远程部署及编译	15
2.2.5 导出 ROS 工作区环境变量	16
2.3 移动机器人里程计构建及 Base Controller 功能测试	17
2.3.1 移动机器人里程计原理介绍	17
2.3.2 移动机器人里程计构建	20
2.3.3 Base Controller 功能测试	21
2.3.4 机器人 Base Controller 参数校准	23

2.3.5 直线运动校准	25
2.3.6 旋转运动校准	25
3 功能开发	26
3.1 使用里程计实现机器人的运动	26
3.1.1 以时间与速度为参考的运动	26
3.1.2 以里程计为参考的运动	30
3.1.3 以里程计为参考的多路径点运动	34
3.2 实现移动机器人的地图构建功能	39
3.2.1 激光雷达驱动安装及测试	39
3.2.2 环境地图构建	41
3.3 实现移动机器人的全自主导航功能	45
3.3.1 使用 AMCL 实现移动机器人的定位功能	46
3.3.2 使用 MoveBase 实现移动机器人的自主导航	49
3.3.3 移动机器人的全自主导航	57

1 课程概述

1.1 课程目标

通过在移动机器人开发学习平台 Abel 上的一系列的开发工作，使学生能够学习和实践移动机器人相关的内容，包含网络环境搭建、里程计构建、SLAM、建图、定位、导航、避障等。

1.2 课程设备

本课程将用到以下的实训设备：Abel 移动机器人开发学习平台，激光雷达，PC（装有 Ubuntu16.04 系统、ROS Kinetic、RoboWare 机器人开发平台）。

1.2.1 Abel 移动机器人开发学习平台

Abel 是济南汤尼机器人科技有限公司专为高校机器人教育打造的一款高性价比的移动机器人开发学习平台，可以帮助高校打造机器人专业的课程，使学生系统的学习移动机器人相关的内容。

Abel 采用二轮差速结构，搭载无线路由器、装有 ROS (Robot Operation System, 机器人操作系统) 的主控板卡、TRD05 电机驱动、电源管理模块，并支持激光雷达、深度相机的接入（这里将会使用镭神 ls01d 激光雷达，奥比中光 Orbbec Astra 深度相机）。另外，Abel 具有开源、免费的软件开发包，提供电机驱动、激光雷达驱动、深度相机驱动、建图算法、导航避障算法等代码。

Abel 模块关系如图 1 所示：

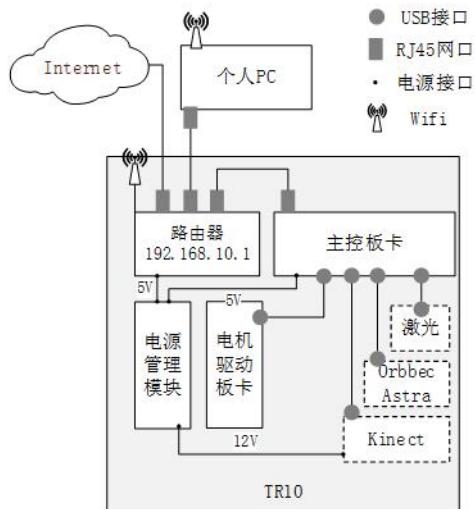


图 1.1: Abel 模块连接示意图

1.2.2 RoboWare 机器人开发平台

RoboWare 是济南汤尼机器人为机器人开发打造的平台，其中 RoboWare Studio 是专门针对 ROS 开发的集成开发环境，可方便的进行 ROS 包的创建、编辑、编译、远程部署及调试等。PC 端已经安装了 RoboWare Studio，可对 Abel 移动机器人开发学习平台进行远程开发及部署。

1.3 基础知识

本课程所需的基础知识为：C++ 或 Python 编程基础，ROS（Robot Operation System，机器人操作系统）基础。

1.4 开发目标

本课程将帮助学生在移动机器人开发学习平台 Abel 上实现以下功能的开发：

- A 使用里程计实现机器人的运动
- B 实现移动机器人的地图构建功能
- C 实现移动机器人的自主导航功能

通过以上功能的开发，学生能够学习和实践移动机器人相关的内容，包含网络环境搭建、里程计构建、SLAM、建图、定位、导航、避障等。

2 基础开发

基础开发部分将完成移动机器人 Abel 的基础功能测试与开发，使 Abel 能够达到比较好的运行状态，主要包括网络配置及测试、工作空间创建及远程部署、移动机器人里程计构建及 Base Controller 功能测试、Base Controller 参数校准等内容。基础开发完成之后，机器人应该具备比较好的运动精度，能够比较精确的按照要求的速度运行。

2.1 网络配置及测试

2.1.1 PC 及主控板卡 IP 获取

Abel 自带路由器，可以构建包含一台 Abel 与一台 PC 的局域网，实现在 PC 端对 Abel 的远程开发。路由器参数为：

表 1: Abel 自带路由器参数表

路由器网关:	192.168.5.1
路由器登录密码:	12345678
Wifi 名称:	Abel-x (Wifi 名称中的 x 为实训现场使用的 Abel 的编号)
密码:	12345678

路由器包括 2 个 LAN 口，1 个 WAN 口。可将 WAN 口连接到外网接入 Internet，本课程只需构建包含 Abel 与 PC 的局域网，因此不需要对路由器做连接外网的配置。

默认 Abel 主控板卡连接到 LAN 口，IP 地址为 192.168.5.x，可通过登录路由器的方式获取到其具体的 IP 地址：

A 将 PC 连接到路由器 Wifi，Wifi 名称为 Abel-x (x 为实训现场使用的 Abel 的编号)，密码为 12345678。连入 Wifi 之后可在 Wifi 属性中查看 PC 的 IP 地址。

B 在 PC 上打开浏览器，登录到 192.168.5.1，登录密码为：12345678。登录后，点击“路由状态”“终端状态”，可在网线连网设备列表中看到 Abel 主控板卡的 IP 地址。



图 2.1: 路由器获取 Abel 主控板卡 IP 示意图

2.1.2 Abel 主控板卡配置

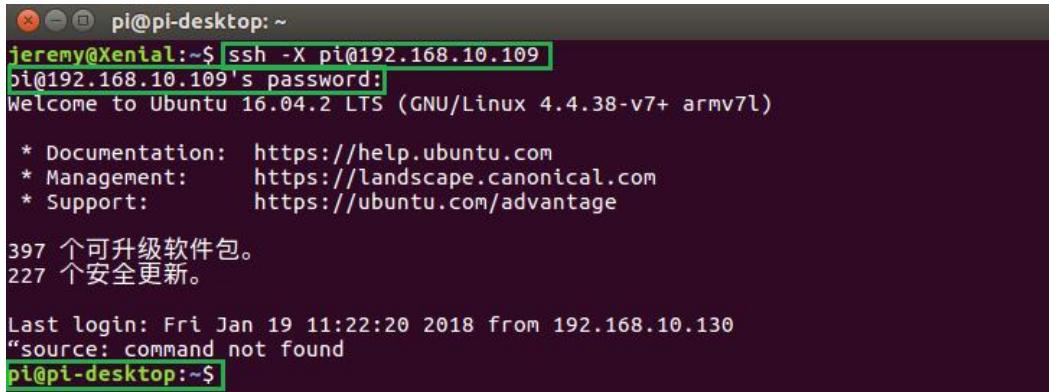
设置 Abel 主控板卡作为 ROS Master，运行 roscore。

默认情况下，Abel 主控板卡的主机名（hostname）为 pi-desktop，IP 地址在 2.1.1 节中已经获取，这里假设 IP 地址为 192.168.5.109。

在 PC 端 Ctrl+Alt+t 打开终端，输入以下指令，SSH 登录到 Abel 主控板卡：

```
ssh -X pi@192.168.5.109
```

指令输入完成后，按 Enter 键后会出现提示输入密码的消息，随后输入系统登录密码（默认为 1），登录成功后会出现以下画面，进入 pi@pi-desktop 工作：



```
pi@pi-desktop: ~
jeremy@Xenial:~$ ssh -X pi@192.168.10.109
pi@192.168.10.109's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.38-v7l armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

397 个可升级软件包。
227 个安全更新。

Last login: Fri Jan 19 11:22:20 2018 from 192.168.10.130
"source: command not found
pi@pi-desktop:~$
```

图 2.2: 路由器获取 Abel 主控板卡 IP 示意图

首先配置/etc/hosts 文件。在打开的终端中，输入以下指令，使用 gedit 工具打开/etc/hosts 文件：

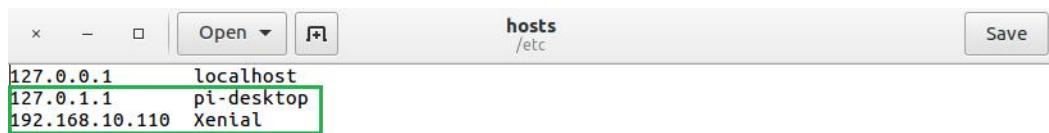
```
sudo gedit /etc/hosts
```

指令输入完成后，按 Enter 键，会出现提示输入密码消息，然后输入系统登录密码（默认为 1），按 Enter 键打开文件：



```
pi@pi-desktop:~$ sudo gedit /etc/hosts
[sudo] password for pi:
```

在打开的/etc/hosts 文件中添加以下内容，输入完成后，进行保存，随后关闭文件：



然后配置 /.bashrc 文件。在已经打开的终端中，输入以下指令，使用 gedit 工具打开 /.bashrc 文件：

```
gedit /.bashrc
```

```
pi@pi-desktop:~$ gedit ~/.bashrc
```

指令输入完成后，按 Enter 键，执行操作：

在打开的 /.bashrc 文件中添加以下内容，输入完成后，进行保存，随后关闭文件：



配置完成后，CTRL+D 关闭所有终端。

2.1.3 PC 配置

PC 的主机名为 Xenial，IP 地址在 2.1.1 节中已经获取，这里假设 IP 地址为 192.168.5.110。

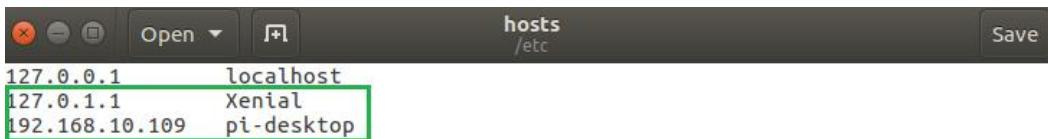
首先配置 /etc/hosts 文件。在 PC 端 Ctrl+Alt+t 打开终端，输入以下指令，使用 gedit 工具打开 /etc/hosts 文件。

```
sudo gedit /etc/hosts
```

指令输入完成后，按 Enter 键后，会出现提示输入密码的消息，然后输入系统登录密码，按 Enter 键打开文件：

```
jeremy@Xenial:~$ sudo gedit /etc/hosts
[sudo] password for jeremy: ■
```

在打开的 /etc/hosts 文件中添加以下内容，输入完成后，进行保存，随后关闭文件：



然后配置 /.bashrc 文件。在已经打开的终端中，输入以下指令，使用 gedit 工具打开 /.bashrc 文件： gedit /.bashrc 指令输入完成后，按 Enter 键，执行操作：

```
jeremy@Xenial:~$ gedit ~/.bashrc
```

在打开的 /.bashrc 文件中添加以下内容，输入完成后，进行保存，随后关闭文件：



配置完成后，CTRL+D 关闭所有终端。

2.1.4 ROS 节点通讯测试

首先，在 PC 端 $\text{Ctrl}+\text{Alt}+\text{t}$ 打开终端，输入以下指令（注意与 2.1.1 节指令相比，这里没有-X 参数），SSH 登录到 Abel 主控板卡：

```
ssh pi@192.168.5.109
```

指令输入完成后，按 Enter 键后会出现提示输入密码的消息，随后输入系统登录密码（默认为 1），登录成功后会出现以下画面，进入 `pi@pi-desktop` 工作：

```
pi@pi-desktop: ~
jeremy@Xenial:~$ ssh pi@192.168.10.109
pi@192.168.10.109's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.38-v7+ armv7l)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

397 个可升级软件包。
227 个安全更新。

Last login: Fri Jan 19 11:49:45 2018 from 192.168.10.110
pi@pi-desktop:~$
```

图 2.3: SSH 登录 Abel 主控板卡操作示意图

在终端中输入以下命令启动 `roscore`：

```
roscore
```

成功启动后会出现以下画面：

```
pi@pi-desktop:~$ [roscore]
... logging to /home/pi/.ros/log/3562b426-fcca-11e7-aff1-b827eba26efc/roslaunch-pi-desktop-1822.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://pi-desktop:36089/
ros_comm version 1.12.12

SUMMARY
=====
PARAMETERS
  * /rosdistro: kinetic
  * /rosversion: 1.12.12
NODES

auto-starting new master
process[master]: started with pid [1833]
ROS_MASTER_URI=http://pi-desktop:11311/
setting /run_id to 3562b426-fcca-11e7-aff1-b827eba26efc
process[rosout-1]: started with pid [1846]
started core service [/rosout]
```

图 2.4: `roscore` 启动成功示意图

然后，在 PC 端打开新的终端，输入以下指令启动 `turtlesim_node` 节点：

```
rosrun turtlesim turtlesim_node
```

```
tr10@Xenial: ~
tr10@Xenial:~$ rosrun turtlesim turtlesim_node
```

节点成功启动后，会出现一只小乌龟（小乌龟的种类各不相同），如下图所示：

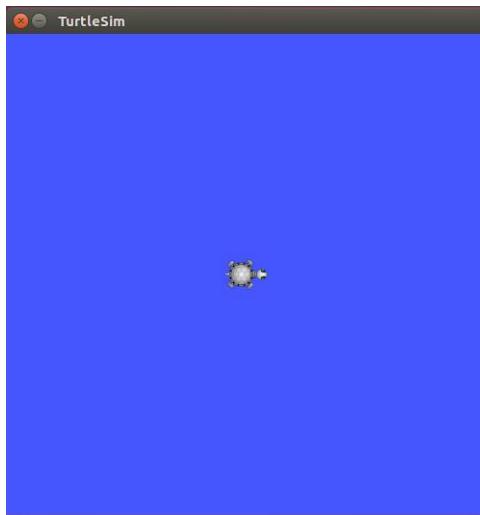


图 2.5: 节点 turtlesim_node 运行成功示意图

接下来，在 PC 端打开新的终端，按照前面介绍的方法 SSH 登录到 Abel 主控板卡，启动 turtle_teleop_key 节点，用于向小乌龟发出控制指令：

```
rosrun turtlesim turtle_teleop_key
```

```
pi@pi-desktop:~$ rosrun turtlesim turtle_teleop_key
```

节点成功启动后，出现以下画面，可以按照提示，通过 PC 键盘控制小乌龟运行：

```
pi@pi-desktop:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
-----
Use arrow keys to move the turtle.
```

如果小乌龟能够运动，那么就证明 PC 与 Abel 主控板卡的 ROS 网络通讯正常：



图 2.6: PC 键盘控制小乌龟运动示意图

节点通讯完成之后，CTRL+C 关闭所有程序，CTRL+D 关闭所有终端。

2.1.5 RoboWare 远程参数配置

为了更加方便的在 PC 端对 Abel 主控板卡进行远程开发，需要对 PC 和 Abel 主控板卡进行远程参数配置。

(1) 配置 SSH 公钥无密登录

首先，在 PC 端生成公钥和私钥。打开终端，执行以下命令：

```
ssh-keygen
```

一直按 Enter 键选择默认选项，会在 /.ssh 目录下生成 id_rsa 和 id_rsa.pub 两个文件。

```
jeremy@Xenial:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jeremy/.ssh/id_rsa):
/home/jeremy/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jeremy/.ssh/id_rsa.
Your public key has been saved in /home/jeremy/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:vfgB0hKMj1Fzo2BucwgZdweQE682pJWYX9ePGlz+J4E jeremy@Xenial
The key's randomart image is:
+---[RSA 2048]----+
| . ++o..
| *o+ . .
|= *.= + o
| O O * + =
| . % = o E +
| o B . + + o
| . o + . = .
| . . . +
+---[SHA256]----+
jeremy@Xenial:~$
```

图 2.7: PC 端生成公钥和私钥操作示意图

然后执行以下指令，将 id_rsa.pub 文件复制到 Abel 主控板卡（注意，将 PC-username 替换为所用 PC 的用户名，pi@192.168.5.109 则替换为所用的 Abel 的用户名和 IP 地址）：

```
scp /home/<PC-username>/.ssh/id_rsa.pub pi@192.168.5.109:/home/pi
```

指令输入完成后，按 Enter 键执行成功后会出现以下画面：

```
jeremy@Xenial:~$ scp /home/jeremy/.ssh/id_rsa.pub pi@192.168.10.109:/home/pi
pi@192.168.10.109's password:
```

图 2.8: 将公钥文件远程拷贝到 Abel 主控板卡操作示意图

将公钥文件 id_rsa.pub 拷贝到 Abel 主控板卡之后，SSH 登录到 Abel 主控板卡：

```
ssh pi@192.168.5.109
```

将 id_rsa.pub 的文件内容追加写入到 Abel 主控板卡的 /.ssh/authorized_keys 文件中:

```
cat id_rsa.pub » /.ssh/authorized_keys
```

修改 authorized_keys 文件的权限:

```
chmod 600 /.ssh/authorized_keys
```

```
jeremy@Xenial:~$ ssh pi@192.168.10.109
pi@192.168.10.109's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.38-v7+ armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

397 个可升级软件包。
227 个安全更新。

Last login: Fri Jan 19 12:14:29 2018 from 192.168.10.130
pi@pi-desktop:~$ cat id_rsa.pub >> ~/.ssh/authorized_keys
pi@pi-desktop:~$ chmod 600 ~/.ssh/authorized_keys
```

图 2.9: 公钥追加写入及 authorized_keys 文件权限修改操作示意图

配置完成后，再登录 Abel 主控板卡就无需输入密码了。再次 SSH 登录 Abel 主控板卡会出现以下画面，这次不再提示输入系统登录密码，而是直接登录：

```
jeremy@Xenial:~$ ssh pi@192.168.10.109
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.38-v7+ armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

397 个可升级软件包。
227 个安全更新。

Last login: Fri Jan 19 12:27:52 2018 from 192.168.10.130
pi@pi-desktop:~$
```

图 2.10: 配置 SSH 公钥后无密登录示意图

接下来，即可在 PC 端用 RoboWare Studio 对主控板卡进行远程开发和调试。

(2) 修改 Abel 主控板卡/etc/profile

在 PC 端打开新的终端，SSH 登录到 Abel 主控板卡:

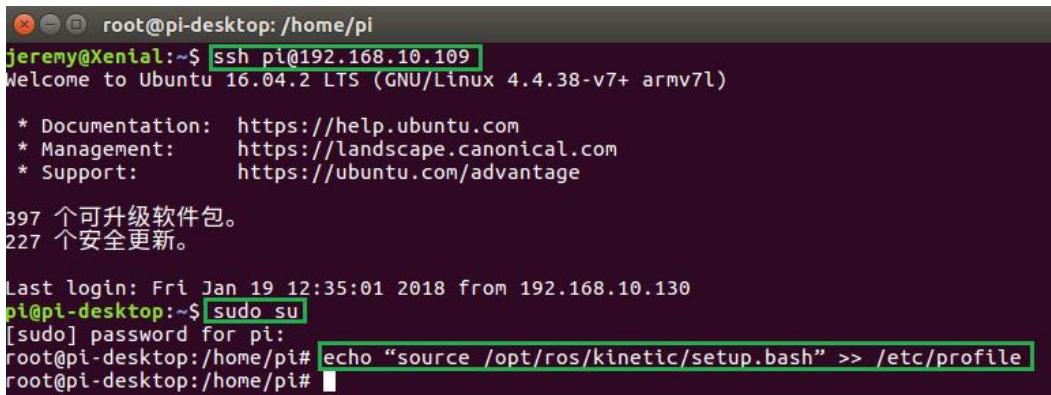
```
ssh pi@192.168.5.109
```

登录后切换到 root 用户权限:

```
sudo su
```

将 ROS 环境变量信息写入到/etc/profile 文件中:

```
echo "source /opt/ros/kinetic/setup.bash" » /etc/profile
```



```

root@pi-desktop:~$ ssh pi@192.168.10.109
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.38-v7+ armv7l)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

397 个可升级软件包。
227 个安全更新。

Last login: Fri Jan 19 12:35:01 2018 from 192.168.10.130
pi@pi-desktop:~$ sudo su
[sudo] password for pi:
root@pi-desktop:/home/pi# echo "source /opt/ros/kinetic/setup.bash" >> /etc/profile
root@pi-desktop:/home/pi#

```

图 2.11: 将 ROS 环境变量信息写入到/etc/profile 文件操作示意图

配置完成后，**CTRL+D** 关闭所有终端。

2.2 创建 ROS 工作空间及远程部署

这一部分将利用 RoboWare Studio 创建 ROS 工作空间，并将移动机器人 Abel 开发的相关代码导入工作空间中，随后将代码远程部署到 Abel 主控板卡。

2.2.1 创建 ROS 工作空间

首先在 File 下拉菜单中选择 New Workspace 选型，然后在弹出的对话框中选择 home 目录，在 Name 行写入 tony_ws，随后点击 Save 即可创建命名为 tony_ws 的 ROS 工作空间：

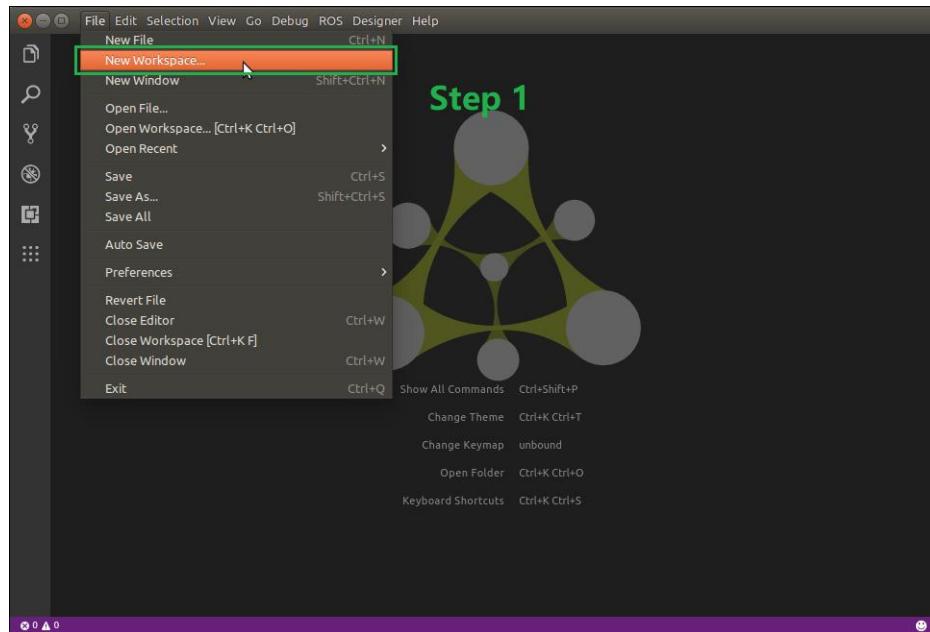


图 2.12: 使用 RoboWare Studio 创建 ROS 工作空间操作示意图 1

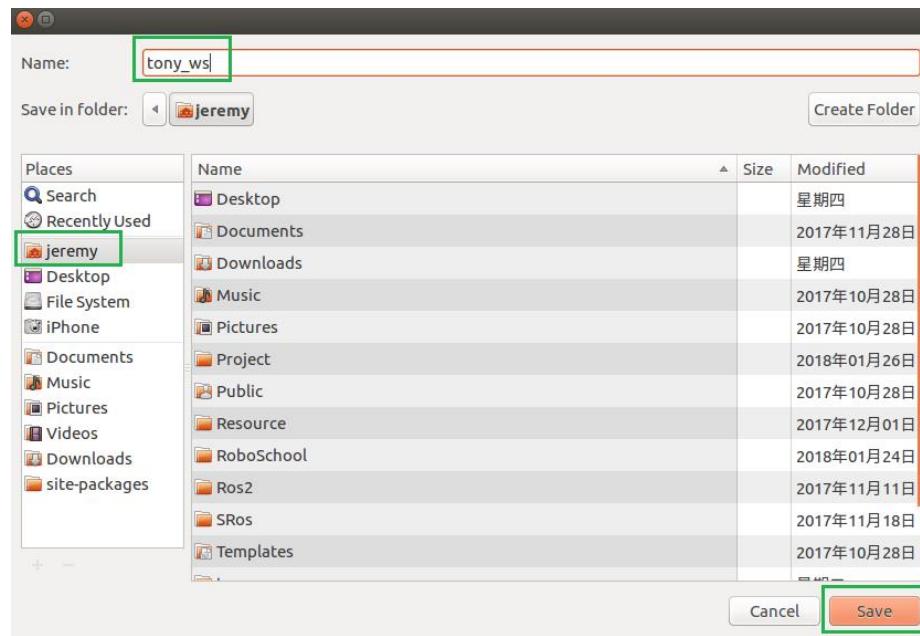


图 2.13: 使用 RoboWare Studio 创建 ROS 工作空间操作示意图 2

成功使用 RoboWare Studio 创建 ROS 工作后，会出现如下图所示的内容，工作空间中当前只有一个包含 CMakeLists.txt 文件的 src 文件夹：

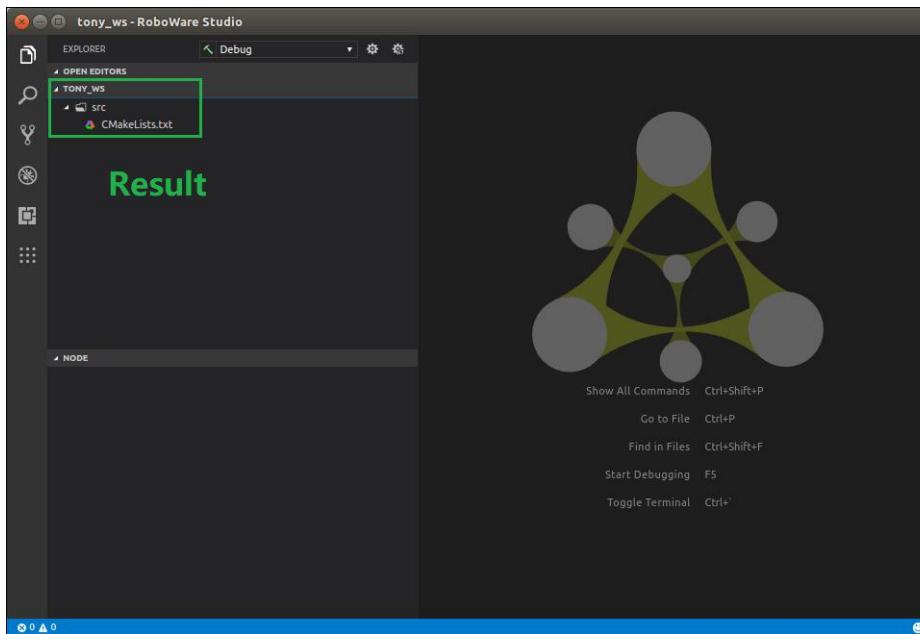


图 2.14: 使用 RoboWare Studio 创建成功的 ROS 工作空间示意图

2.2.2 添加 Abel 开发相关 ROS 源码到工作空间

工作空间创建完成之后，按照以下操作将 Abel 开发相关的 ROS 源码放入所创建的 tony_ws 工作空间中，首先在 View 下拉菜单中选择 Integrated Terminal 选项：

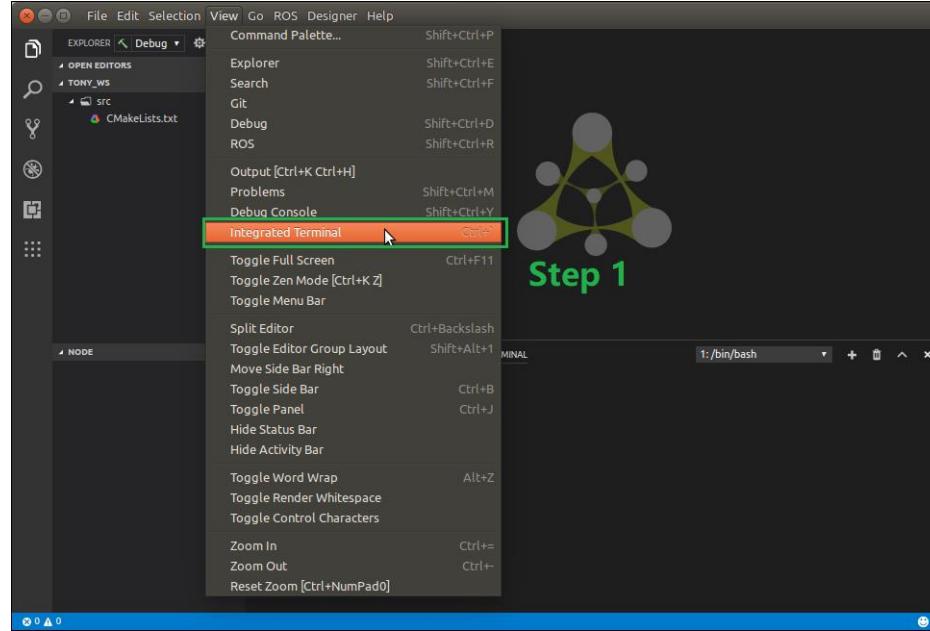


图 2.15: 在 RoboWare Studio 中打开集成终端示意图

在打开的集成终端中依次输入下图中所示的命令，将 Abel 开发相关的源代码复制到之前创建的工作空间 tony_ws 的 src 目录下，复制之后，会看到 tony_ws 工作空间下的 src 目录下多了 Abel 文件夹，其中包含 Abel 开发相关的 Abel_controller, Abel_navigation, Abel_sensor 等 packages，在后续的开发工作中，我们会用到这些 packages：

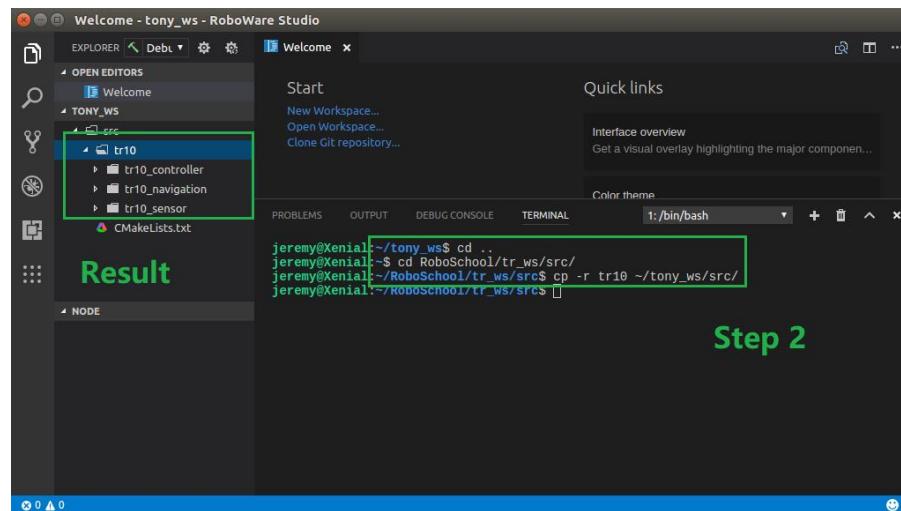


图 2.16: 将 Abel 开发相关代码导入 tony_ws 工作空间操作示意图

2.2.3 远程参数配置

在将 PC 的 tony_ws 工作空间部署到 Abel 主控板卡之前，需要先配置 Abel 主控板卡的远程参数。按照以下提示依次设置 Abel 主控板卡 IP 地址、用户名、PC 秘钥文件名、工程在 Abel 主控板卡的部署路径等参数：

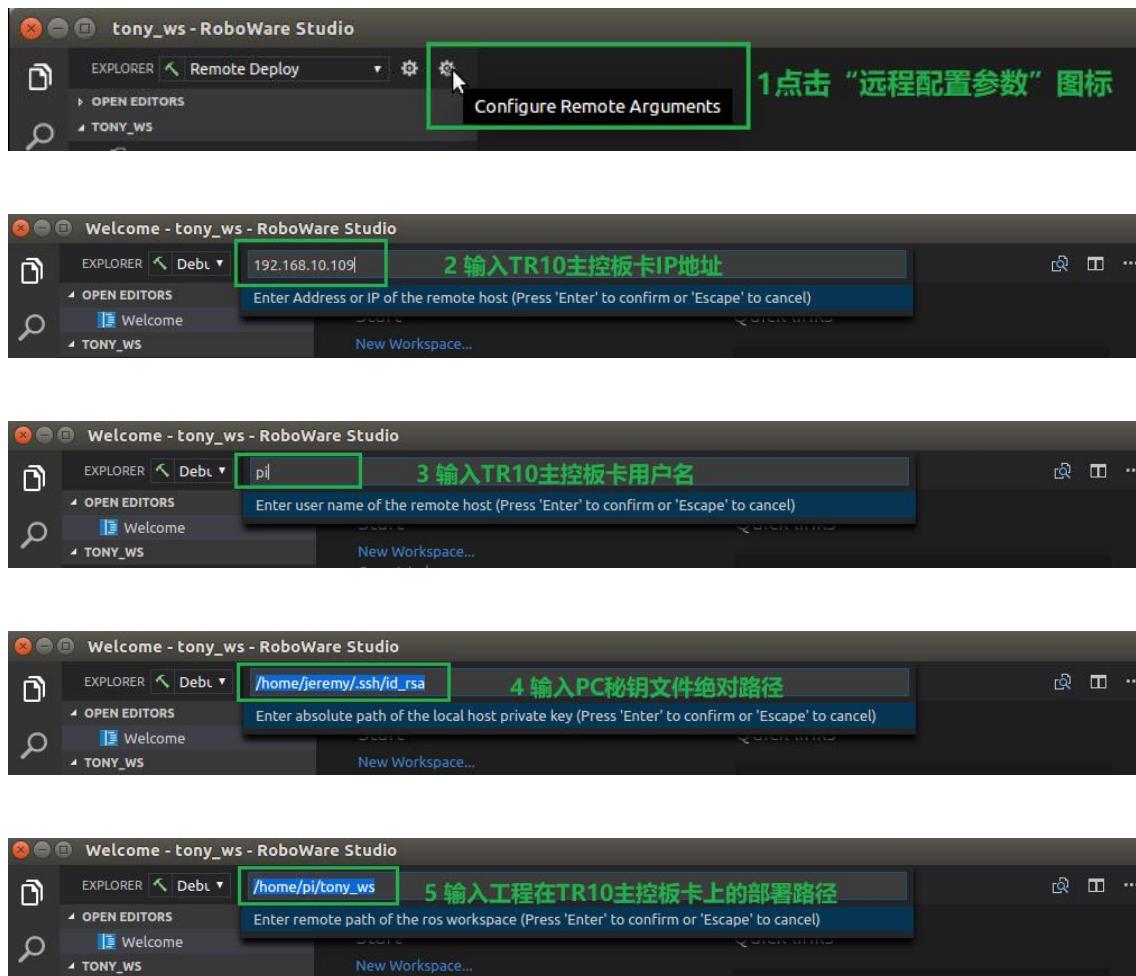


图 2.17: RoboWare Studio 远程参数配置操作示意图

2.2.4 远程部署及编译

远程参数设置完成之后，在下图所示的下拉列表中选中“Remote Deploy”远程部署选型，然后点击左侧的锤子图表进行部署。RoboWare 的输出窗口会显示部署情况，远程部署成功后，会出现下图中类似于 Result 的消息：

远程部署完成后，在下拉列表中选中“Release(remote)”远程编译选型，然后点击左侧的锤子图标进行远程编译。RoboWare 的输出窗口会显示编译情况，远程编译成功之后，会出现下图中类似于 Result 的消息：

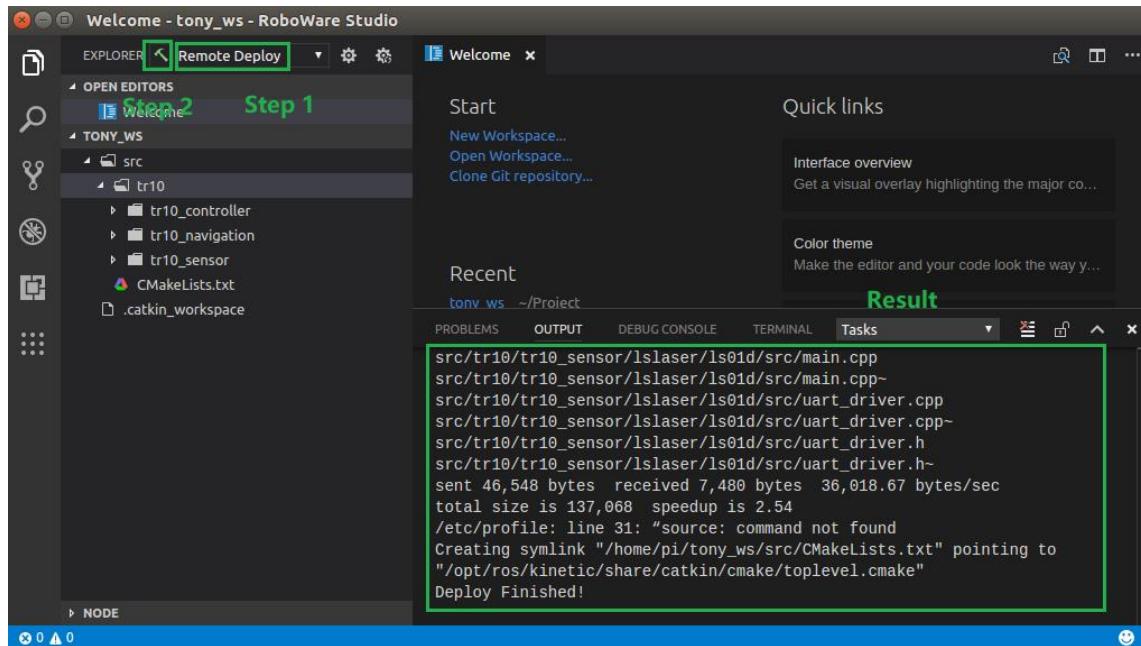


图 2.18: 使用 RoboWare Studio 进行远程部署示意图

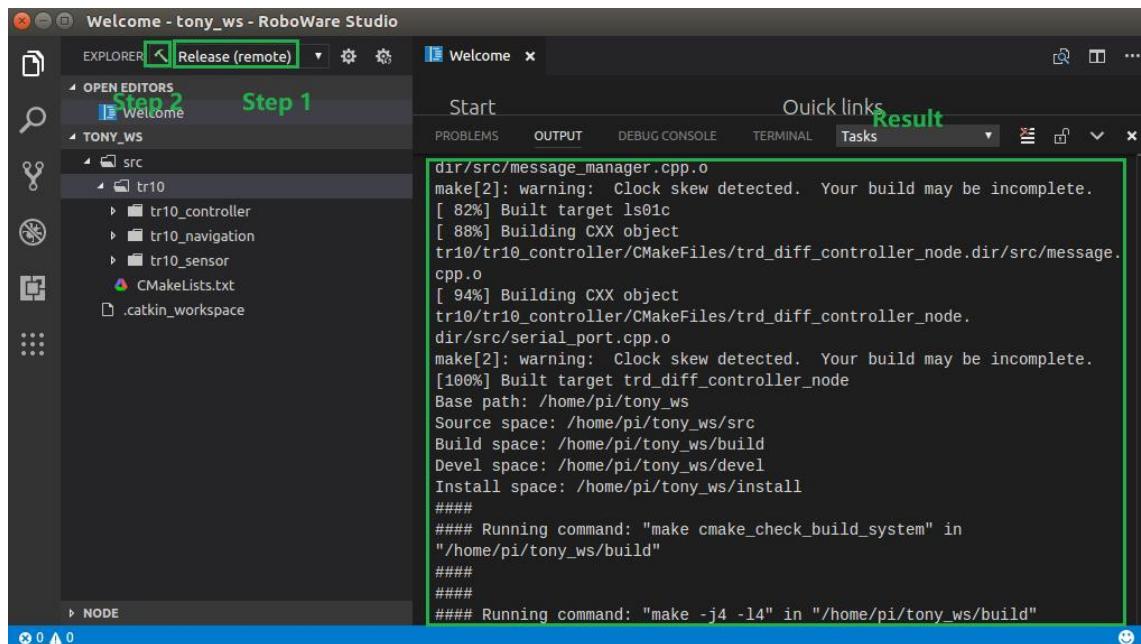


图 2.19: 使用 RoboWare Studio 进行远程编译示意图

2.2.5 导出 ROS 工作区环境变量

在 RoboWare Studio 中，选择“ROS” “Open Remote ./bashrc”，将以下内容加入到.bashrc 文件中并保存：

```
source /tony_ws/devel/setup.bash
```

这样，当 Abel 主控板卡启动 ROS 后，PC 的 ROS 节点也可运行。

```

File Edit Selection View Go Debug ROS Designer Help
EXPLORER Release (remote) ⚙ ...
OPEN EDITORS .bashrc .vscode
TONY_WS
  SRC
    tr10
      tr10_controller
      tr10_navigation
      tr10_sensor
      CMakeLists.txt
      .catkin_workspace
  NODE
    ls01c
    ls01d
    tr10_controller
Build Shift+Ctrl+B
Open ~./.bashrc
Run roscore
Run RViz
Run rqt
Run rqt-reconfigure
Run rqt-graph
Open Remote ~./.bashrc Step 1
Run Remote roscore
112 if [ -f /usr/share/bash-completion/bash_completion ]; then
113   . /usr/share/bash-completion/bash_completion
114 elif [ -f /etc/bash_completion ]; then
115   . /etc/bash_completion
116 fi
117
118
119 source /opt/ros/kinetic/setup.bash Step 2
120 source ~/tony_ws/devel/setup.bash
121
122 export ROS_HOSTNAME=pi-desktop
123 export ROS_MASTER_URI=http://192.168.10.109:11311
124
125
126
127
Ln 120, Col 34 Spaces: 4 UTF-8 LF Shell Script (Bash)

```

图 2.20: 导出 ROS 工作区环境变量操作示意图

保存后，即完成了 Abel 主控板卡的部署和配置，可进行下一步的功能调试和开发。

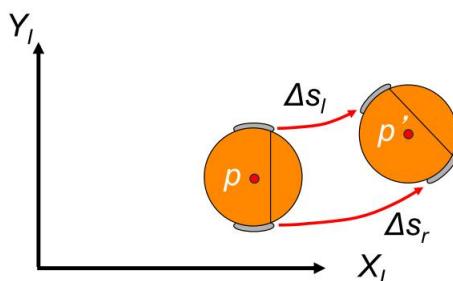
2.3 移动机器人里程计构建及 Base Controller 功能测试

这一部分将构建 Base Controller 的重要组成部分——里程计，并实现 PC 远程控制移动机器人 Abel 运动。由于机器人将会产生实际的运动，因此应确保机器人周围有足够的运动空间，以避免碰撞。

2.3.1 移动机器人里程计原理介绍

里程计接收电机的编码器的反馈信号，估计机器人位置变化及速度信息，这些信息可被导航规划器使用，规划出有效的运动指令。

Abel 移动机器人采用的是差分驱动，在这里我们将根据差分驱动模型构建里程计。首先我们来看一下差分驱动模型里程计的原理。



如图 2-21 所示，如果机器人从已知的 p 点开始运动，右侧车轮与左侧车轮走过的距离分别为 ΔS_r 和 ΔS_l ，那么运动一个周期之后的新位置 p' 是什么呢？

图 2.21: 里程计原理示意图 1

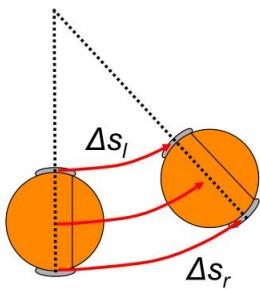


图 2.22: 里程计原理示意图 2

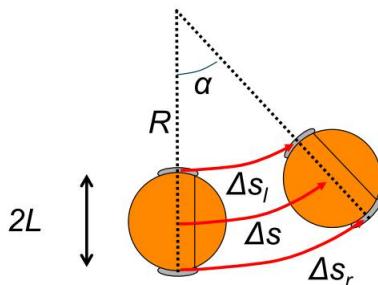


图 2.23: 里程计原理示意图 3

我们首先计算机器人走过的距离 ΔS 和角度的改变量 $\Delta\theta$ 。

如图 2.22 所示，假设机器人中心经过一个恒定半径的圆弧。

如图 2.23 所示，机器人走过的圆弧设定为 ΔS ，圆半径设定为 R ，对应的弧度为 α 。已知差分驱动移动机器人的两轮间距为 $2L$ ，那么可以分别计算出 ΔS_r 、 ΔS_l 和 ΔS ：

$$\Delta S_r = (R + 2L)\alpha$$

$$\Delta S_l = R\alpha$$

$$\Delta S = (R + L)\alpha$$

通过 ΔS_r 和 ΔS_l ，可以分别计算出 $R\alpha$ 与 $L\alpha$ ：

$$R\alpha = \Delta S_l$$

$$L\alpha = (\Delta S_r - R\alpha)/2 = (\Delta S_r)/2 - (\Delta S_l)/2$$

因此可以计算出机器人中心点走过的距离 ΔS 为：

$$\Delta S = (R + L)\alpha = R\alpha + L\alpha$$

$$\Delta S = (\Delta S_l + \Delta S_r)/2$$

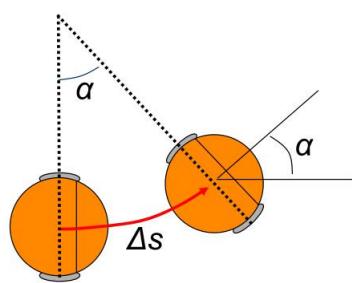


图 2.24: 里程计原理示意图 4

计算完 ΔS 后，接下来我们计算角度的改变量 $\Delta\theta$ 。

通过图 2.24 可以看到，角度的改变量 $\Delta\theta$ 与机器人中心走过的圆弧对应的弧度是一致的：

$$\Delta\theta = \alpha$$

根据 ΔS_r 和 ΔS_l ，我们可以通过以下方式计算出半径 R ：

$$(\Delta S_l)/R = (\Delta S_r)/(R + 2L)$$

$$(R + 2L)\Delta S_l = R\Delta S_r$$

$$2L\Delta S_l = R(\Delta S_r - \Delta S_l)$$

$$R = (2L\Delta S_l)/(\Delta S_r - \Delta S_l)$$

计算出半径 R 之后，可以很容易的计算出弧度 α :

$$\alpha = (\Delta S_l)/R = (\Delta S_r - \Delta S_l)/2L$$

因此机器人的角度改变量为:

$$\Delta\theta = (\Delta S_r - \Delta S_l)/2L$$

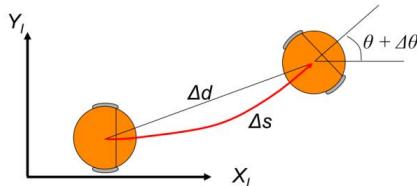


图 2.25: 里程计原理示意图 5

这样我们就计算出了机器人走过的距离 ΔS 和角度的改变量 $\Delta\theta$ ，接下来便可以计算机器人在全局坐标系中的位姿改变。
我们知道当弧度很小的时候，可以近似为直线，因此如图 2.25 所示，我们采用 Δd 代替 ΔS 。

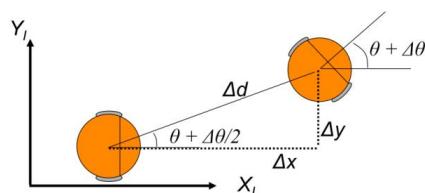


图 2.26: 里程计原理示意图 6

如图 2.26 所示，我们可以近似计算出机器人中心点在全局坐标系中的位置变化 Δx 与 Δy :

$$\Delta x = \Delta S \cos(\theta + \Delta\theta/2)$$

$$\Delta y = \Delta S \sin(\theta + \Delta\theta/2)$$

假如机器人在 p 点的全局位姿为:

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

那么经过一个周期的运动之后，新位置 p' 为：

$$p' = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta S \cos(\theta + \Delta\theta/2) \\ \Delta S \sin(\theta + \Delta\theta/2) \\ (\Delta S_r - \Delta S_l)/2L \end{bmatrix}$$

以上我们根据机器人当前位姿以及编码器返回的数据，求出了机器人经过一个运动周期之后的新的全局位姿 p' ，这就是里程计工作的原理。

下一节中，我们将编写程序实现以上的计算过程，构建里程计。里程计将接收编码器获取的数据，计算出机器人当前的全局位姿，并可以进行逆解算，即将全局运动速度指令分解为两个轮子的运动速度，作为控制指令发送到驱动板卡。

2.3.2 移动机器人里程计构建

在 2.2 节中已经将 Abel 开发相关的 ROS 源代码导入了 tony_ws 工作空间并进行了远程部署，接下来通过 TODO 的形式进行 Abel 移动机器人里程计的构建。

首先使用 RoboWare Studio 打开 Abel/Abel_controller/src 目录下的 trd_diff_controller.cpp 文件，根据提示，完成构建 Abel 里程计部分 (TRDDiffController::publishOdom 函数) 的代码练习 (TODO 练习)。

代码编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果 (在 2.2.4 节已经展示过)。

远程部署完成后，选择“Release(remote)”模式，并点击旁边绿色的小锤子进行编译，编译成功后会出现以下 Result 部分的画面：

```

void TRDDiffController::publishOdom(){
    double dx, dy, dtheta;
    double d, dleft, dright;
    time_current = ros::Time::now();
    double elapsed = (time_current - time_prev).toSec();
    time_prev = time_current;
    dleft = left_coeff * PI * wheel_diameter * (message_manager.encoder_left -
    dright = right_coeff * PI * wheel_diameter * (message_manager.encoder_right
    d = (dleft + dright) / 2;
    dtheta = (dright - dleft) / base_width;
    if(d != 0){
        dx = cos(dtheta) * d;
        dy = -sin(dtheta) * d;
    }
}

[100%] Linking CXX executable
/home/jeremy/tony_ws/devel/lib/tr10_controller/trd_diff_controller_node
[100%] Built target trd_diff_controller_node
Base path: /home/jeremy/tony_ws
Source space: /home/jeremy/tony_ws/src
Build space: /home/jeremy/tony_ws/build
Devel space: /home/jeremy/tony_ws/devel
Install space: /home/jeremy/tony_ws/install
#####
##### Running command: "make /home/jeremy/tony_ws/src -DCMAKE_BUILD_TYPE=Debug
-DCATKIN_DEVEL_PREFIX=/home/jeremy/tony_ws/devel
-DCMAKE_INSTALL_PREFIX=/home/jeremy/tony_ws/install -G Unix Makefiles" in
"/home/jeremy/tony_ws/build"
#####
##### Running command: "make -j2" in "/home/jeremy/tony_ws/build"

```

图 2.27：移动机器人里程计构建及部署操作示意图

2.3.3 Base Controller 功能测试

里程计代码构建完成并编译成功之后，进行 Base Controller 的功能测试。

首先，选择 Abel_controller/launch 目录下的 Abel_controller.launch 文件，点击右键，在弹出的列表中选择“Run Remote Launch File”会运行 launch 文件，启动 Abel Base Controller：

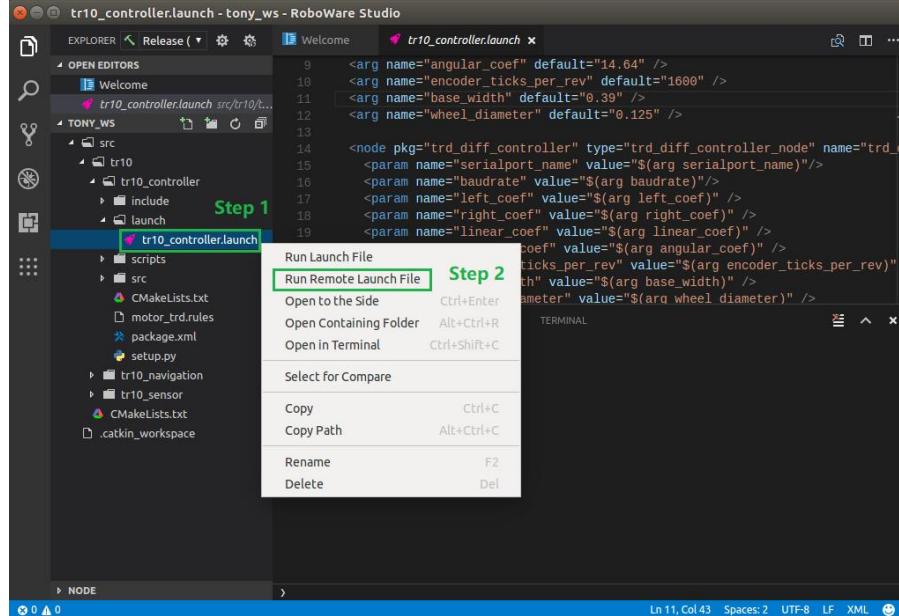


图 2.28: RoboWare Studio 远程启动 Abel_controller 文件操作示意图

Abel Base Controller 启动成功后，将会出现如下图所示的画面：

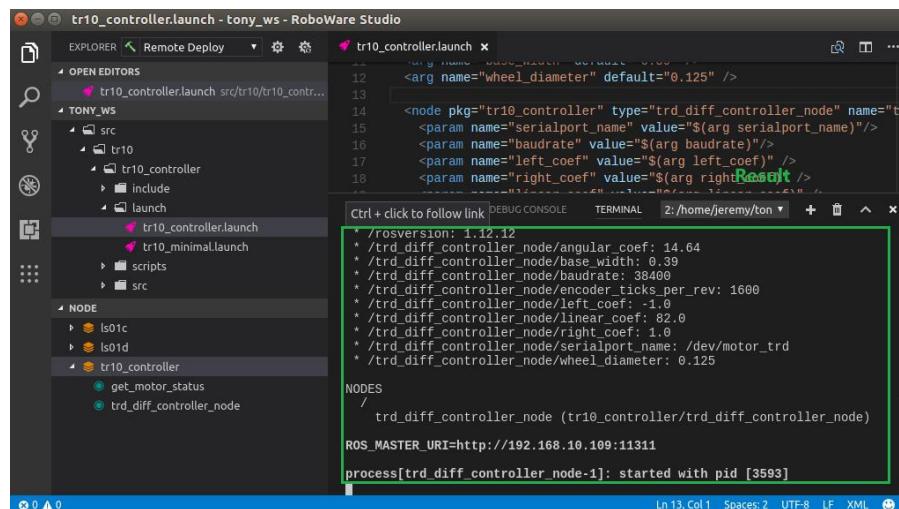


图 2.29: Abel_controller 节点成功启动示意图

然后选择 Abel_controller/scripts 目录下的 keyboard_teleop.py 脚本，点击右键，在弹出的列表中选择“Run Python File in Terminal”会在集成终端中运行该 Python 脚本，

启动键盘控制节点：

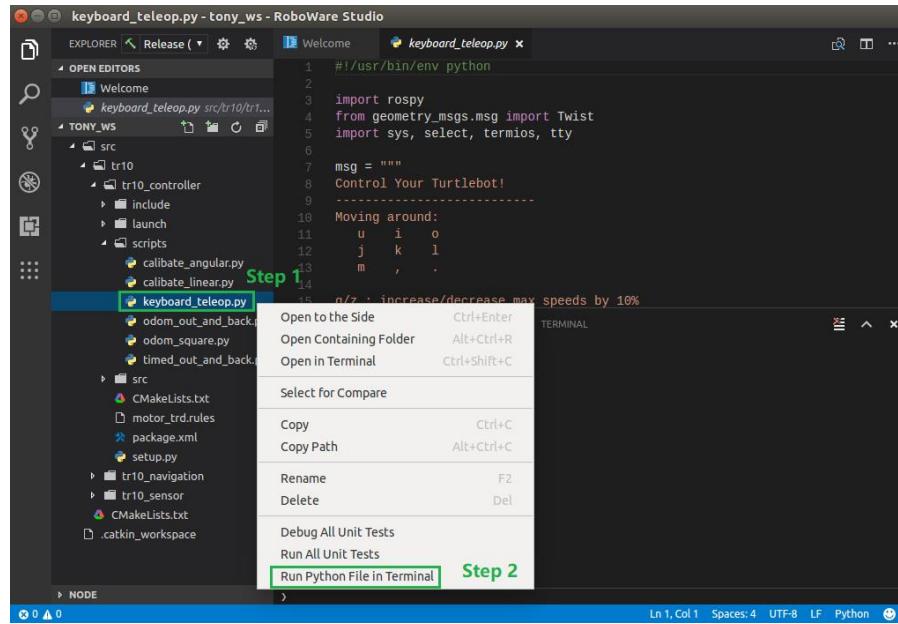


图 2.30: RoboWare Studio 启动键盘控制节点操作示意图

键盘控制节点打开后，终端内会出现相应的操作说明，按照操作说明可以通过 PC 端的键盘远程操控机器人运动。需要注意的是，在运动之前需确保机器人周围有足够的空间，并且开始运动时使用低速运行：

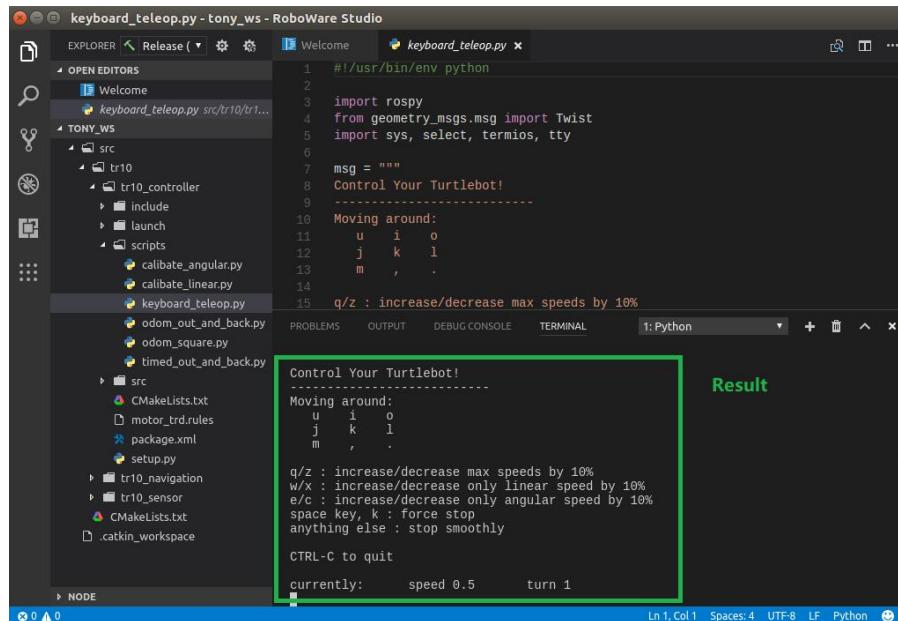


图 2.31: 键盘控制节点成功启动示意图

测试完成之后，**CTRL+C** 结束所有的程序，**CTRL+D** 关闭所有的终端。

2.3.4 机器人 Base Controller 参数校准

这部分将实现机器人控制器参数的校准，以使机器人获取比较可靠的运动精度。

首先启动 Abel Base Controller。选择 Abel_controller/launch 目录下的 Abel_controller.launch 文件，点击右键，在弹出的列表中选择“Run Remote Launch File”会运行 launch 文件，启动 Abel Base Controller：

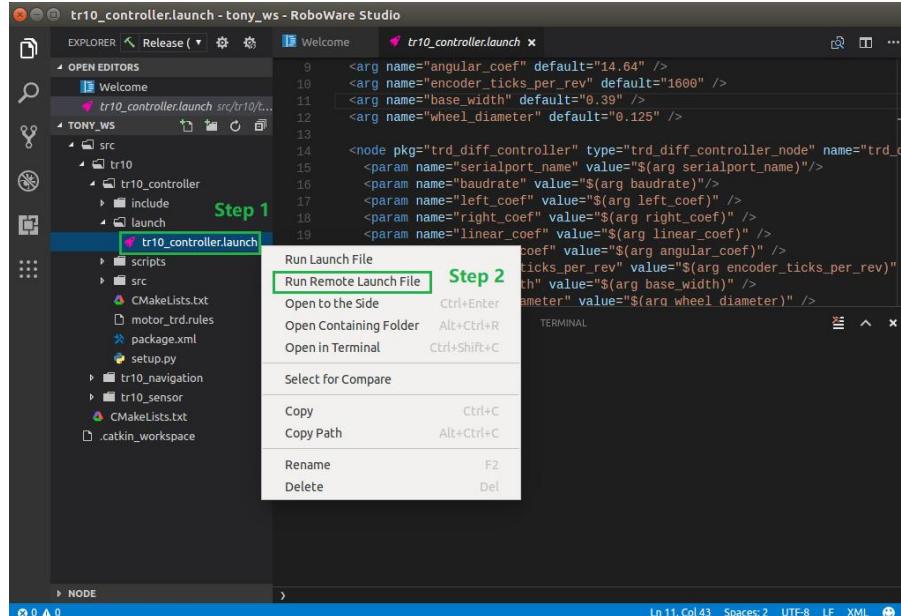


图 2.32: RoboWare Studio 远程启动 Abel_controller 文件操作示意图

Abel Base Controller 启动成功后，将会出现下图所示的画面：

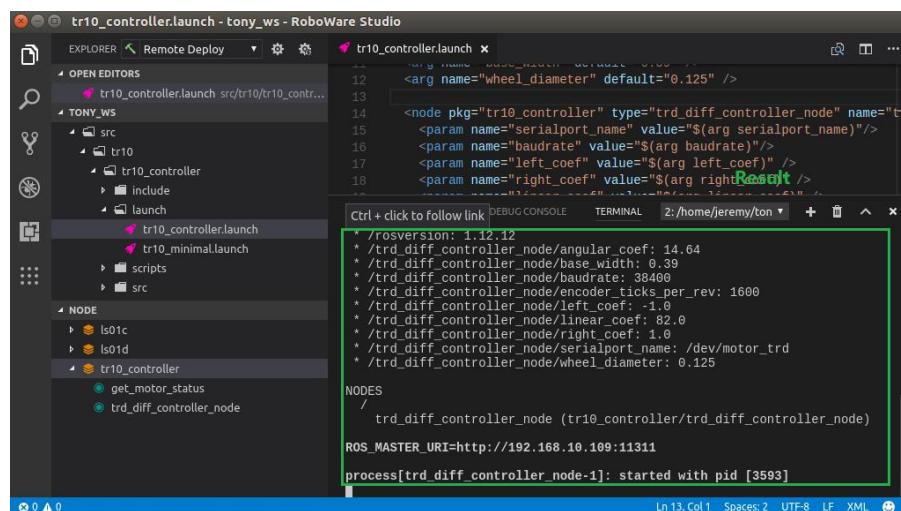


图 2.33: Abel_controller 节点成功启动示意图

然后选择 Abel_controller/scripts 目录下的 keyboard_teleop.py 脚本，点击右键，在弹出的列表中选择“Run Python File in Terminal”会在集成终端中运行该 Python 脚本，

启动键盘控制节点：

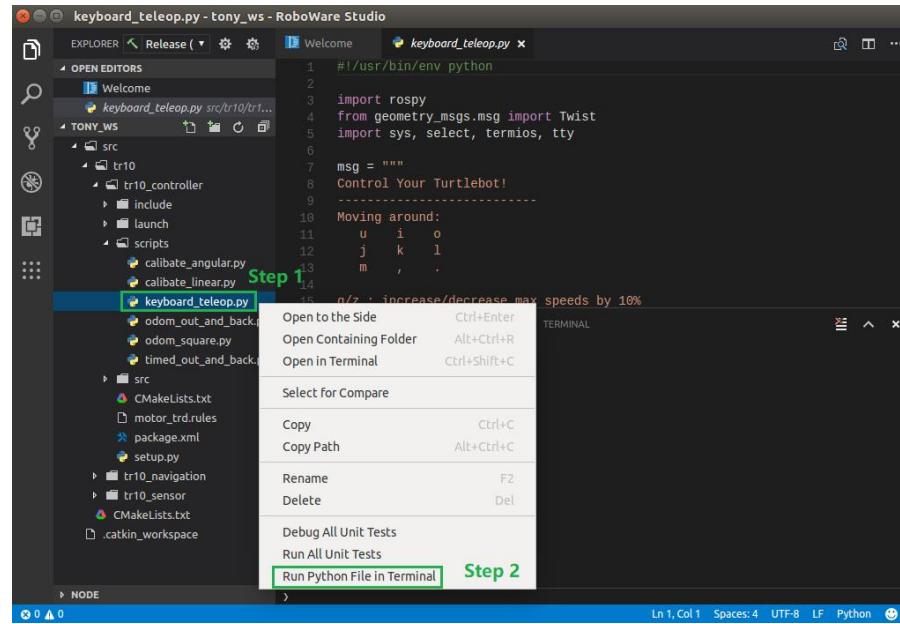


图 2.34: RoboWare Studio 启动键盘控制节点操作示意图

键盘控制节点打开后，终端内会出现相应的操作说明，按照操作说明可以通过 PC 端的键盘远程操控机器人运动。需要注意的是，在运动之前需确保机器人周围有足够的空间，并且开始运动时使用低速运行：

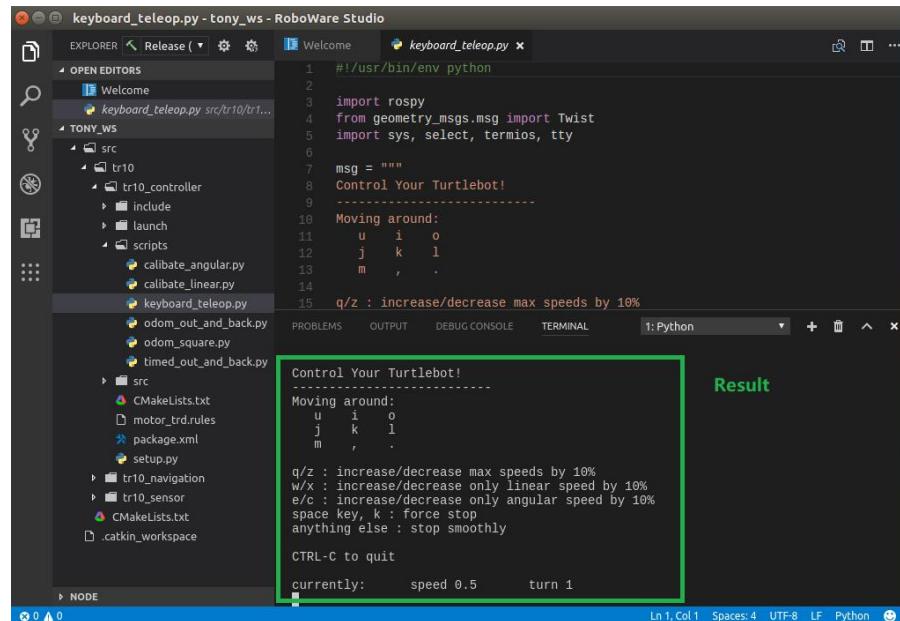
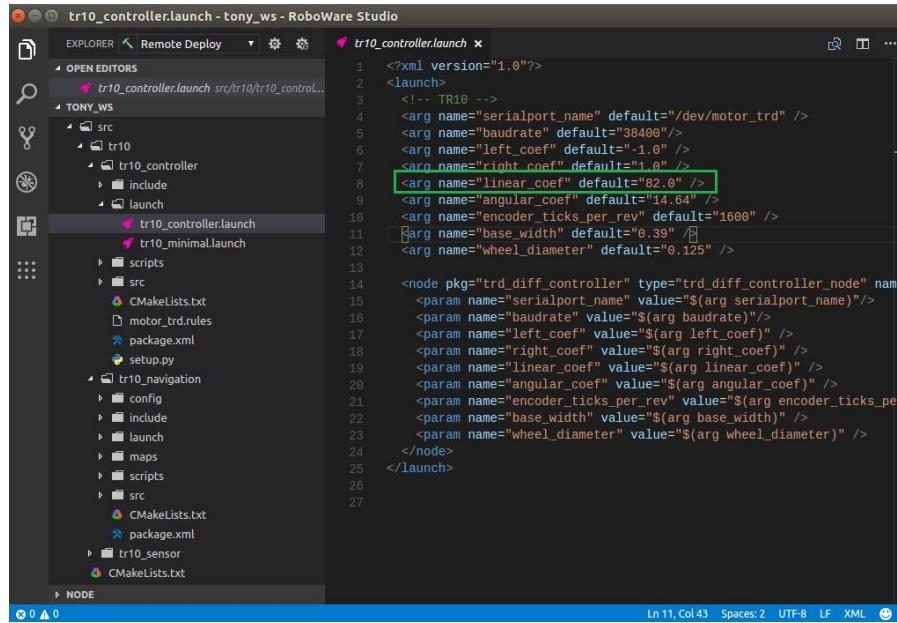


图 2.35: 键盘控制节点成功启动示意图

接下来分别进行直线运动校准及旋转运动校准。

2.3.5 直线运动校准

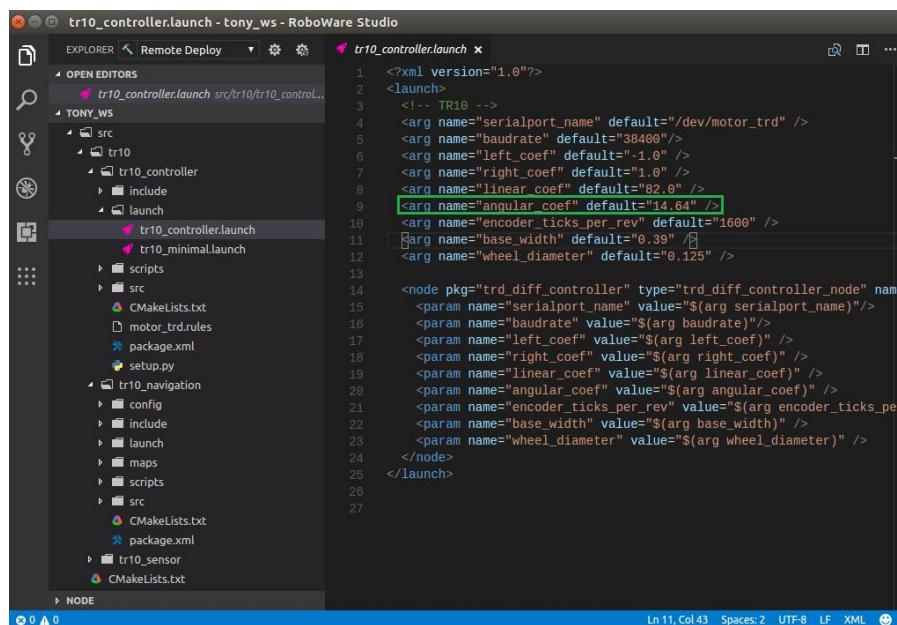


```
<?xml version="1.0"?>
<launch>
  <!-- TR10 -->
  <arg name="serialport_name" default="/dev/motor_trd" />
  <arg name="baudrate" default="38400"/>
  <arg name="left_coef" default="-1.0" />
  <arg name="right_coef" default="1.0" />
  <arg name="linear_coef" default="82.0" /> highlighted
  <arg name="angular_coef" default="14.64" />
  <arg name="encoder_ticks_per_rev" default="1600" />
  <arg name="base_width" default="0.39" />
  <arg name="wheel_diameter" default="0.125" />

  <node pkg="trd_diff_controller" type="trd_diff_controller_node" name="trd_diff_controller_node">
    <param name="serialport_name" value="$(arg serialport_name)"/>
    <param name="baudrate" value="$(arg baudrate)"/>
    <param name="left_coef" value="$(arg left_coef)" />
    <param name="right_coef" value="$(arg right_coef)" />
    <param name="linear_coef" value="$(arg linear_coef)" />
    <param name="angular_coef" value="$(arg angular_coef)" />
    <param name="encoder_ticks_per_rev" value="$(arg encoder_ticks_per_rev)" />
    <param name="base_width" value="$(arg base_width)" />
    <param name="wheel_diameter" value="$(arg wheel_diameter)" />
  </node>
</launch>
```

图 2.36: 直线运动校准参数示意图

2.3.6 旋转运动校准



```
<?xml version="1.0"?>
<launch>
  <!-- TR10 -->
  <arg name="serialport_name" default="/dev/motor_trd" />
  <arg name="baudrate" default="38400"/>
  <arg name="left_coef" default="-1.0" />
  <arg name="right_coef" default="1.0" />
  <arg name="linear_coef" default="82.0" />
  <arg name="angular_coef" default="14.64" /> highlighted
  <arg name="encoder_ticks_per_rev" default="1600" />
  <arg name="base_width" default="0.39" />
  <arg name="wheel_diameter" default="0.125" />

  <node pkg="trd_diff_controller" type="trd_diff_controller_node" name="trd_diff_controller_node">
    <param name="serialport_name" value="$(arg serialport_name)"/>
    <param name="baudrate" value="$(arg baudrate)"/>
    <param name="left_coef" value="$(arg left_coef)" />
    <param name="right_coef" value="$(arg right_coef)" />
    <param name="linear_coef" value="$(arg linear_coef)" />
    <param name="angular_coef" value="$(arg angular_coef)" />
    <param name="encoder_ticks_per_rev" value="$(arg encoder_ticks_per_rev)" />
    <param name="base_width" value="$(arg base_width)" />
    <param name="wheel_diameter" value="$(arg wheel_diameter)" />
  </node>
</launch>
```

图 2.37: 旋转运动校准参数示意图

校准完成后，**CTRL+C** 结束所有的程序，**CTRL+D** 关闭所有的终端。

3 功能开发

在上一部分，我们完成了对 Abel 移动机器人底层控制功能的开发，通过校准，Abel，能够比较精确的跟随速度指令。我们可以在此基础上进行高级控制功能的开发，使机器人能够逐渐地自主完成复杂的任务。

3.1 使用里程计实现机器人的运动

移动机器人 Abel 的底层控制功能使我们可以直接向 Base Controller 发送速度指令控制机器人运动，但这种遥控的方式需要人为参与机器人的运动过程，通常只是在测试与校准的时候使用。接下来会对控制方式做进一步的提升，通过 ROS 节点向机器人发送速度指令，机器人的运动过程不再有人为参与。

这一部分将会通过 ROS 节点的编写使 Abel 移动机器人依次完成一系列运动目标：向正前方运动 1 米，旋转 180 度，然后返回起始点，再旋转 180 度恢复初始位姿。首先实现依赖时间和速度的开环控制方式，然后加入里程计信息，实现依赖里程计的闭环控制方式。

以上功能完成之后，将会使机器人实现稍微复杂的动作，即从正方形的一个顶点位置出发，依次经过其余三个顶点，最后回到初始位置，恢复初始位姿。

3.1.1 以时间与速度为参考的运动

A 任务设置

通过代码编写使使机器人依次完成以下动作：

- 1 以 0.2m/s 的速度向正前方运动 5s（理论上向前运动 1m 的距离）
- 2 以 1rad/s 的角速度逆时针旋转 180 度
- 3 以 0.2m/s 的速度向后方运动 5s（理论上向后运动 1m 的距离）
- 4 以 1rad/s 的角速度逆时针旋转 180 度

B 功能实现

首先使用 RoboWare Studio 打开 Abel/Abel_controller/scripts 目录下的 timed_out_and_back.py 文件，根据提示，完成实现以上机器人动作的代码练习 (TODO 练习)。

代码编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果：

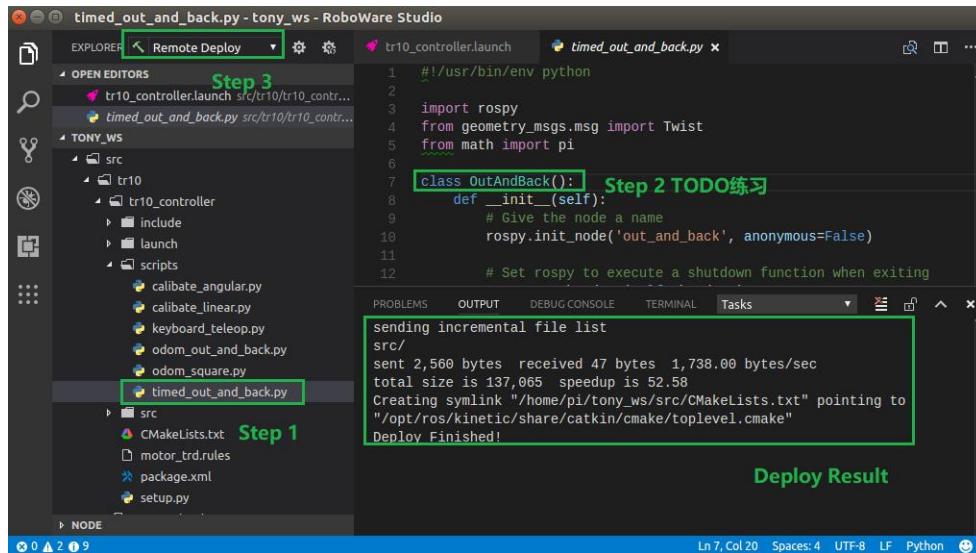


图 3.1: 以时间与速度为参考的运动程序构建及部署操作示意图

C 功能测试

代码编写并部署完成后在移动机器人 Abel 上进行验证，首先确保机器人周围有足够的运动空间，在其正前方预留大约 1.5m 的空间。

选择 Abel_controller/launch 目录下的 Abel_minimal.launch 文件，点击右键，在弹出的列表中选择“Run Remote Launch File”会运行 launch 文件，启动 Abel Base Controller 并加载 Abel 的模型参数：

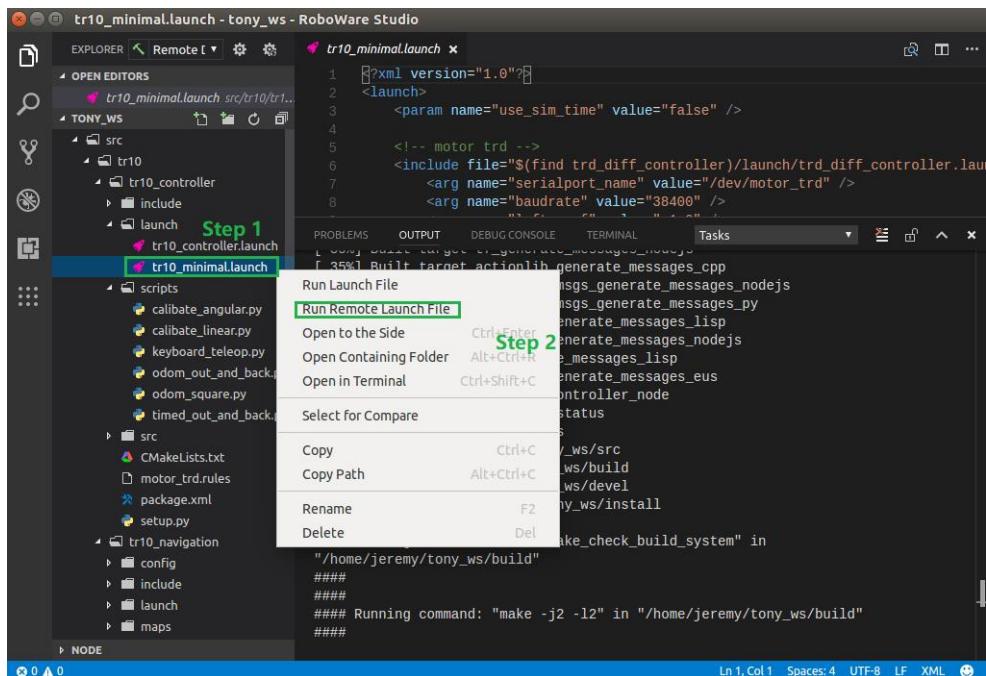


图 3.2: RoboWare Studio 远程启动 Abel_minimal.launch 文件操作示意图

成功启动 Abel_minimal.launch 文件后，将会出现如下图所示的画面：

```
<?xml version="1.0"?>
<launch>
    <param name="use_sim_time" value="false" />
    <!-- load the URDF/Xacro model of our robot -->
    <param name="robot_description" textfile="$(find tr10_controller)/urdf/tr10.urdf" />
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" args="" />
    <!-- motor trd -->
    <include file="$(find tr10_controller)/launch/tr10_controller.launch">
        <arg name="serialport_name" value="/dev/motor_trd" />
        <arg name="baudrate" value="38400" />
        <arg name="left_coeff" value="-1.0" />
        <arg name="right_coeff" value="1.0" />
    </include>
</launch>
```

PROBLEMS: * /trd_diff_controller_node/serialport_name: /dev/motor_trd
* /trd_diff_controller_node/wheel_diameter: 0.125
* /use_sim_time: False

NODES:

- / robot_state_publisher (robot_state_publisher/state_publisher)
- /trd_diff_controller_node (tr10_controller/trd_diff_controller_node)

ROS_MASTER_URI=http://192.168.10.108:11311

process[robot_state_publisher-1]: started with pid [13620]
process[trd_diff_controller_node-2]: started with pid [13621]

图 3.3: Abel_minimal.launch 文件成功启动示意图

在 PC 端 Ctrl+Alt+t 打开另一个新的终端，输入以下指令，启动 rviz，观察机器人的运动情况：

rosrun rviz rviz -d ‘rospack find Abel_controller’ /rviz/timed_out_and_back.rviz

启动 rviz 之后，会出现以下的窗口（注意实际运行时机器人的位置可能是不同的，但要勾选图中标出的里程计与机器人模型两个 RViz Display）：

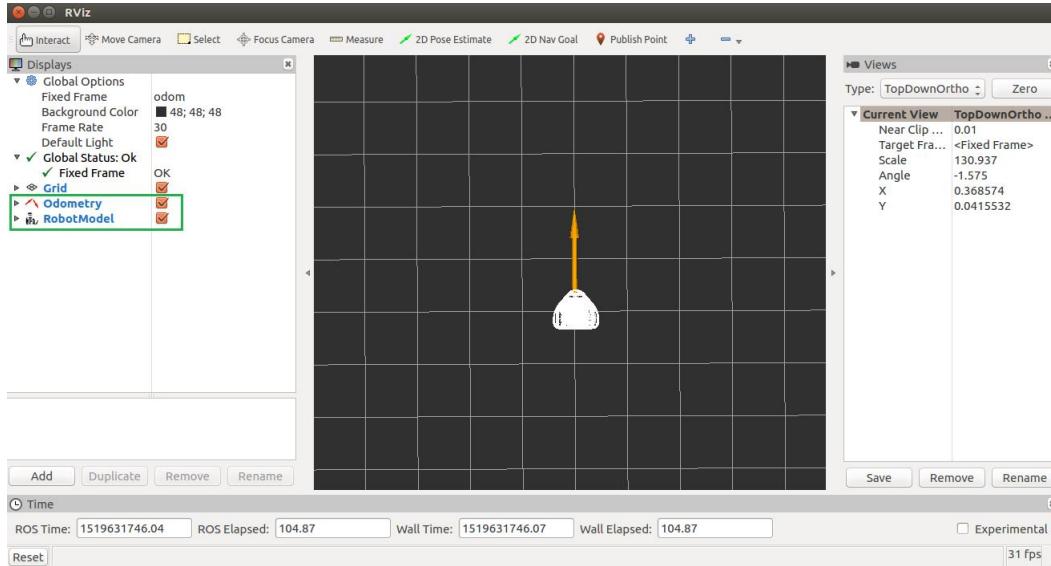


图 3.4: 通过 RViz 观察以时间与速度为参考的运动示意图

选择 Abel_controller/scripts 目录下的 timed_out_and_back.py 文件，点击右键，

在弹出的列表中选择“Run Python File in Terminal”会启动 `timed_out_and_back.py` 节点：

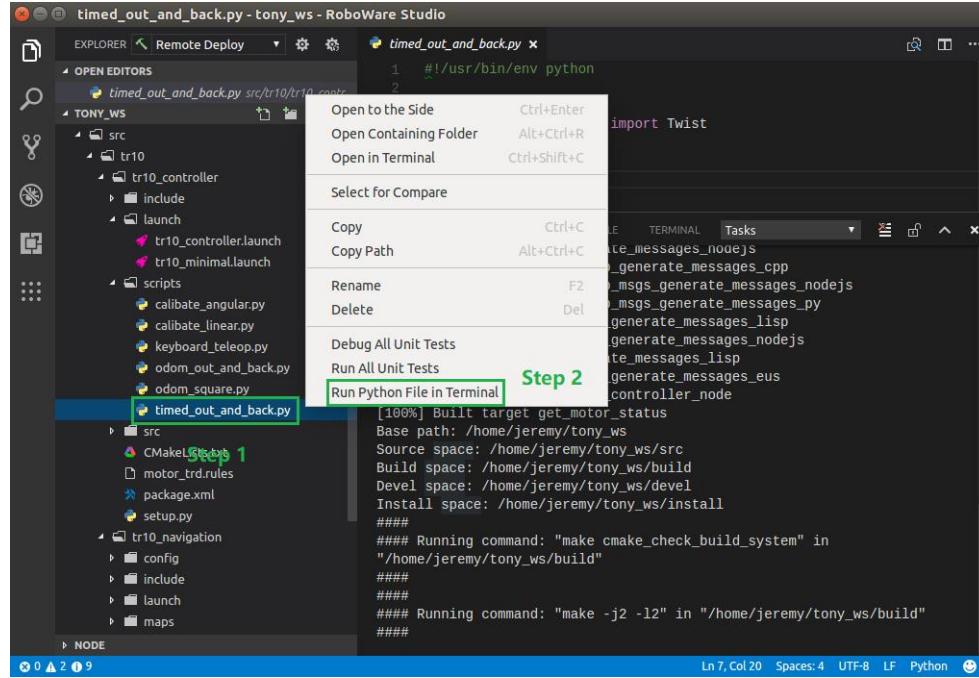


图 3.5: RoboWare Studio 启动 `timed_out_and_back.py` 脚本操作示意图

按照任务设置的要求，Abel 会依次完成以下动作：向正前方运动 1 米，旋转 180 度，然后返回起始点，再旋转 180 度恢复初始位姿。在 RViz 中会看到类似于下图的效果（实际运行效果会有所不同）：

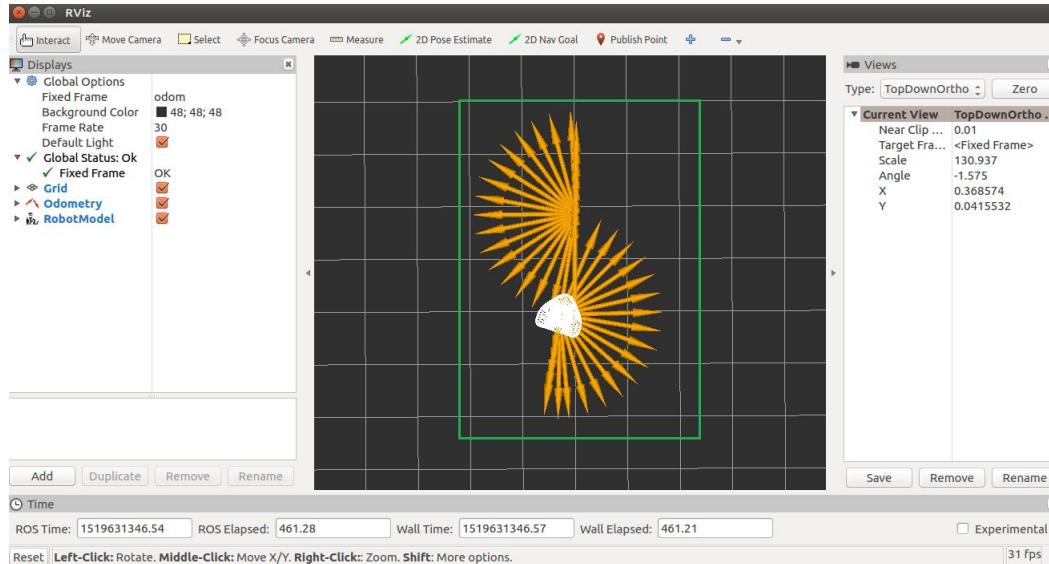


图 3.6: Abel 以时间与速度为参考的运动示意图

功能测试完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

D 小节思考

本节中我们构建的 `timed_out_and_back` 节点以时间与速度为参考向 Abel 发布运动命令，即已知 Abel 的运动速度与目标距离，计算出理论运行时间，然后在这个时间周期中持续发布/`cmd_vel` 话题（Abel 不会保持某一速度，要保持运行需要持续接收新的/`cmd_vel`），Abel Base Controller 会接收该话题并驱动电机运动。

但实际运行时，我们会发现 Abel 没有走出 1m 远，也没有旋转到 180 度，最终 Abel 不会恢复原来的位姿，这一点在 rviz 中也会表现出来，里程计信息会指示机器人的运动出现了较大的偏移。这是什么原因造成的呢？

小提示

这是因为这种控制方式属于开环控制，没有使用任何的反馈信息，我们向机器人发出了控制命令，但我们不会知道机器人是否真的完成了我们指定的运动。虽然在基础开发部分我们已经进行了控制器参数校准，但并不能消除所有误差因素，因此在实际运行时会发现机器人出现了比较大的偏移。

3.1.2 以里程计为参考的运动

在基础开发部分，我们已经提到过，里程计是一种利用从移动传感器获得的数据来估计物体位姿随时间的变化而改变的方法。以时间为参考的运动方式相当于开环运动，随着时间的累积会产生比较大的误差，而里程计数据的使用可以帮助我们实现机器人的闭环控制，能够在一定程度上减小误差，提升控制精度。

A 任务设置

通过代码的编写使机器人依次完成以下动作：

- 1 以 0.2m/s 的速度向正前方运动 5s（理论上向前运动 1m 的距离）
- 2 以 1rad/s 的角速度逆时针旋转 180 度
- 3 以 0.2m/s 的速度向后方运动 5s（理论上向后运动 1m 的距离）
- 4 以 1rad/s 的角速度逆时针旋转 180 度

B 功能实现

首先使用 RoboWare Studio 打开 Abel/Abel_controller/scripts 目录下的 `odom_out_and_back.py` 文件，根据提示，完成实现以上机器人动作的代码练习（TODO 练习）。

代码编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果：

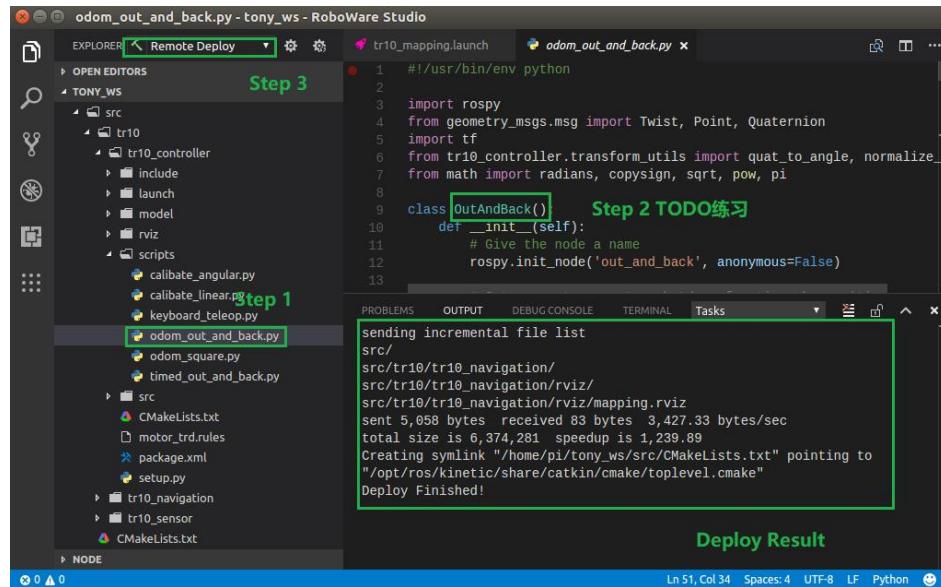


图 3.7: 以里程计参考的运动程序构建及部署操作示意图

C 功能测试

代码编写及部署完成后在移动机器人 Abel 上进行验证，首先确保机器人周围有足够的运动空间，在其正前方预留大约 1.5m。

选择 Abel_controller/launch 目录下的 Abel_minimal.launch 文件，点击右键，在弹出的列表中选择“Run Remote Launch File”会运行 launch 文件，启动 Abel Base Controller 并加载 Abel 模型参数：

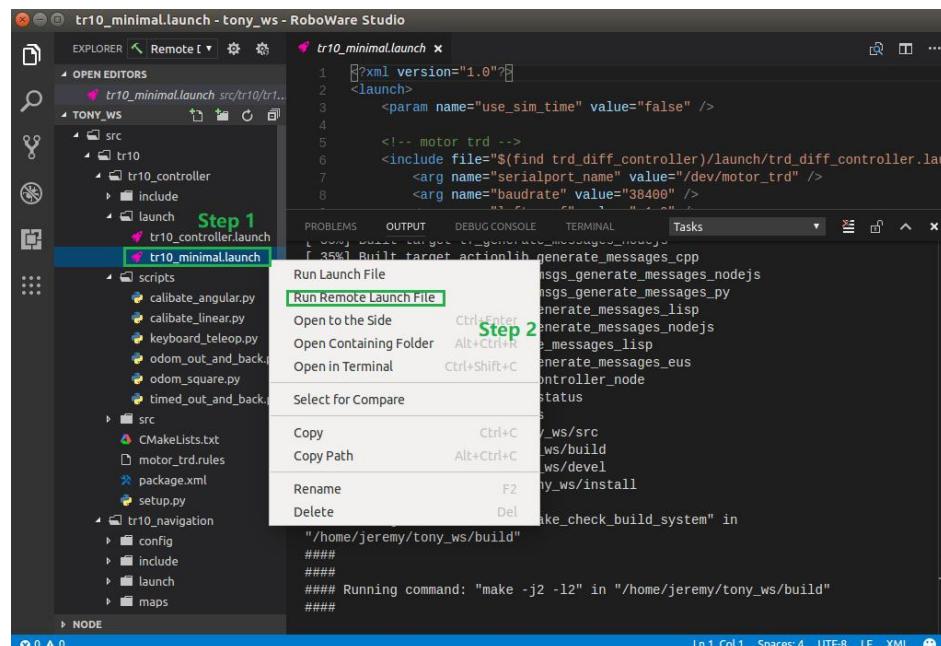


图 3.8: RoboWare Studio 远程启动 Abel_minimal.launch 文件操作示意图

成功启动 Abel_minimal.launch 文件后，将会出现如下图所示的画面：

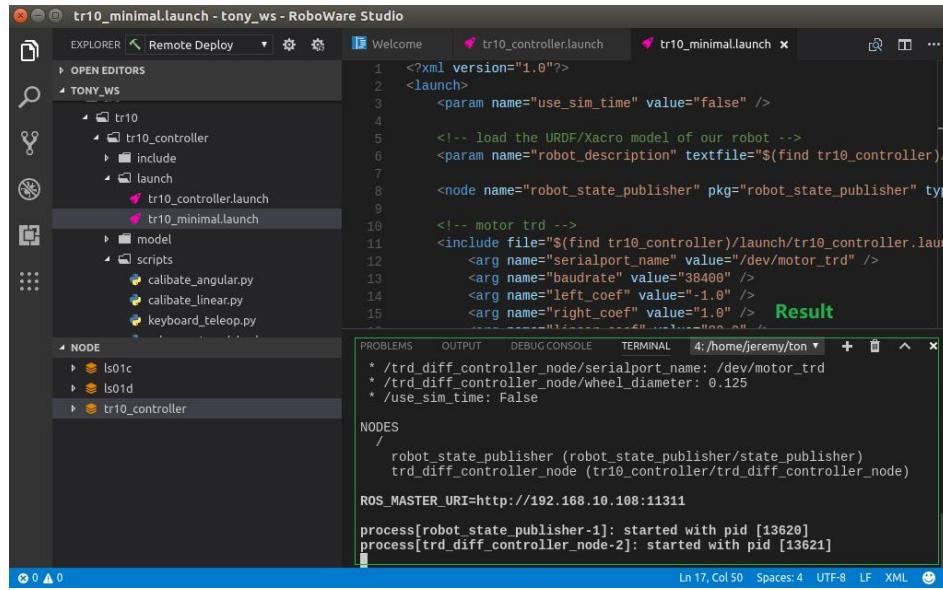


图 3.9: Abel_minimal.launch 文件成功启动示意图

在 PC 端 Ctrl+Alt+t 打开另一个新的终端，输入以下指令，启动 rviz，观察机器人的运动情况：

rosrun rviz rviz -d ‘rospack find Abel_controller’ /rviz/odom_out_and_back.rviz

启动 rviz 之后，会出现以下的窗口（注意实际运行时机器人的位置可能是不同的，但要勾选图中标出的里程计与机器人模型两个 RViz Display）：

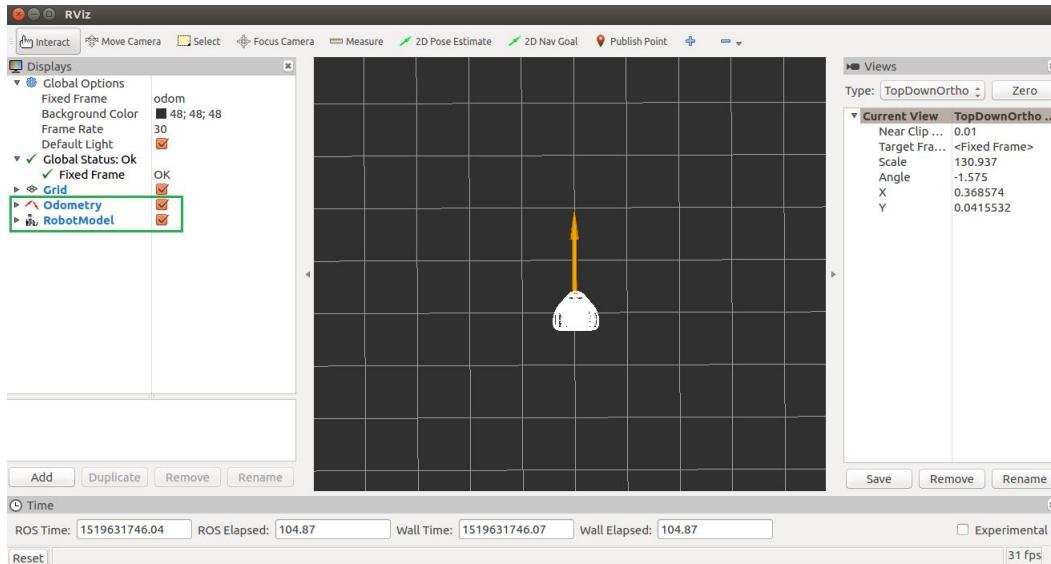


图 3.10: 通过 RViz 观察以里程计为参考的运动示意图

选择 Abel_controller/scripts 目录下的 odom_out_and_back.py 文件，点击右键，在弹出的列表中选择“Run Python File in Terminal”启动 odom_out_and_back.py 节

点：

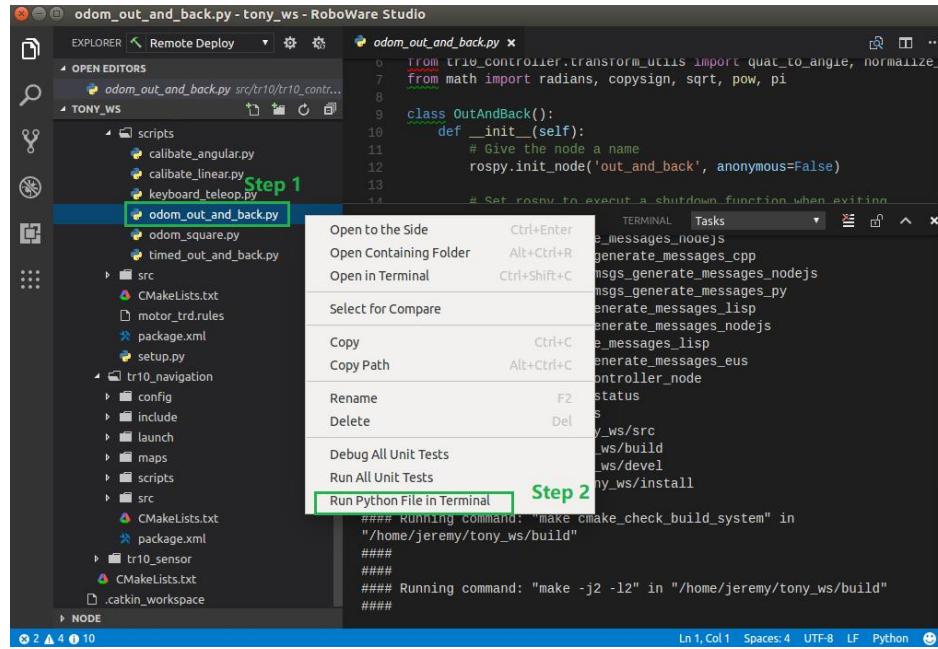


图 3.11: RoboWare Studio 启动 odom_out_and_back.py 脚本操作示意图

按照任务的设定，Abel 同样会依次完成以下动作：向正前方运动 1 米，旋转 180 度，然后返回起始点，再旋转 180 度恢复初始位姿。在 RViz 中会看到类似于下图的效果（实际运行效果会有所不同）：

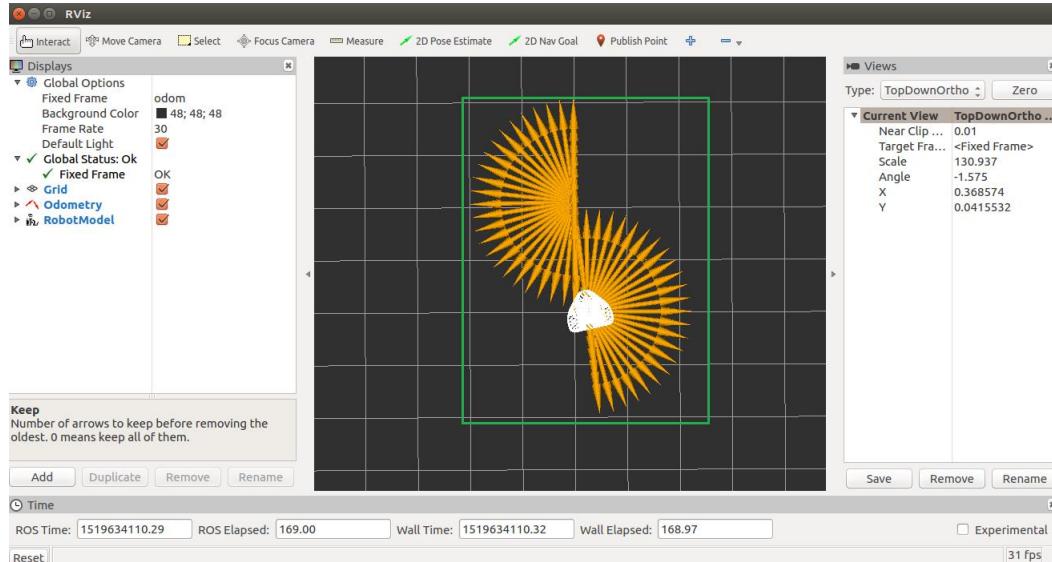


图 3.12: Abel 以里程计为参考的运动示意图

功能测试完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

D 小节思考

本节我们构建的 `odom_out_and_back` 节点在控制逻辑中加入了里程计信息作为反馈，实现了依赖里程计信息的闭环控制。节点会实时的从 `tf` 变换中获取机器人位置和角度信息，而任务设置中的四个小任务的运动目标都是已知的，这样可以通过当前位置和角度与目标位置和角度的差值判断是否已经到达目标点，如果未到达，则按照速度要求继续发布速度信息，如果到达目标位置，则停止发送。

对比以时间与速度为参考的控制方式会发现机器人的运动准确度获得了一定程度的提升，但运动过程仍然出现了比较大的偏移，没有准确的实现任务设置的要求，这是什么原因造成的呢？

小提示

虽然我们在这一节中实现了采用里程计信息的闭环控制方式，但我们依然不能排除掉所有的误差因素。Abel 移动机器人接收编码器返回的信息构建里程计，而编码器自身就存在误差，我们无法消除，因此虽然实现了闭环控制，但如果仅仅依赖测量机器人自身信息的传感器很难达到令人满意的控制效果。

3.1.3 以里程计为参考的多路径点运动

A 任务设置

接下来完成一个稍微复杂一些以里程计为参考的多路径点运动，机器人从正方形一个顶点出发，向前运动 1m，到达第二个顶点，逆时针旋转 90 度，再向前运动 1m 到达第三个顶点，依次类推，最后回到第一个顶点，且恢复初始方向。

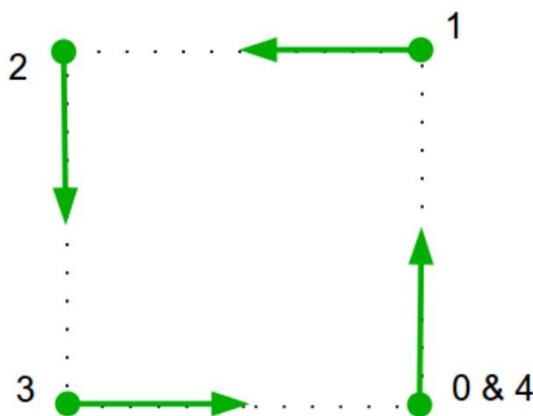


图 3.13: 以里程计为参考的多路径点运动任务示意图

B 功能实现

首先使用 RoboWare Studio 打开 Abel/Abel_controller/scripts 目录下的 odom_square.py 文件，根据提示，完成实现以上机器人动作的代码练习（TODO 练习）。

代码编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果：

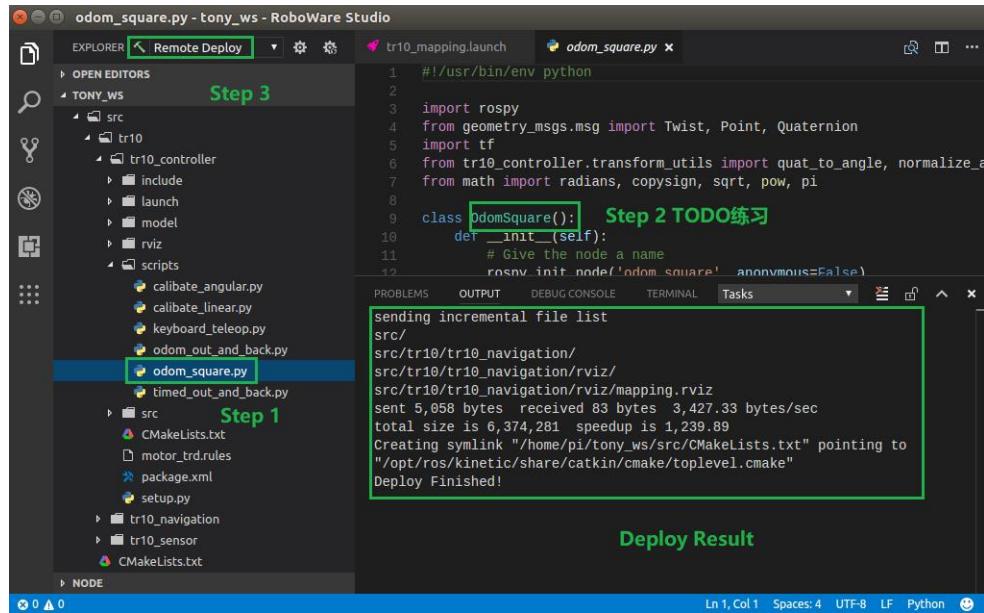


图 3.14: 以里程计为参考的多路径点运动程序构建及部署操作示意图

C 功能测试

代码编写及部署完成后在移动机器人 Abel 上进行验证，首先确保机器人周围有足够的运动空间，预留边长约为 1.5m 的正方形区域。

如图 3.15 所示，选择 Abel_controller/launch 目录下的 Abel_minimal.launch 文件，点击右键，在弹出的列表中选择“Run Remote Launch File”会运行 launch 文件，启动 Abel Base Controller 并加载 Abel 模型参数：

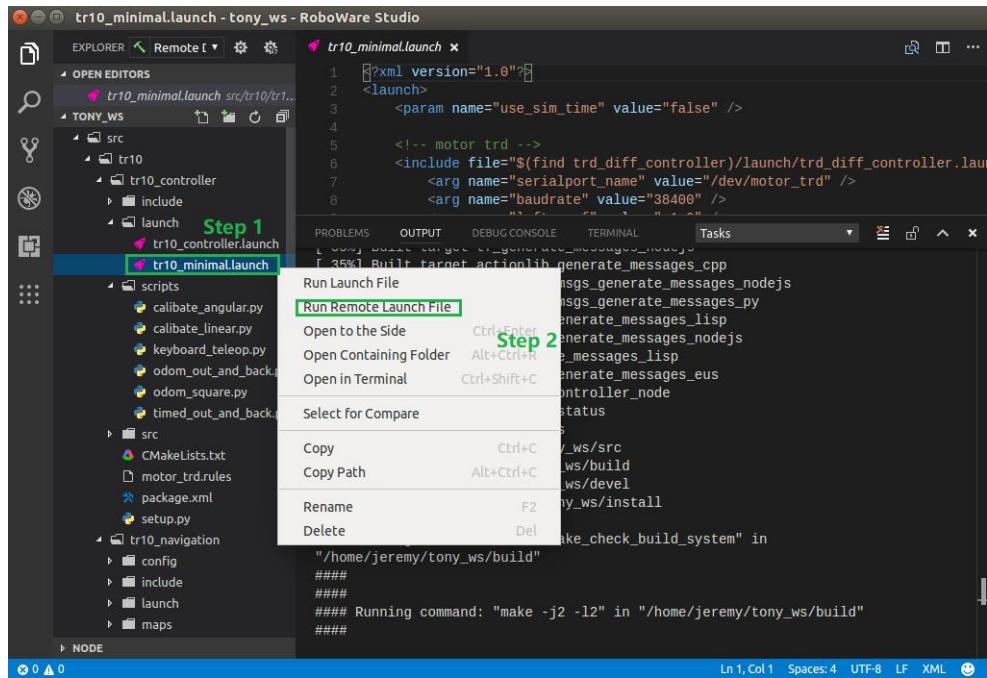


图 3.15: RoboWare Studio 远程启动 Abel_minimal.launch 文件操作示意图

成功启动 Abel_minimal.launch 文件后，将会出现如图 3.16 所示的画面：

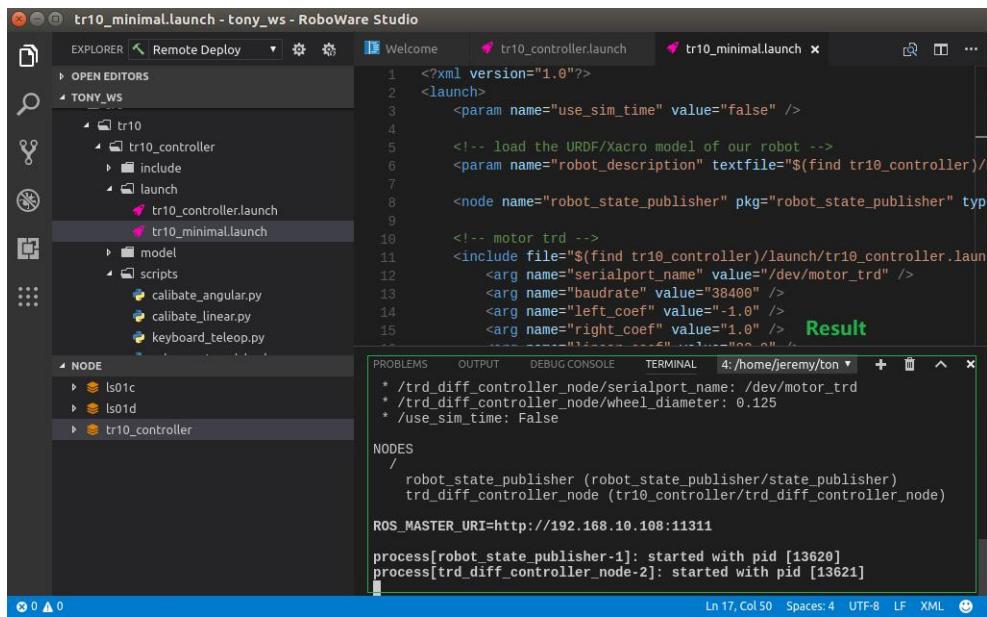


图 3.16: Abel_minimal.launch 文件成功启动示意图

在 PC 端 $\text{Ctrl}+\text{Alt}+\text{t}$ 打开另一个新的终端，输入以下指令，启动 rviz，观察机器人的运动情况：

```
rosrun rviz rviz -d 'rospack find Abel_controller' /rviz/odom_square.rviz
```

启动 rviz 之后，会出现以下的窗口（注意实际运行时机器人的位置可能是不同的，但要勾选图中标出的里程计与机器人模型两个 RViz Display）：

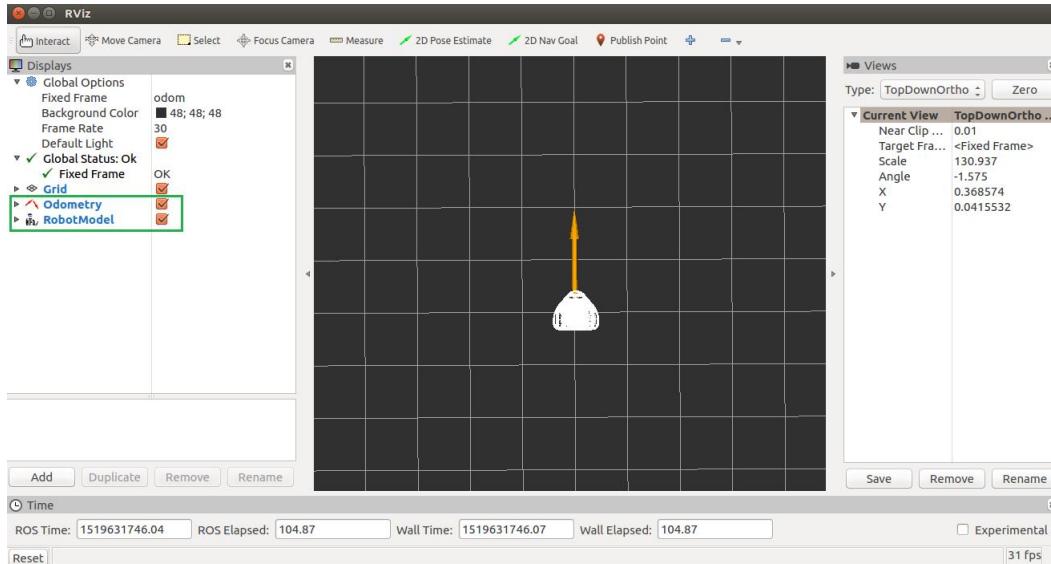


图 3.17: 通过 RViz 观察以里程计为参考的多路径点运动示意图

选择 Abel_controller/scripts 目录下的 odom_square.py 文件，点击右键，在弹出的列表中选择“Run Python File in Terminal”会启动 odom_square.py 节点：

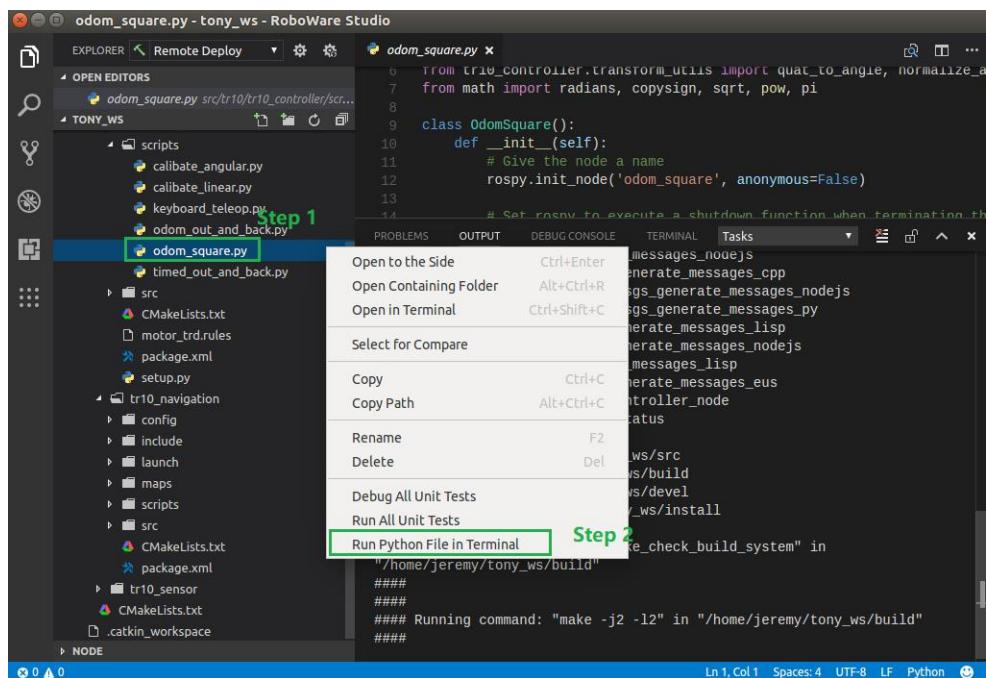


图 3.18: RoboWare Studio 启动 timed_square.py 脚本操作示意图

按照节点的设定，Abel 移动机器人会走完一个正方形，正方形的每个顶点都是一个路径点，机器人从一个顶点出发，依次经过其它 3 个顶点之后回到起始点。在 RViz 中会

看到类似于下图的效果（实际运行效果会有所不同）：

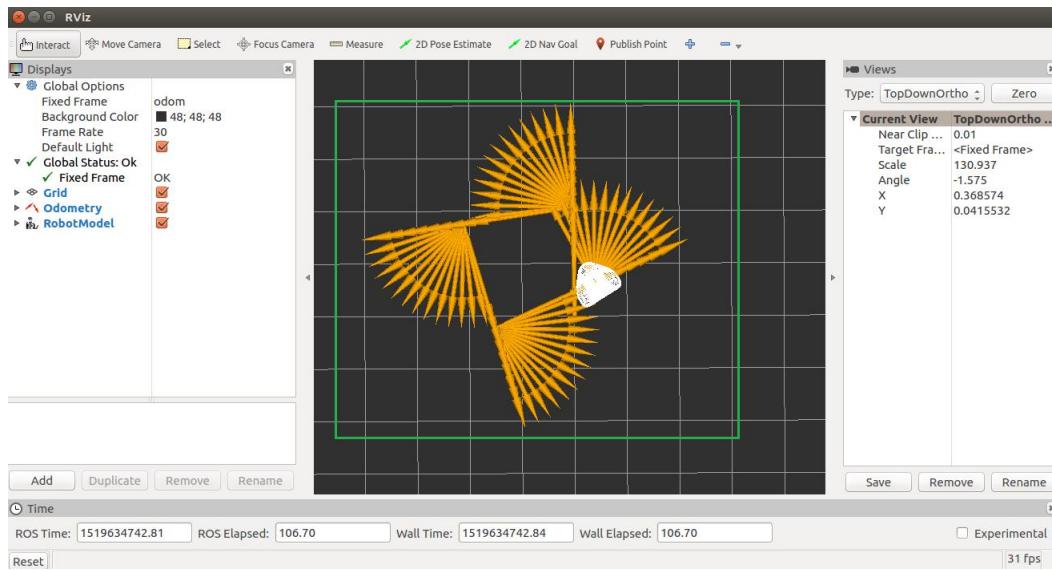


图 3.19: Abel 以里程计为参考的多路径点运动示意图

功能测试完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

D 小节思考

本节我们构建的 `odom_square` 节点同 `odom_out_and_back` 节点一样，在控制逻辑中加入了里程计信息作为反馈，实现了依赖里程计信息的闭环控制，不同之处是目标点的设定，将正方形的四个顶点作为子目标点。

实际运行时，会发现机器人能够大概走出正方形，但不会产生标准的正方形路径，而且看起来产生的偏移会越来越大，这是什么原因造成的？有没有进一步提升的方法呢？

小提示

这是因为当前使用的依赖里程计信息的闭环控制，只依赖机器人内部的数据，不参考任何的外部标志物。一般来说，如果机器人依赖内部的传感器数据运行比较长的时间，都会产生比较大的累积误差。而如果能够在机器人的运动中结合路标或者其它的外部参考物，那么会使精度得到比较大的提升。

3.2 实现移动机器人的地图构建功能

在上一部分中，机器人使用航位推算法来完成运动，这种仅仅依赖机器人内部数据的方式随着时间的累积会产生比较大的误差，因此需要结合路标或其它外部参考物以提供更多的信息帮助机器人完成运动任务。

一种方法是为机器人提供运行场景的地图，而 SLAM 方法可以帮助我们做到这一点。SLAM 是同步定位与地图构建（Simultaneous Localization and Mapping）的缩写，可以帮助机器人在未知的环境中进行地图构建同时进行定位。

当前主要通过激光雷达或 RGB-D 相机来实现 SLAM，其中 RGB-D 相机可通过 ROS 中的 depthimage_to_laserscan 包构建虚拟的激光信息。

另外，ROS 提供了 gmapping 包实现 SLAM 方法，其中包含 slam_gmapping 节点，可通过接收到的激光扫描数据与机器人里程计数据构建 2-D 可占用栅格地图（2-D Occupancy Grid Map）。

在这一部分，我们将会使用激光雷达与 Gmapping 包实现机器人的地图构建功能。

3.2.1 激光雷达驱动安装及测试

为了使用 gmapping package 中的 slam_gmapping 节点完成地图构建任务，需要使用激光雷达，以提供周围环境的信息。激光雷达通过 USB 接口接入 Abel 主控板卡。

在 2.2.2 节中，已经将 Abel_sensor 包部署到了 tony_ws 中，这个包中包含了激光雷达 ls01d 的驱动，编译成功后，驱动成功进行了安装，接下来是进行测试。

选择 Abel_sensor/launch 目录下的 ls01d.launch 文件，点击右键，在弹出的列表中选择“Run Remote Launch File”会运行 launch 文件，启动 ls01d 激光雷达驱动。

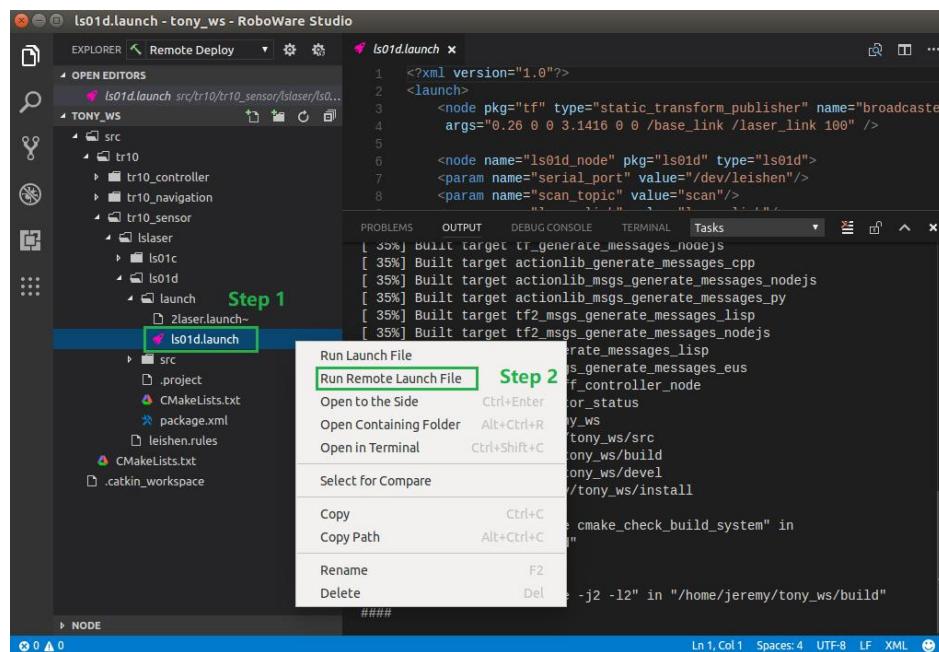


图 3.20: RoboWare Studio 远程启动 ls01d.launch 文件操作示意图

成功启动 Abel_minimal.launch 文件后，将会出现如下图所示的画面：

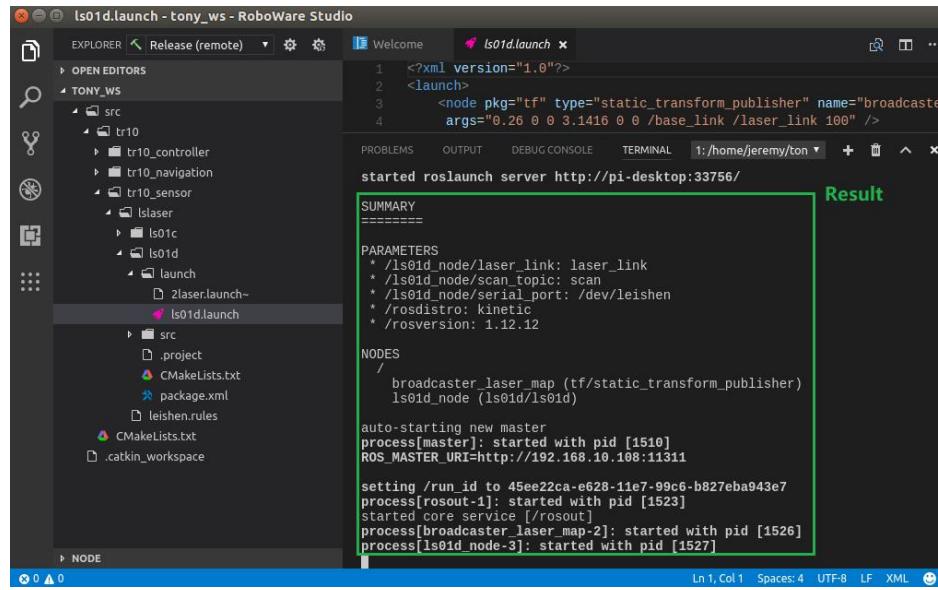


图 3.21: ls01d.launch 文件成功启动示意图

在 PC 端 Ctrl+Alt+t 打开另一个新的终端，输入以下指令，启动 rviz，查看激光雷达所扫描的周围环境信息：

rosrun rviz rviz -d ‘rospack find Abel_sensor’ /rviz/laser_ls01d.rviz

启动 rviz 之后，会出现以下的窗口（注意勾选 Laser Scan RViz Display）：

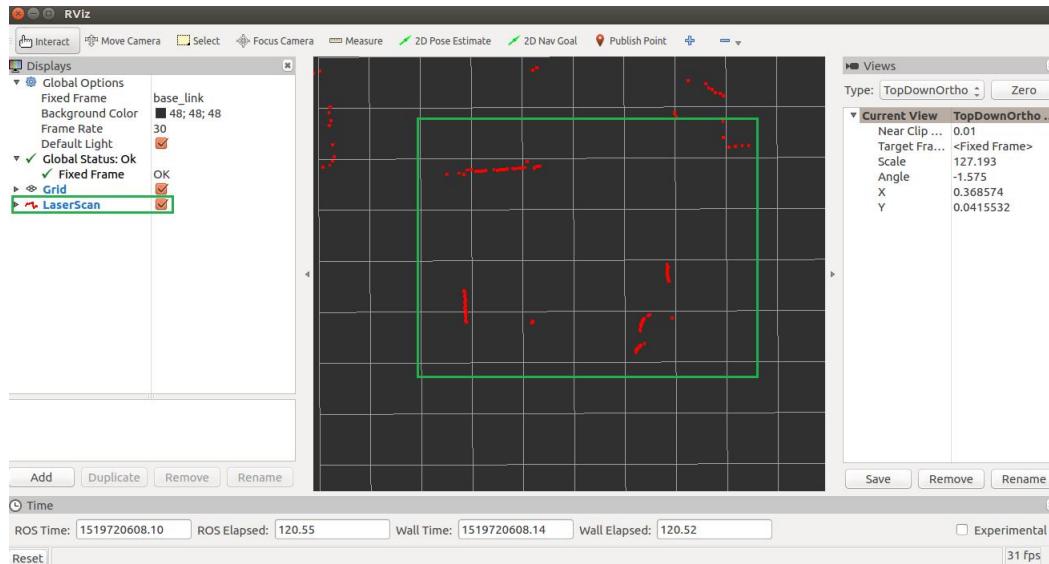


图 3.22: 激光雷达测试示意图

功能测试完成之后，CTRL+C 关闭所有程序，CTRL+D 关闭所有终端。

3.2.2 环境地图构建

激光雷达驱动安装完成之后便可以使用 gmapping package 中的 slam_gmapping 节点对机器人运行环境的地图进行构建。

在 2.2.2 节，已经将 Abel_navigation 包部署到 tony_ws 中，这个包里面包含了机器人建图与自主导航相关的内容。

首先是构建使用 slam_gmapping 建图的 launch 文件，该文件已经存放于 Abel_navigation 目录下的 launch 文件夹中，文件命名为 Abel_mapping.launch。需要确保通过该 launch 文件启动 Abel Base Controller，激光雷达驱动以及 slam_gmapping 节点（TODO 练习）。Launch 文件编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果（在 2.2.4 节已经展示过）。

远程部署完成之后，在该 launch 文件上点击右键，在弹出的窗口上选择“Run Remote Launch File”选项，按照要求，该 launch 文件会启动 Abel Base Controller，加载机器人模型参数，启动激光雷达驱动以及 slam_gmapping 节点。

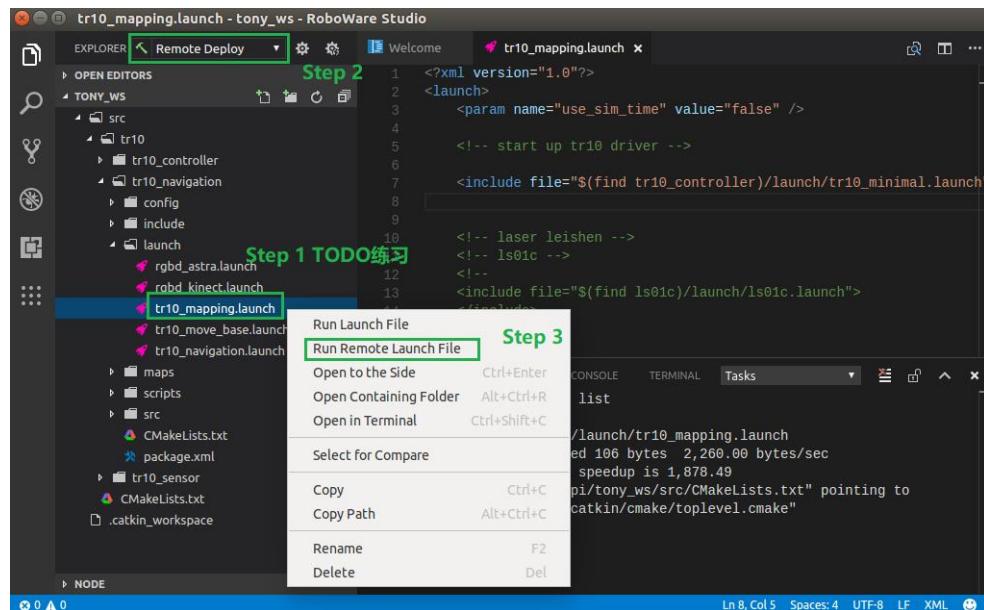


图 3.23: RoboWare Studio 远程启动 Abel_mapping.launch 文件操作示意图

启动成功之后，会出现如下的画面：

The screenshot shows the RoboWare Studio interface with the 'tr10_mapping.launch - tony_ws' project open. In the terminal window, the output of the launch file is displayed, showing the start of various ROS nodes:

```

process[master]: started with pid [6103]
ROS_MASTER_URI=http://192.168.10.108:11311
setting /run_id to e7ee8a54-e62f-11e7-99c6-b827eba943e7
process[rosout-1]: started with pid [6116]
started core service [/rosout]
process[robot_state_publisher-2]: started with pid [6134]
process[trd_diff_controller_node-3]: started with pid [6135]
process[broadcaster_laser_map-4]: started with pid [6136]
process[lso1d_node-5]: started with pid [6142]
process[slam_gmapping-6]: started with pid [6156]
Laser Pose= 0.26 0 7.43383e-06

```

图 3.24: Abel_mapping.launch 文件成功启动示意图

然后选择 Abel_controller/scripts 目录下的 keyboard_teleop.py 脚本，点击右键，在弹出的列表中选择“Run Python File in Terminal”会在集成终端中运行该 Python 脚本，启动键盘控制节点：

The screenshot shows the RoboWare Studio interface with the 'keyboard_teleop.py - tony_ws' project open. A context menu is open over the 'keyboard_teleop.py' file in the editor, with the 'Run Python File in Terminal' option highlighted. Step 1 indicates the right-click action, and Step 2 indicates the selected terminal command.

图 3.25: RoboWare Studio 启动键盘控制节点操作示意图

键盘控制节点打开后，终端内会出现相应的操作说明，按照操作说明可以通过 PC 端

的键盘远程操控机器人运动。需要注意的是，在运动之前需确保机器人周围有足够的空间，并且开始运动时使用低速运行。

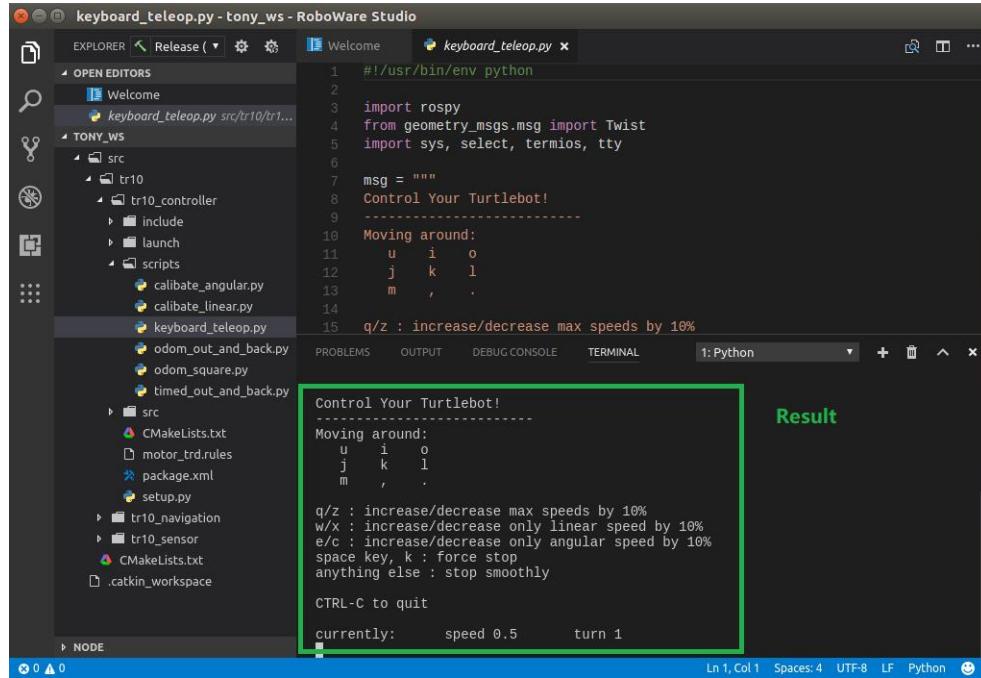


图 3.26: 键盘控制节点启动成功示意图

在 PC 端 $\text{Ctrl}+\text{Alt}+\text{t}$ 打开另一个新的终端，输入以下指令，启动 rviz，实时查看地图构建的进度：

```
rosrun rviz rviz -d 'rospack find Abel_navigation' /rviz/mapping.rviz
```

启动 rviz 之后，会出现以下的窗口，使用键盘控制机器人在实验场地中慢慢移动，Abel 便会实时进行地图构建并将结果显示在 rviz 中：

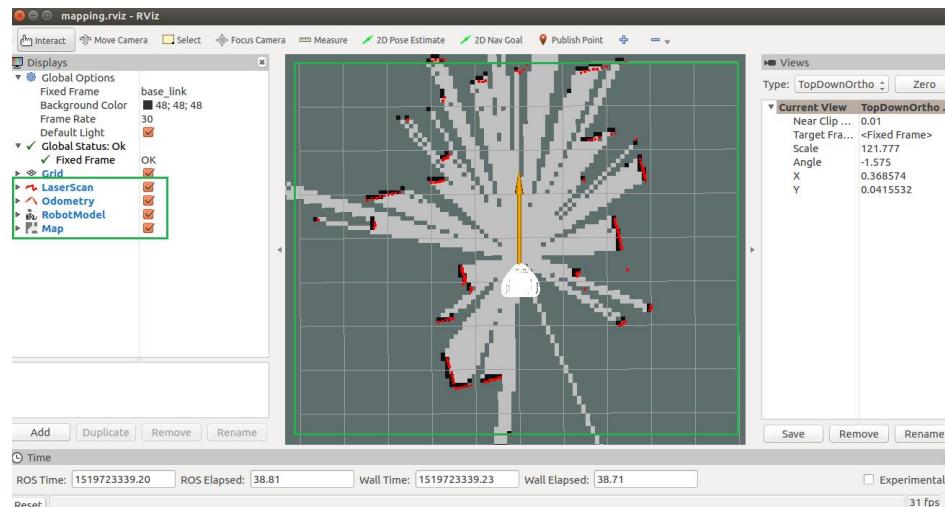


图 3.27: Abel 使用 slam_gmapping 建图功能示意图

下图是运行一段时间后构建的地图：

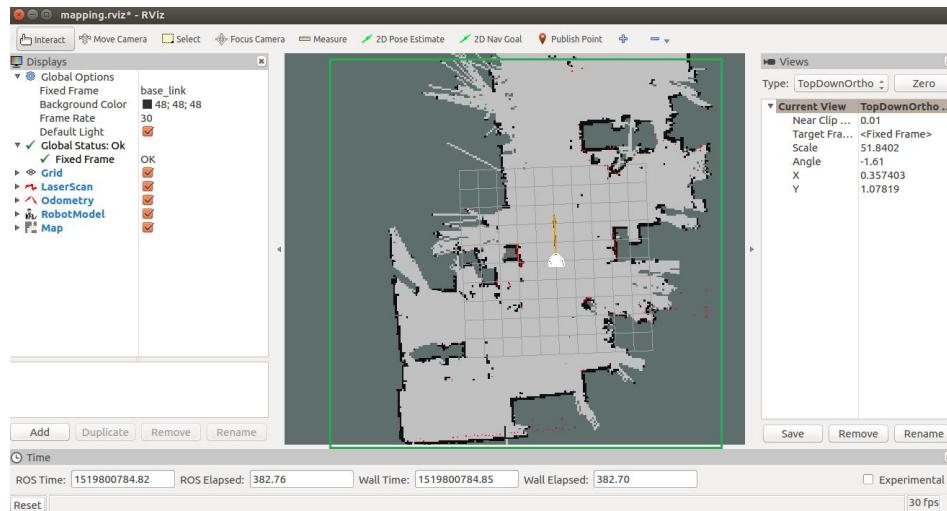


图 3.28: Abel 建图功能运行一段时间后构建的地图示意图

控制机器人运行一段时间，构建出比较满意的地图之后，可以将地图进行保存。首先在 PC 端打开新的终端，输入以下指令切换到 maps 文件夹：

```
roscd Abel_navigation/maps
```

然后输入以下指令将地图进行保存：

```
rosrun map_server map_saver -f my_map
```

以上指令会将 slam_gmapping 构建的地图保存于 Abel_navigation package 的 maps 文件夹中，第二条指令中的“-f”用于地图命名，这里用的名字是“my_map”，可根据实际情况为场景地图命名。指令执行成功后，会出现下图所示的信息：

```
INFO] [1519802243.228956106]: Waiting for the map
INFO] [1519802244.805644361]: Received a 384 X 608 map @ 0.050 m/pix
INFO] [1519802244.805753275]: Writing map occupancy data to my_map.pgm
INFO] [1519802244.866717040]: Writing map occupancy data to my_map.yaml
INFO] [1519802244.868490321]: Done
```

图 3.29: 地图存储成功显示信息示意图

地图保存完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

可以进入 maps 文件夹查看构建的地图，会发现两个文件：my_map.pgm 是地图图像，my_map.yaml 则是地图的描述文件，包含大小，分辨率等参数。

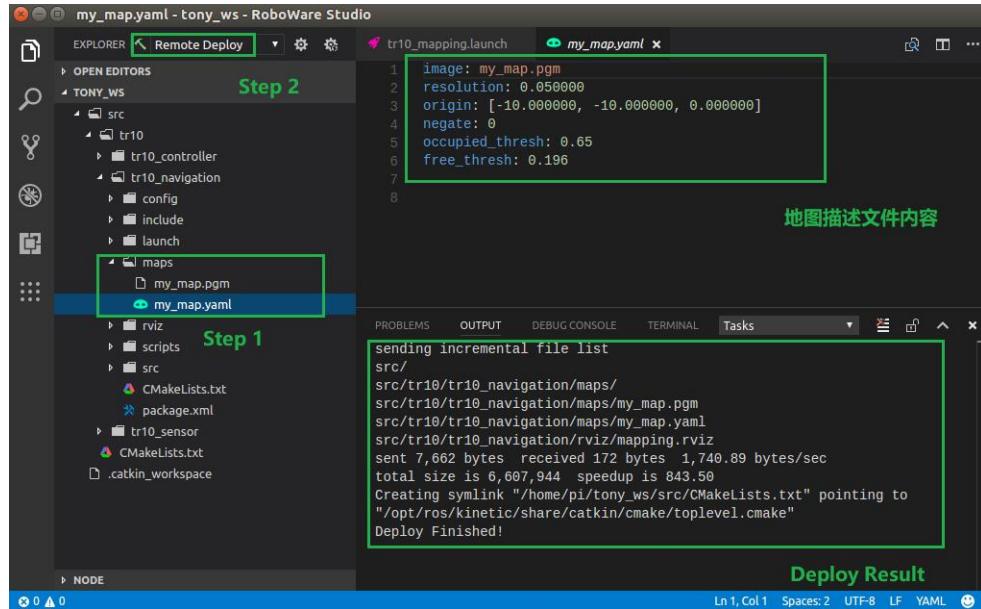


图 3.30: RoboWare Studio 远程部署地图文件及查看地图描述文件示意图

这一节的功能开发实现了移动机器人 Abel 的建图功能，并建立好了周围场景的地图，该地图可用于后续的自主导航功能，地图构建的质量越符合实际场景，自主导航功能的效果就越好，因此可通过多次构建以获取高质量的地图。

3.3 实现移动机器人的全自主导航功能

在上一部分中，实现了移动机器人 Abel 的地图构建功能，构建好的地图可用于实现移动机器人的自主导航功能，为规划器提供周围环境的信息。

ROS 的 Navigation 包中用于实现自主导航的框架是 move_base，它包含 global planner, local planner, global costmap, local costmap 及 recovery behaviors 等模块。已知机器人的初始位姿及目标位姿，global planner 会结合 global costmap 规划出全局路径发送到 local planner，local planner 会结合里程计数据，激光扫描数据及 local costmap 规划出最优的速度指令，通过/cmd_vel 话题进行发布。Base controller 会接收/cmd_vel 话题以驱动机器人运动。MoveBase 的控制框图为：

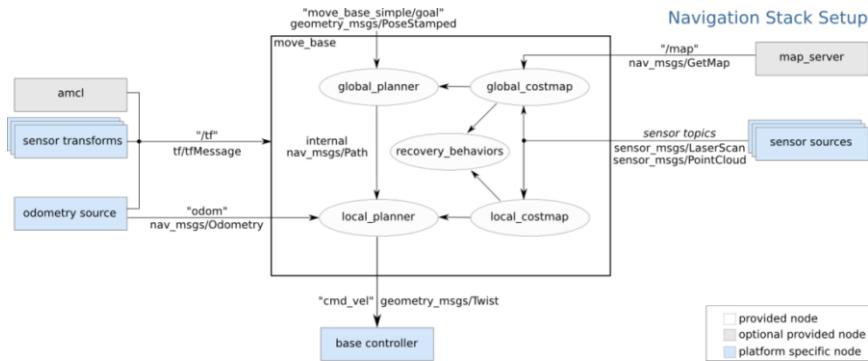


图 3.31: MoveBase 控制框架示意图

MoveBase 在运行时需要使用 4 个配置文件，这些文件包含其各个模块相关的参数，如机器人几何信息，速度限制，局部规划器的规划时间步长等。四个文件分别为：

```
base_local_planner_params.yaml
costmap_common_params.yaml
global_costmap_params.yaml
local_costmap_params.yaml
```

以上 yaml 格式的文件分别对应 local planner，global costmap 与 local costmap 等模块，这些参数决定了 MoveBase 每个模块的行为，具体的参数含义可以参考 ROS 官方 Wiki。

3.3.1 使用 AMCL 实现移动机器人的定位功能

AMCL 是 MoveBase 提供的移动机器人定位单元，实现了自适应蒙特卡洛定位（adaptive Monte Carlo Localization）算法，接下来我们构建 ROS 文件加入 amcl 定位单元，实现移动机器人的自主定位功能。

首先是构建 launch 文件，该文件已经存放于 Abel_navigation 包目录下的 launch 文件夹中，命名为 Abel_navigation.launch。应该确保该文件可以启动 Abel Base Controller，加载 Abel 模型参数，启动激光雷达驱动，启动 map_server 加载场景地图，启动 AMCL 节点，启动 MoveBase 节点并加载相应的配置参数（TODO 练习）。

Launch 文件编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果：

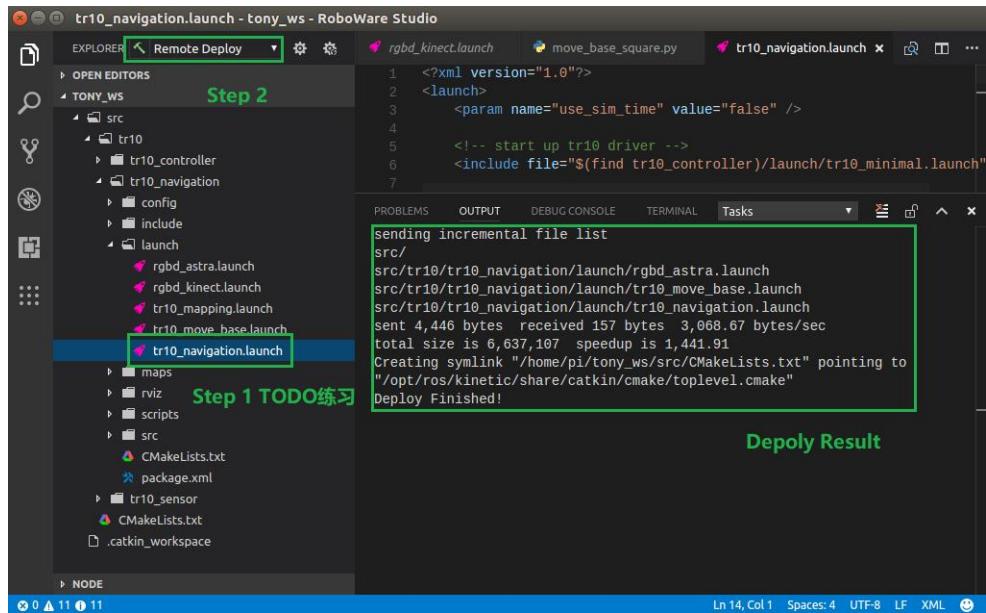


图 3.32: RoboWare Studio 对 Abel_navigation.launch 文件构建及部署操作示意图

远程部署完成之后，在该 launch 文件上点击右键，在弹出的窗口上选择“Run Remote Launch File”选项，启动 launch 文件：

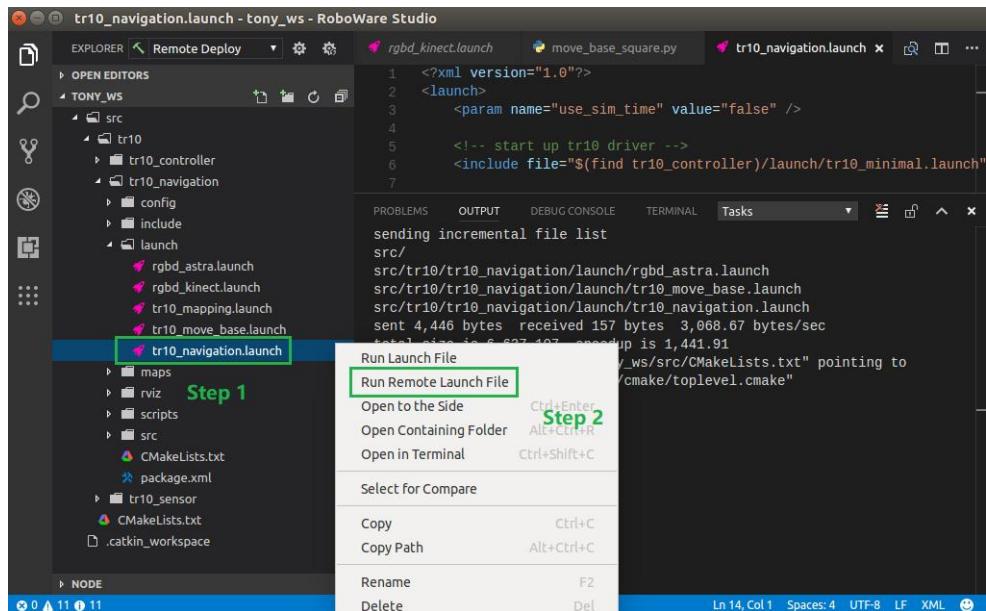


图 3.33: RoboWare Studio 远程启动 Abel_navigation.launch 文件操作示意图

按照要求，该 launch 文件会启动 Abel Base Controller，启动激光雷达驱动，启动 map_server 加载场景地图，启动 AMCL 节点，启动 MoveBase 节点并加载相应的配置参数。

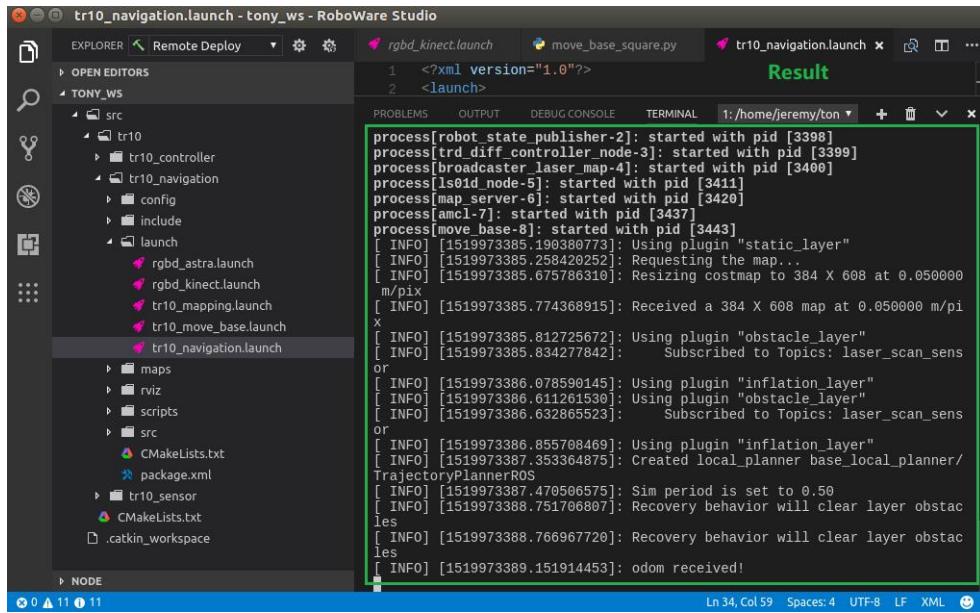


图 3.34: Abel_navigation.launch 文件远程启动成功示意图

在 PC 端 Ctrl+Alt+t 打开另一个新的终端，输入以下指令，启动 rviz，实时查看机器人的运动情况：

```
rosrun rviz rviz -d 'rospack find Abel_navigation' /rviz/move_base_amcl.rviz
```

启动 rviz 之后，可以选择加载 Pose Array 信息，另外可以选择加入里程计，全局规划路径，局部规划路径等信息，会出现类似于下图所示的窗口：

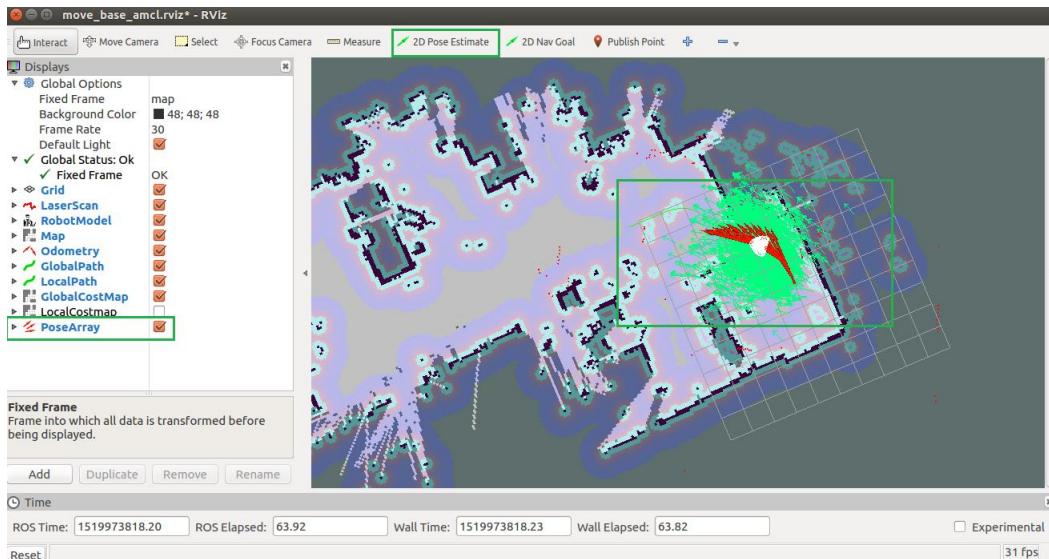


图 3.35: Abel 利用 AMCL 进行定位意图

Amcl 并不会判断机器人的初始位置，因此程序启动之后需要为机器人指定初始位置。在 Rviz 中，点击 2D Pose Estimate，然后在地图上正确的位置点击确认机器人的初

始位置，操作之后在机器人周围会出现一片箭头云，概率化的表示 amcl 估算的机器人位置，开始的时候箭头会比较分散，随着机器人的运动箭头将逐渐收敛。

在建好的地图中实现定位功能之后，移动机器人 Abel 具备了实现自主导航的基础，接下来我们会用多种方式向机器人发送导航目标，实现导航功能。

功能测试完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

3.3.2 使用 MoveBase 实现移动机器人的自主导航

ROS 中使用 MoveBase 框架实现移动机器人的自主导航。启动相应的程序后，可以在 rviz 显示的地图上通过 2D Nav Goal 设置导航目标点，机器人会自动规划出有效的路径，避开障碍物，local_planner 会规划出有效的速度，发布/cmd_vel 话题，controller 会接收速度命令，进而控制机器人运动。

3.3.2.1 通过 RViz 发布导航目标

首先是构建 launch 文件，该文件已经存放于 Abel_navigation 包目录下的 launch 文件夹中，命名为 Abel_move_base.launch。应该确保该文件可以启动 Abel Base Controller，启动 map_server 加载场景地图，启动 AMCL 节点，启动 MoveBase 节点并加载相应的配置参数（TODO 练习）。

Launch 文件编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果：

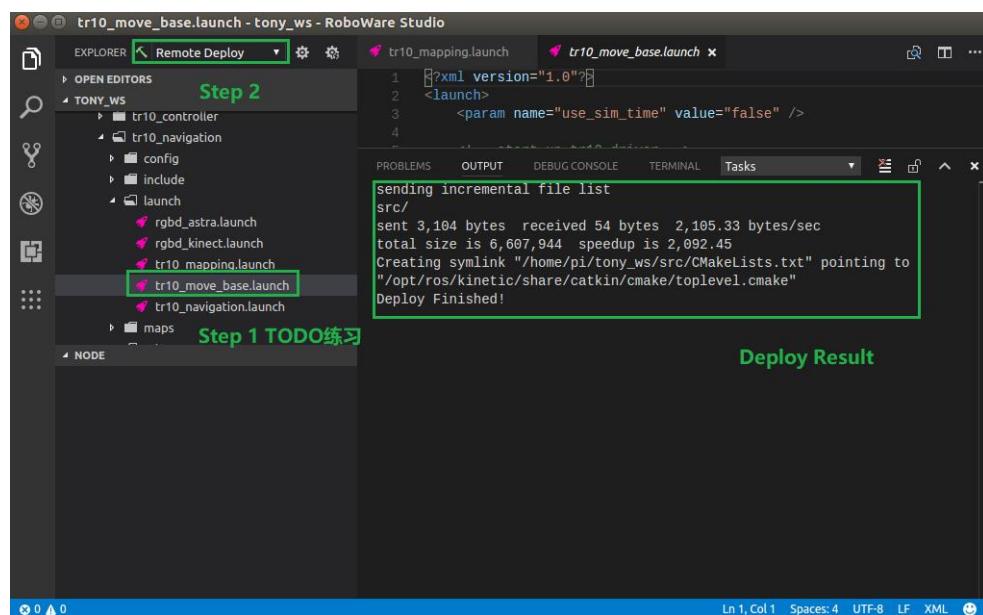


图 3.36: Abel_move_base.launch 文件构建及部署操作示意图

远程部署完成之后，在该 launch 文件上点击右键，在弹出的窗口上选择“Run Remote Launch File”选项，启动 launch 文件。

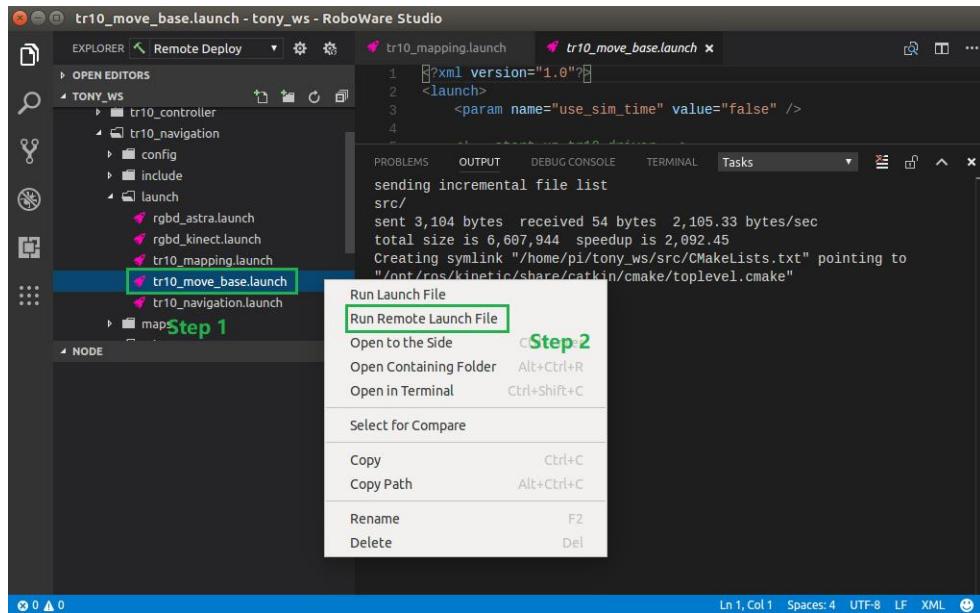


图 3.37: RoboWare Studio 远程启动 Abel_move_base.launch 文件启动操作示意图

按照要求，该 launch 文件会启动 Abel Base Controller，启动 map_server 加载场景地图，启动 MoveBase 节点并加载相应的配置参数（注意这里并没有启动激光雷达驱动，因此不具备动态避障功能）。

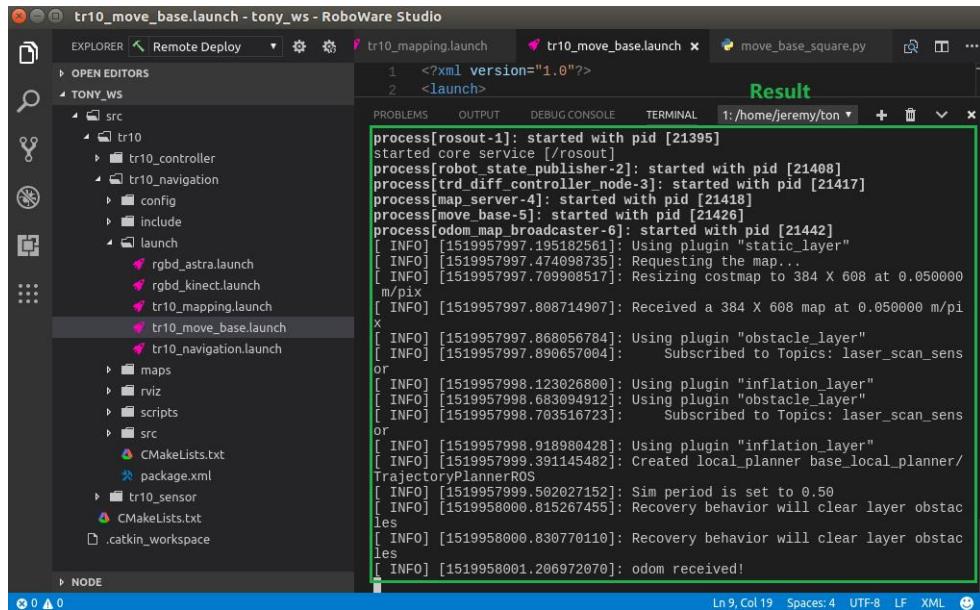


图 3.38: Abel_move_base.launch 文件启动成功示意图

在 PC 端打开新的终端，输入以下指令，启动 rviz，可以实时查看机器人的运动情况：

```
rosrun rviz rviz -d 'rospack find Abel_navigation' /rviz/move_base_rviz_cmd.rviz
```

启动 rviz 之后，会出现类似于下图所示的窗口：

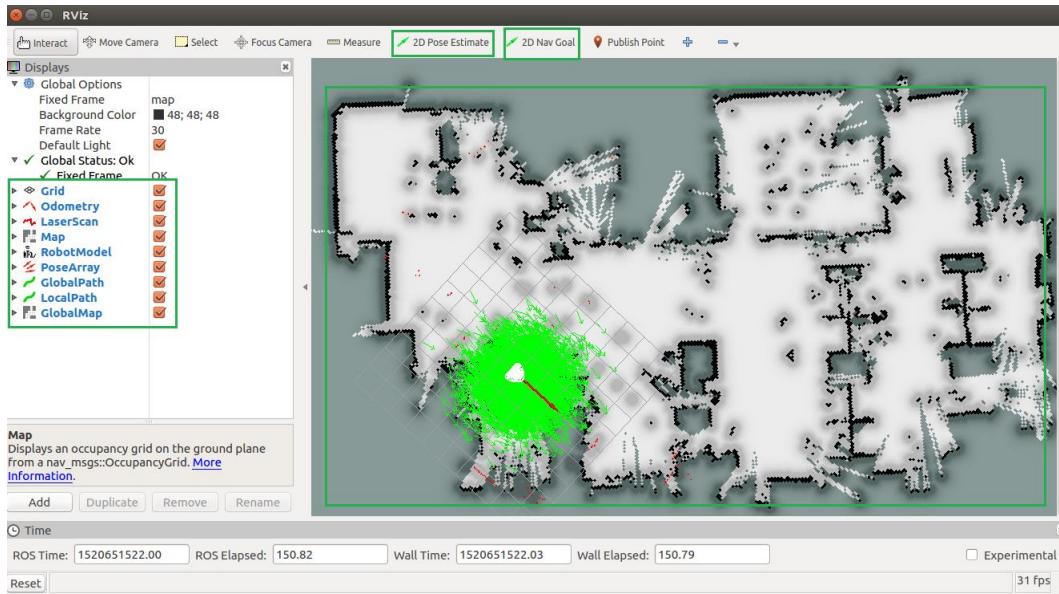


图 3.39: RViz 中查看 Abel 使用 MoveBase 进行导航的信息示意图

在 rviz 中可以查看里程计，全局规划路径，局部规划路径、Costmap 等信息。首先点击 2D Pose Estimate 在地图上选择 Abel 的实际位置，定位完成之后点击 Rviz 工具栏中的 2D Nav Goal 在地图上为机器人设置目标，设置完成后，机器人会自动规划出有效的路径并移动到目标位置，但由于为使用激光雷达，因此不能实现动态避障，只能避开地图信息中的静态障碍物。

通过 rviz 发布命令自主完成自主导航任务之后，**CTRL+C** 关闭当前运行的程序及 Rviz，**CTRL+D** 关闭所有终端。

3.3.2.2 通过 ROS 节点发布导航目标

刚才我们使用 Rviz 中的 2D Nav Goal 为机器人设置目标，接下来将使用 Python 构建的 ROS 节点为机器人设置导航目标，让机器人走完一个正方形，正方形的每个顶点都是一个路径点，机器人从一个顶点出发，依次经过其它 3 个顶点之后回到起始点，恢复初始位姿。Python 节点应该将正方形的顶点设定为 Abel 的导航目标。

在上一个环节中，我们已经构建好了 `Abel_move_base.launch` 文件用于启动使用 MoveBase 进行导航时所需的节点并加载配置参数。接下来是进行 Python 节点代码的编写，使用 RoboWare Studio 打开 `Abel/Abel_navigation/scripts` 目录下的 `move_base_square.py` 文件，根据提示，完成实现以上机器人动作的代码练习（TODO 练习）。

代码编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果（在 2.2.4 节已经展示过）。

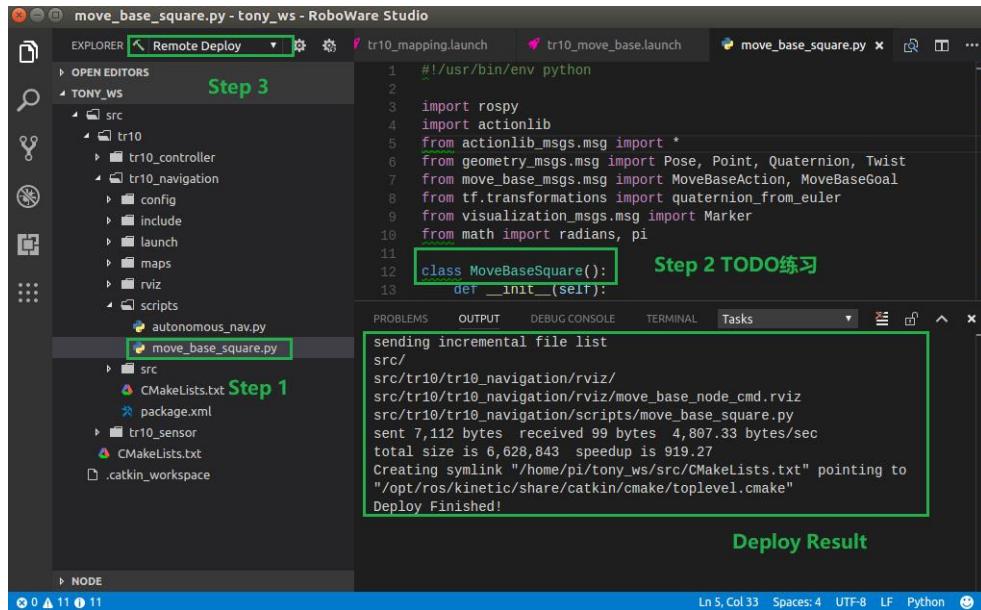


图 3.40: 通过 ROS 节点发布导航目标程序构建及部署操作示意图

节点代码编写并部署完成后，进行功能测试，首先将机器人重新放置到合适的位置，预留边长约为 1.5m 的正方形区域。

在 Abel_navigation/launch 文件夹下找到上一小节中构建的 Abel_move_base.launch 文件，在该 launch 文件上点击右键，在弹出的窗口上选择“Run Remote Launch File”选项，启动 launch 文件：

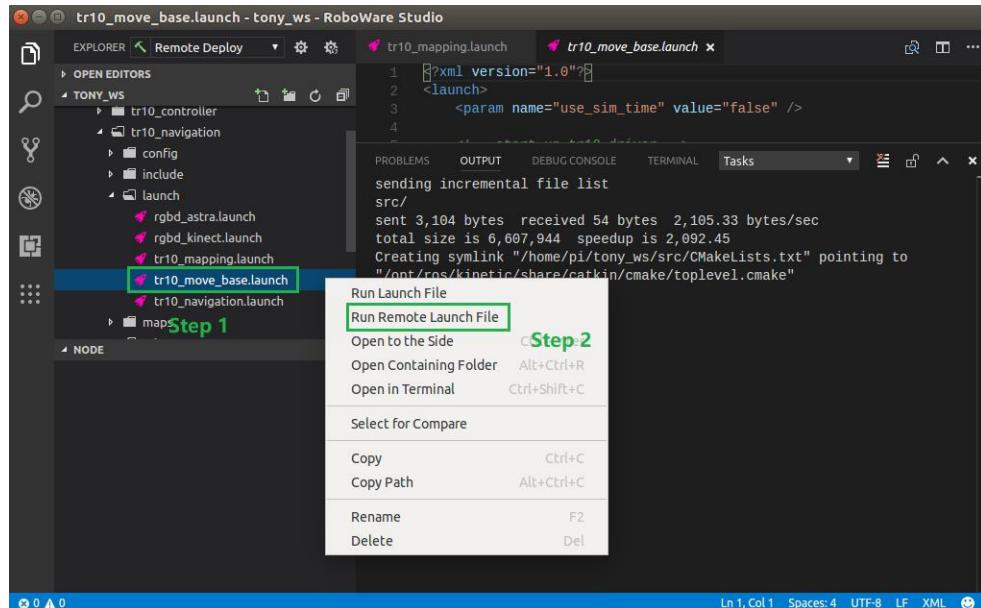


图 3.41: RoboWare Studio 远程启动 Abel_move_base.launch 文件操作示意图

按照要求，该 launch 文件会启动 Abel Base Controller，启动 map_server 加载场景

地图，启动 MoveBase 节点并加载相应的配置参数（注意这里并没有启动激光雷达驱动，因此不具备动态避障功能）。

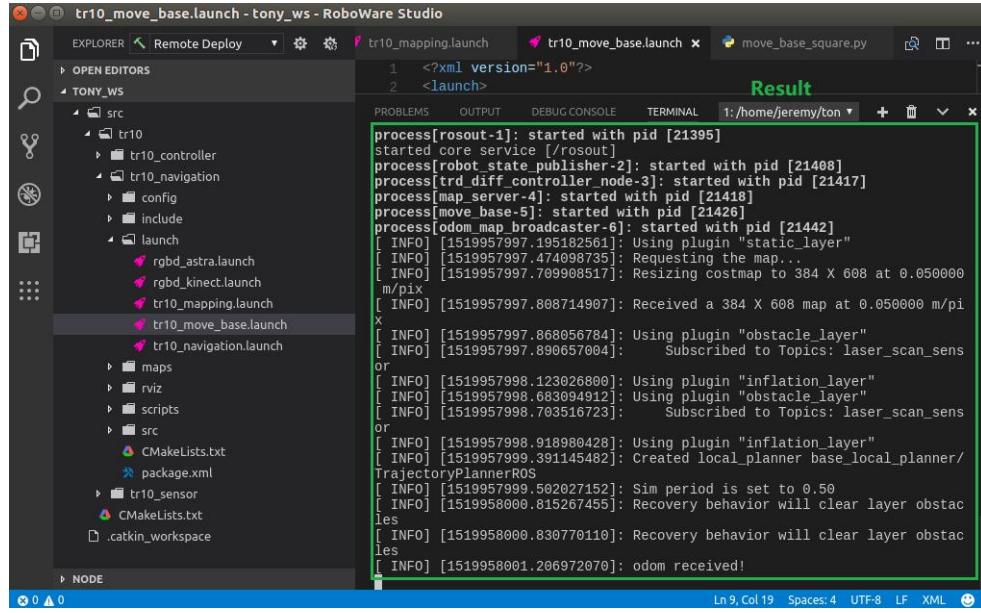


图 3.42: Abel_move_base.launch 文件远程启动成功示意图

在 PC 端打开新的终端，输入以下指令，启动 rviz，实时查看机器人的运动情况：

```
rosrun rviz rviz -d 'rospack find Abel_navigation' /rviz/-move_base_node_cmd.rviz
```

启动 rviz 之后，会出现类似于下图所示的窗口，点击 2D Pose Estimate 将 Abel 定位到正确的位置：

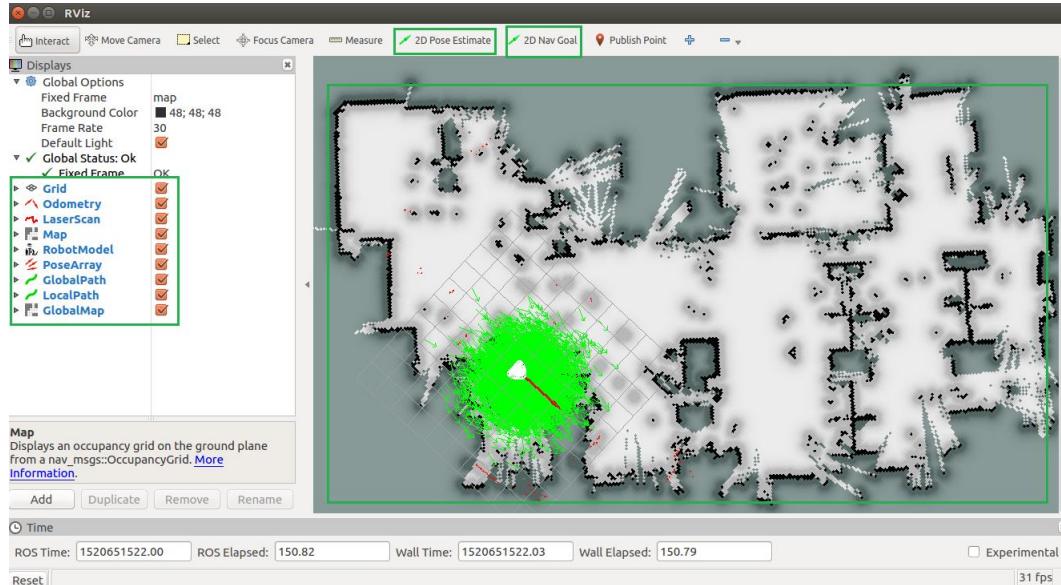


图 3.43: RViz 中查看 Abel 使用 MoveBase 进行导航的信息示意图

在 Abel_navigation/scripts 目录下选择我们刚刚构建的 move_base_square.py 文件，点击右键，在弹出的列表中选择“Run Python File in Terminal”会启动 move_base_square.py 节点：

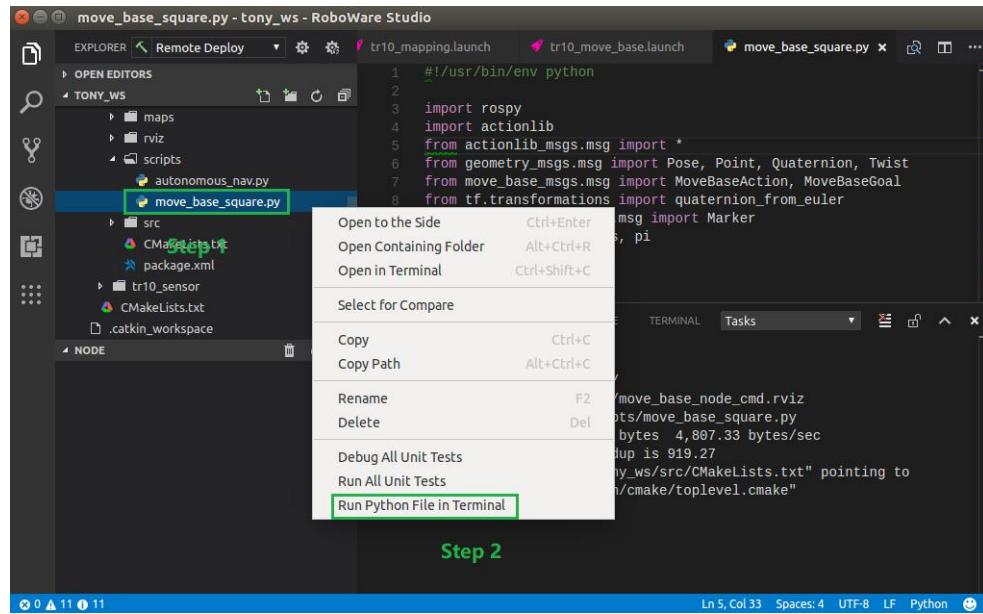


图 3.44: RoboWare Studio 远程启动 move_base_square.py 脚本操作示意图

该节点会向机器人发送命令，控制机器人依次走过正方形的四个顶点。在 RViz 中会看到类似于下图的效果（使用的地图不同，实际运行效果会有所不同）：

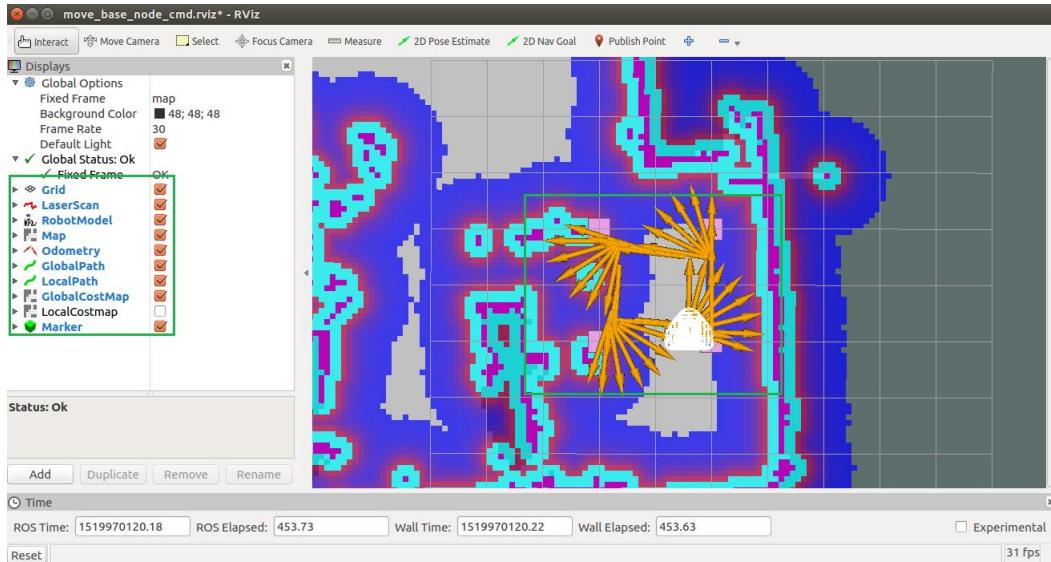


图 3.45: Abel 利用 MoveBase 进行正方形行走示意图

行驶过程中机器人会避开加载的场景地图中的静态障碍物，但由于没有使用激光雷达获取外部的数据，因此并不能实现动态避障。

当然也可以改变机器人的路径点，使 Abel 可以走过更复杂的路径，这些都可以作为这一小节的扩展练习。我们需要了解的是，MoveBase 最主要的作用不是进行精确的路径跟随，而是准确的到达导航目标位置同时避开障碍物。

另外，相比于使用里程计的闭环控制方法部分正方形行走节点 `odom_square.py`，程序的编写更加简单，不必直接监测里程计，`move_base` 会帮助我们完成相关的工作。

3.3.2.3 实现机器人的动态避障功能

除了帮助移动机器人规划出有效的路径到达目标以外，MoveBase 还可以帮助机器人实现避障，障碍物包含地图中的静态障碍物，也包含像突然出现的行人这样的动态障碍物。但由于前面的功能开发中没有启用激光雷达驱动，无法准确获取外部的参考物信息，因此不能实现动态避障，接下来我们会启动激光雷达驱动。

首先，修改 `Abel_move_base.launch` 文件，以支持启动激光雷达驱动。到 `Abel_navigation` 包目录下的 `launch` 文件夹中，找到 `Abel_move_base.launch` 文件根据提示完成修改。应该确保该文件在原有功能之外能够启动激光雷达的驱动（TODO 练习）。

Launch 文件修改完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果。

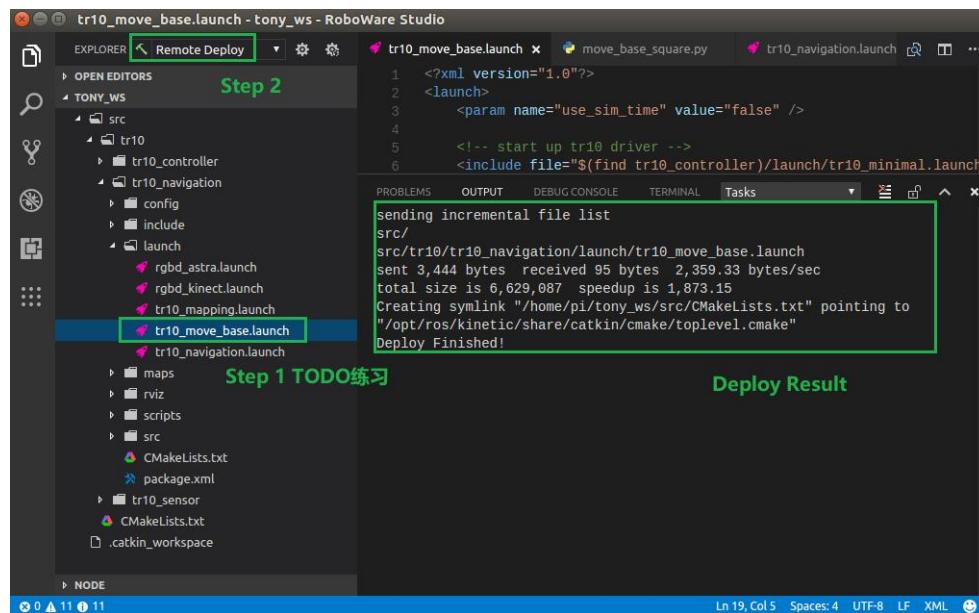


图 3.46: RoboWare Studio 对 `Abel_move_base.launch` 文件修改及部署操作示意图

远程部署完成之后，在该 `launch` 文件上点击右键，在弹出的窗口上选择“Run Remote Launch File”选项，启动 `launch` 文件：

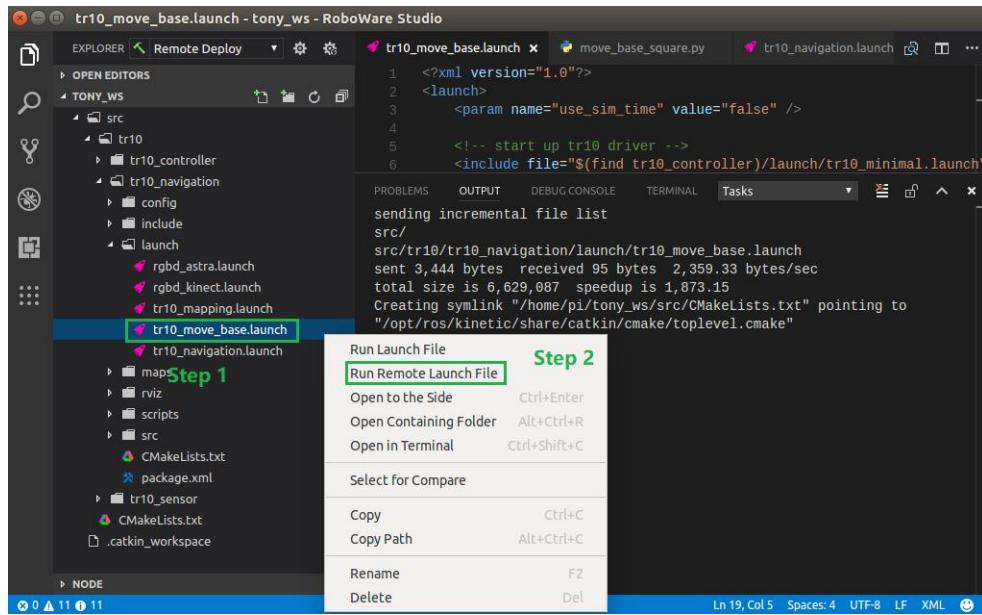


图 3.47: RoboWare Studio 远程启动 Abel_move_base.launch 文件操作示意图

按照要求，该 launch 文件会启动 Abel Base Controller，激光雷达驱动，启动 map_server 加载场景地图，启动 MoveBase 节点并加载相应的配置参数。

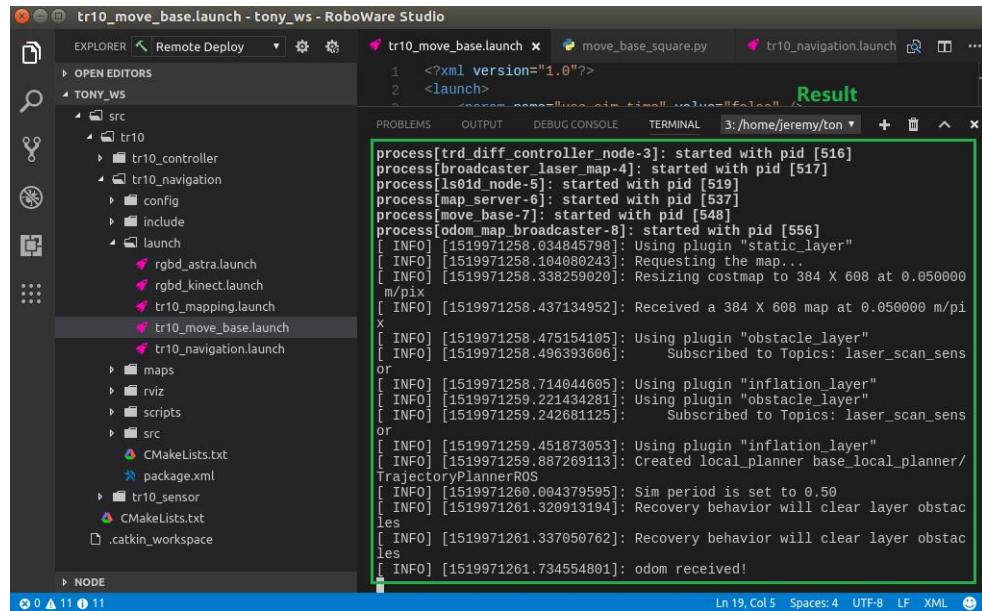


图 3.48: Abel_move_base.launch 文件远程启动成功示意图

在 PC 端打开新的终端，输入以下指令，启动 rviz，实时查看机器人的运动情况：

```
rosrun rviz rviz -d 'rospack find Abel_navigation' /rviz/-move_base_obstacle_avoid.rviz
```

启动 rviz 之后，会出现以下的窗口：

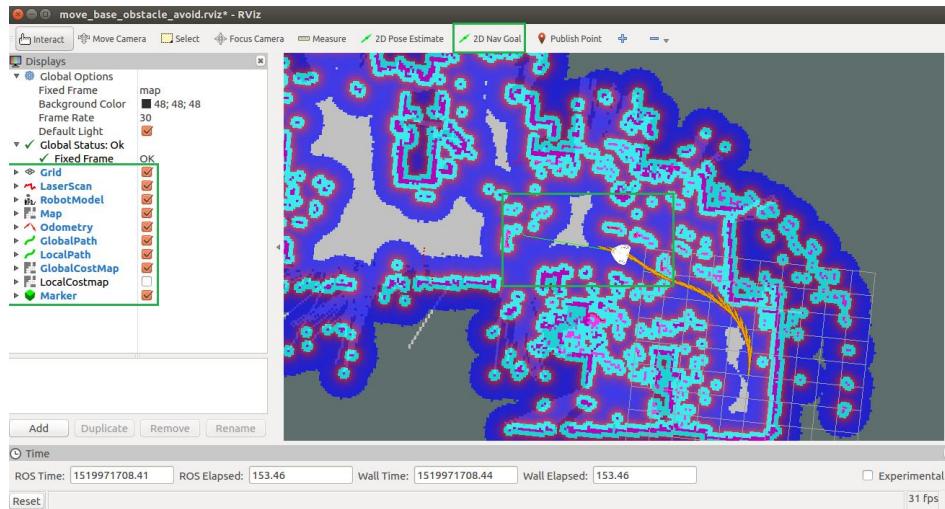


图 3.49: Abel 利用 MoveBase 和激光雷达进行动态避障示意图

启动 rviz 后加载里程计信息，另外可以选择加入全局规划路径，局部规划路径等信息。点击 Rviz 工具栏中的 2D Nav Goal 可以在地图上为机器人设置目标，设置完成后，机器人会自动规划出有效的路径并移动到目标位置，如果行驶过程中遇到行人或临时出现的障碍物，机器人会实现有效的动态避障。

功能测试完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

3.3.3 移动机器人的全自主导航

A 任务设置

这部分将编程实现移动机器人 Abel 的完全自主导航功能，即程序化的设定导航目标，机器人自主完成导航任务。

通过代码的编写，使 Abel 实现以下功能：

- 1 在地图内初始化一系列目标位置
- 2 随机选择下一个目标（每个目标被选择的机会均等）
- 3 发送正确的 MoveBaseGoal 到 move_base action server
- 4 记录到导航目标是否成功，时间间隔及距离等信息
- 5 在每个目标位置停留一段时间（时间可配置）
- 6 重复以上步骤

B 功能实现

首先进行代码的编写，使用 RoboWare Studio 打开 Abel/Abel_navigation/scripts 目录下的 autonomous_nav.py 文件，根据提示，完成实现以上机器人动作的代码练习 (TODO 练习)。

Launch 文件编写完成之后，先选择“Remote Deploy”模式，并点击旁边绿色的小锤子进行远程部署，RoboWare Studio 会显示部署结果。

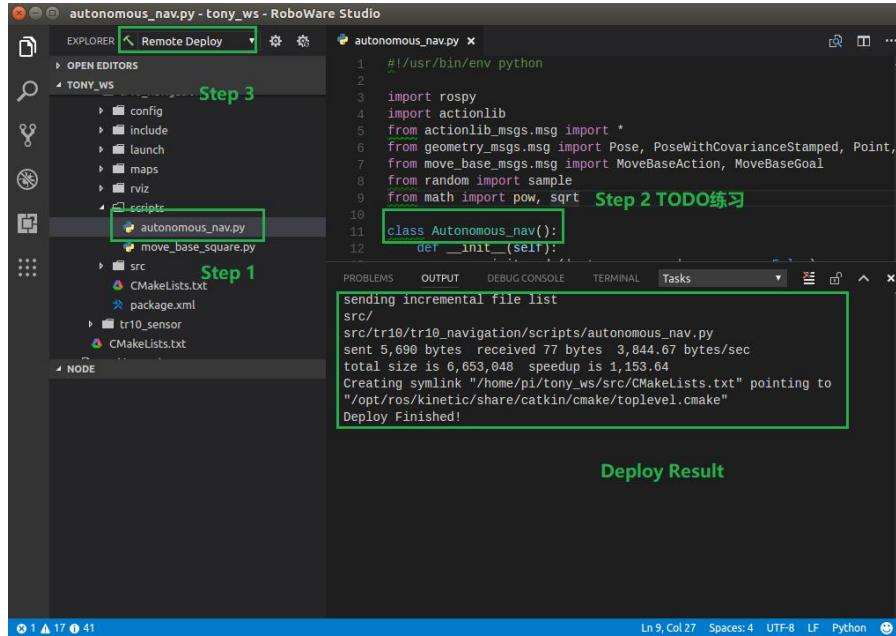


图 3.50: RoboWare Studio 对 autonomous_nav.py 文件构建及部署操作示意图

C 功能测试

代码编写及部署完成后，进行实际的功能测试。在 Abel_navigation/launch 文件夹下找到上一小节中构建的 Abel_navigation.launch 文件，在该 launch 文件上点击右键，在弹出的窗口上选择“Run Remote Launch File”选项，启动 launch 文件：

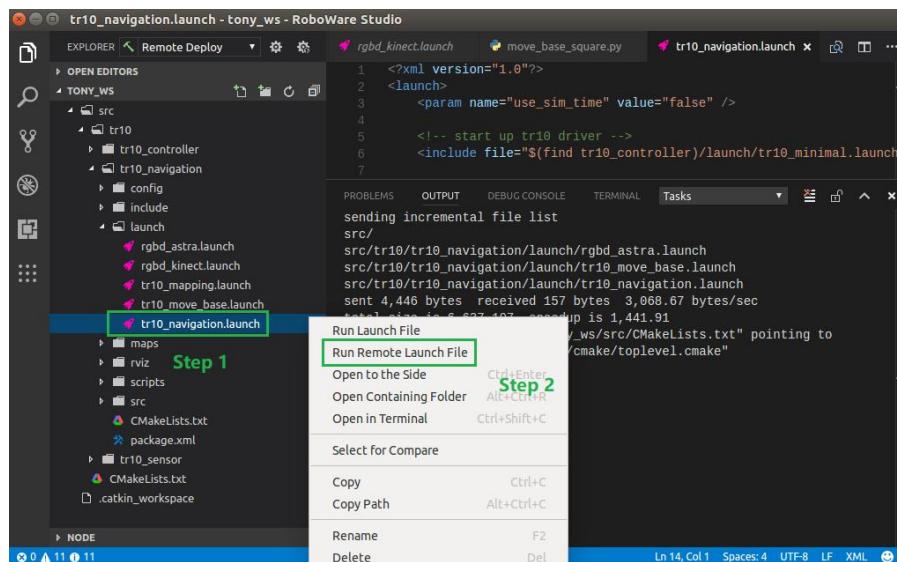


图 3.51: RoboWare Studio 远程启动 Abel_navigation.launch 文件操作示意图

按照要求，该 launch 文件会启动 Abel Base Controller，启动激光雷达驱动，启动 map_server 加载场景地图，启动 AMCL 节点，启动 MoveBase 节点并加载相应的配置参数。

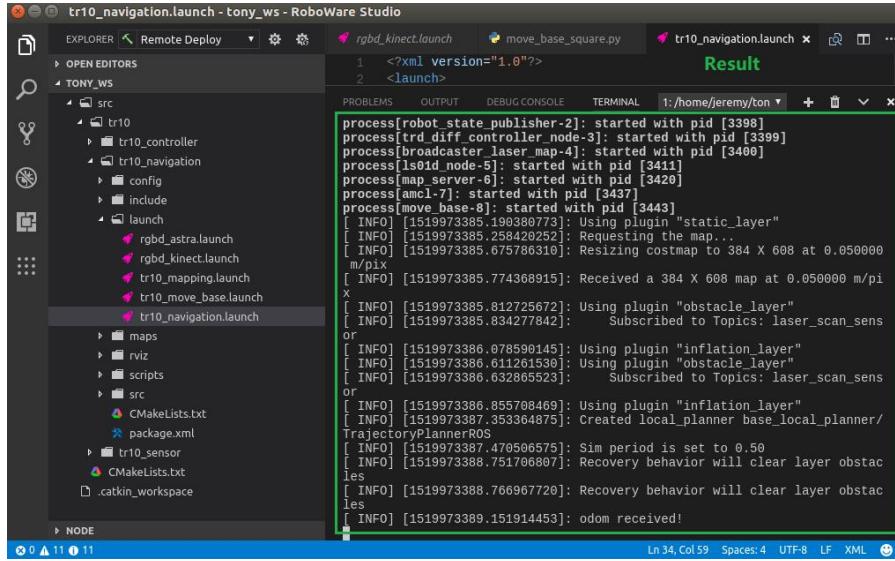


图 3.52: Abel_navigation.launch 文件远程启动成功示意图

在 PC 端打开新的终端，输入以下指令，启动 rviz，实时查看机器人的运动情况：
`rosrun rviz rviz -d ‘rospack find Abel_navigation’ /rviz/autonomous_nav.rviz`
启动 rviz 之后，在 rviz 中可以选择加载里程计信息，另外可以选择加入全局规划路径，局部规划路径等信息。会出现类似于下图的窗口：

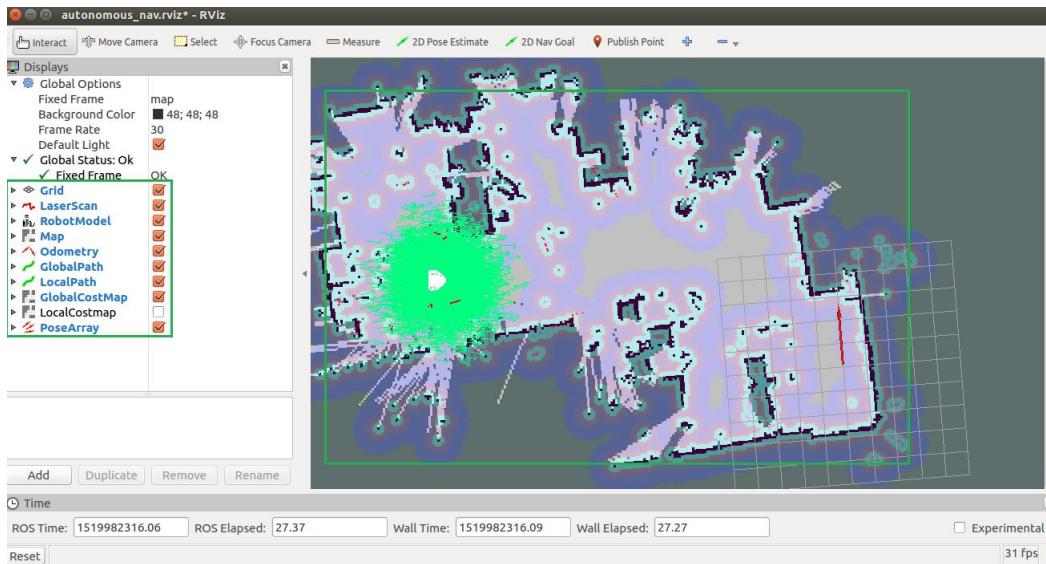


图 3.53: RViz 中查看 Abel 自主导航进行巡检的信息示意图

在 Abel_navigation/scripts 目录下选择我们刚刚构建的 autonomous_nav.py 文

件，点击右键，在弹出的列表中选择“Run Python File in Terminal”会启动 autonomous_nav.py 节点：

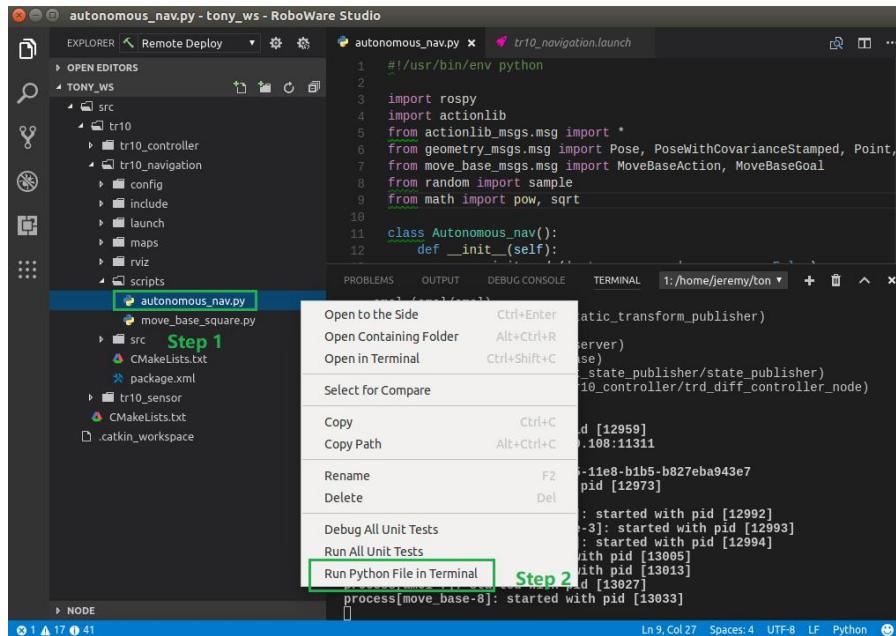


图 3.54: RoboWare Studio 远程启动 autonomous_nav.py 脚本操作示意图

节点成功启动之后，首先在 RViz 工具栏选择 2D Pose Estimate 对 Abel 初始定位，然后 Abel 会启动巡检任务，接下来会看到节点产生的一系列消息，提供巡检的进度，并提供到每个目标点的运行时间、距离及成功率等信息：

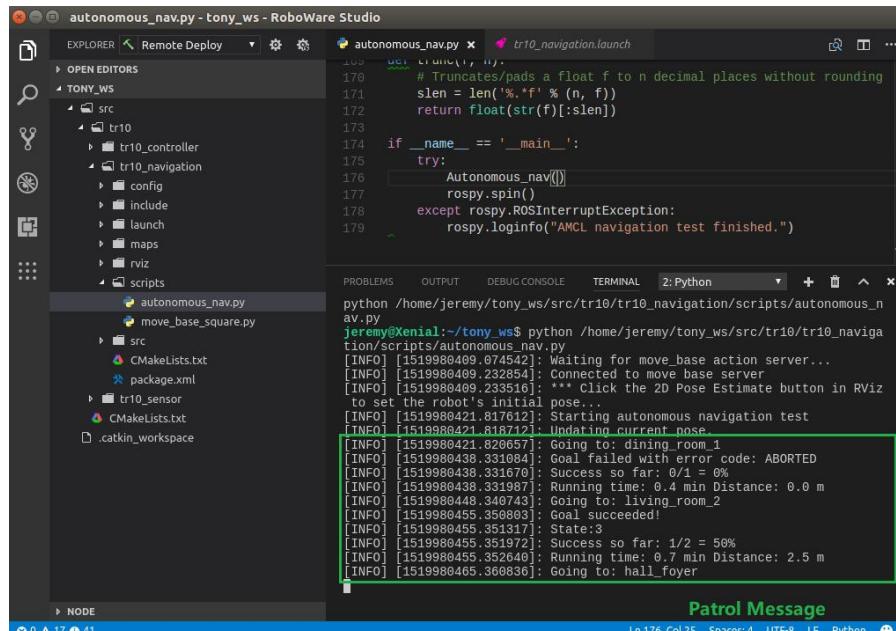


图 3.55: Abel 执行巡检任务时巡检信息提升示意图

通过 RViz 可以对 Abel 进行初始定位并查看 Abel 执行巡检任务时的运行效果：

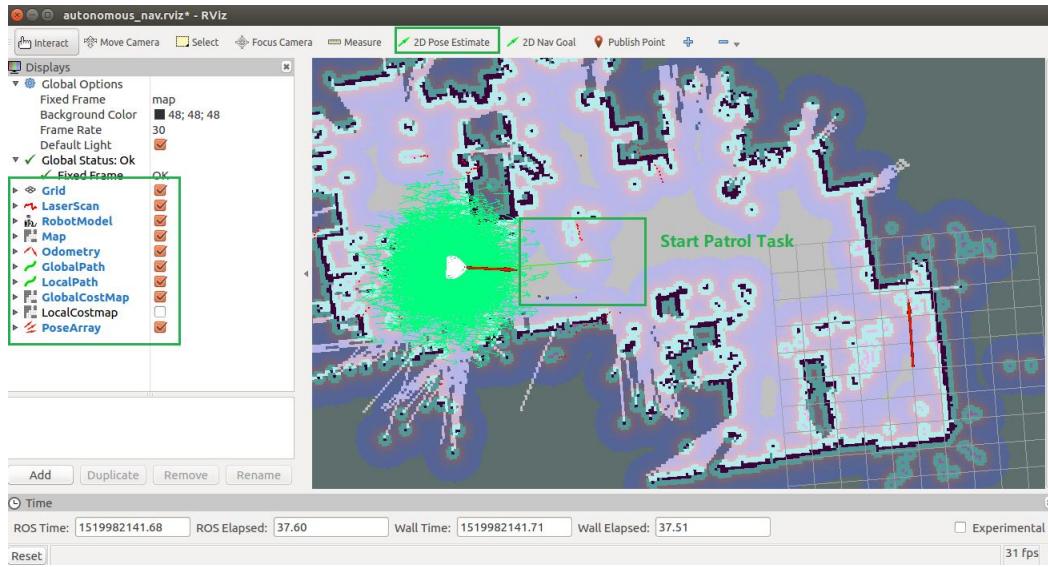


图 3.56: RViz 中查看 Abel 执行巡检任务示意图

功能测试完成之后，**CTRL+C** 关闭所有程序，**CTRL+D** 关闭所有终端。

D 小节思考

实际开发的时候可以实现其它功能，如取消一系列固定目标点设置，随机选择一系列目标位置；每次到达目标位置，机器人都对周围环境进行扫描并记录相关数据等。