ROS基础 – 实训套件

实验案例 – 教学参考手册

7 语音交互

ROS基础 – 实训套件

实验案例 – 教学参考手册

6 06 Move\_base与AMCL参数设置

ROS基础 – 实训套件

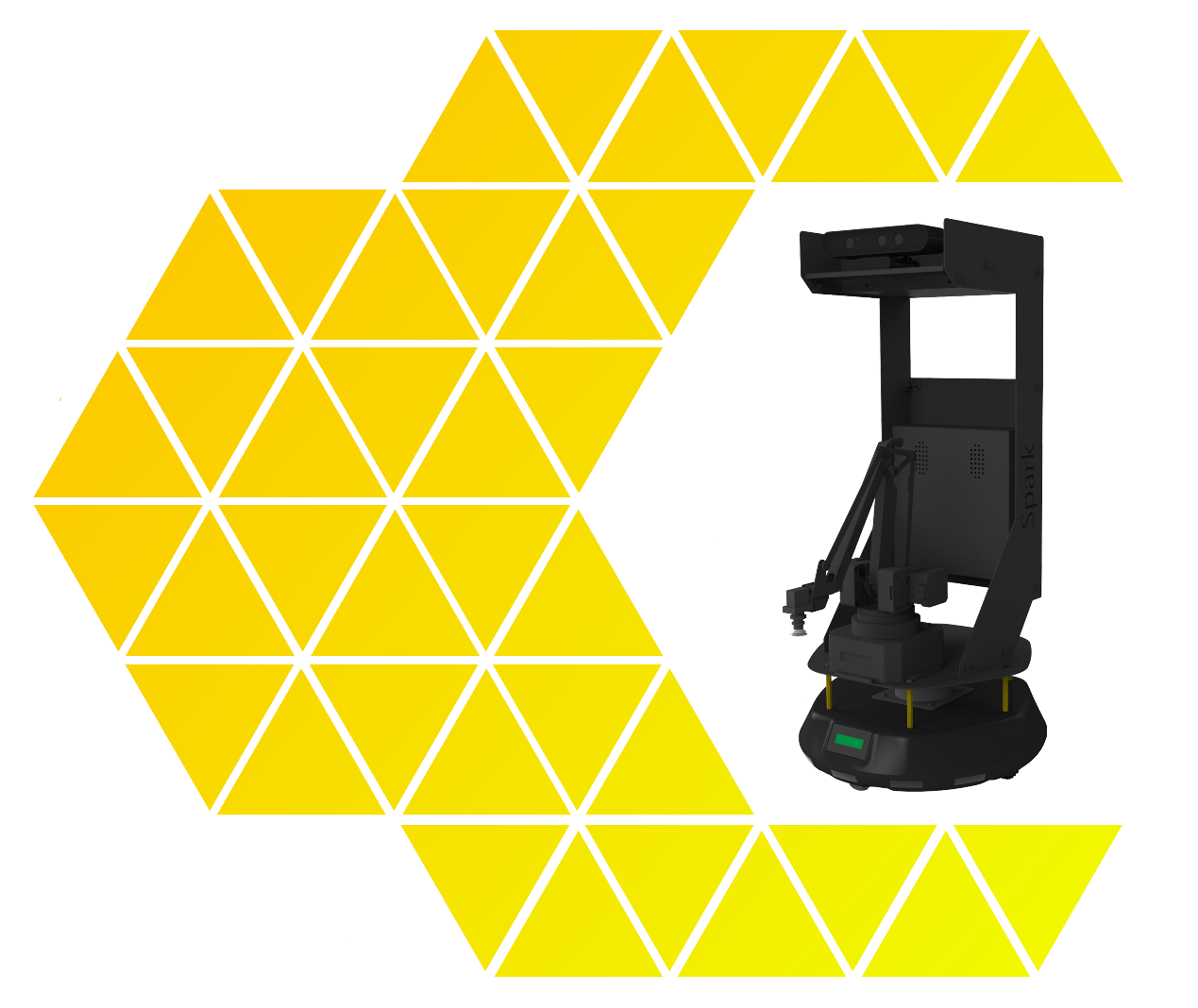
实验案例 – 教学参考手册

5 创建仿真机器人与现实机器人的同步

语音交互技术与应用

实验案例

7 语音指令控制



ROS基础 – 实训套件

实验案例 – 教学参考手册

7 语音交互

**目 录**

[一、实验名称：语音指令控制 1](#_Toc486434492)

[1、相关技能 1](#_Toc486434493)

[2、相关知识点 1](#_Toc486434494)

[3、实现效果 1](#_Toc486434495)

[4、实验要求 2](#_Toc486434496)

[5、实现思路 2](#_Toc486434497)

[6、验证与测试 2](#_Toc486434498)

[7、参考答案 2](#_Toc486434499)

# 一、实验名称：语音指令控制

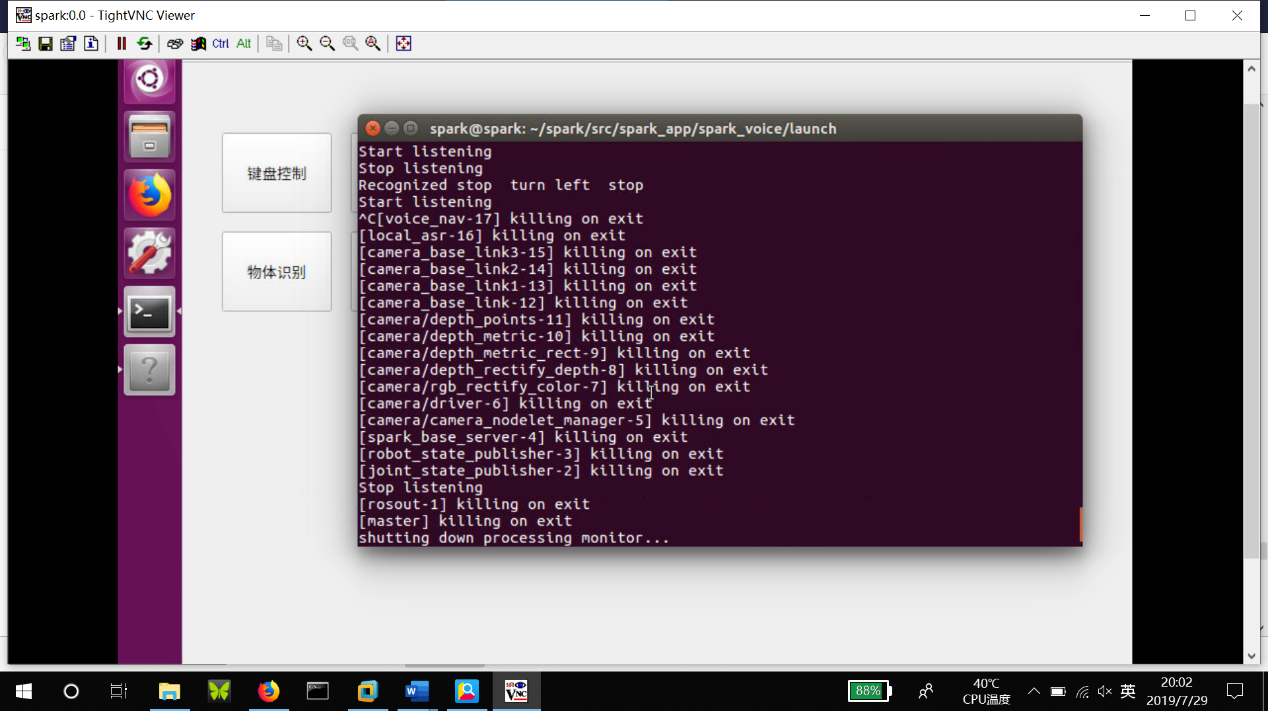
## 1、相关技能

* 通过语音指令控制SPARK移动

## 2、相关知识点

* 语音识别
* 移动导航
* 参数调整

## 3、实现效果



1 语音控制spark移动

## 4、实验要求

**本实验要求：安装语音识别功能，安装spark驱动，安装move\_base，实现对spark的语音控制**

* 安装语音识别功能
* 安装Spark驱动
* 安装move\_base

## 5、实现思路

5-1 安装pyaudio

sudo apt-get install python-pyaudio python3-pyaudio

5-2 安装Spark驱动

sudo cp $BASEPATH/doc/rules/3ilidar-usb-serial.rules /etc/udev/rules.d/

sudo cp $BASEPATH/doc/rules/uarm-usb-serial.rules /etc/udev/rules.d/

sudo cp $BASEPATH/doc/rules/spark-usb-serial.rules /etc/udev/rules.d/

sudo cp $BASEPATH/doc/rules/orbbec-usb.rules /etc/udev/rules.d/556-orbbec-usb.rules

sudo udevadm trigger

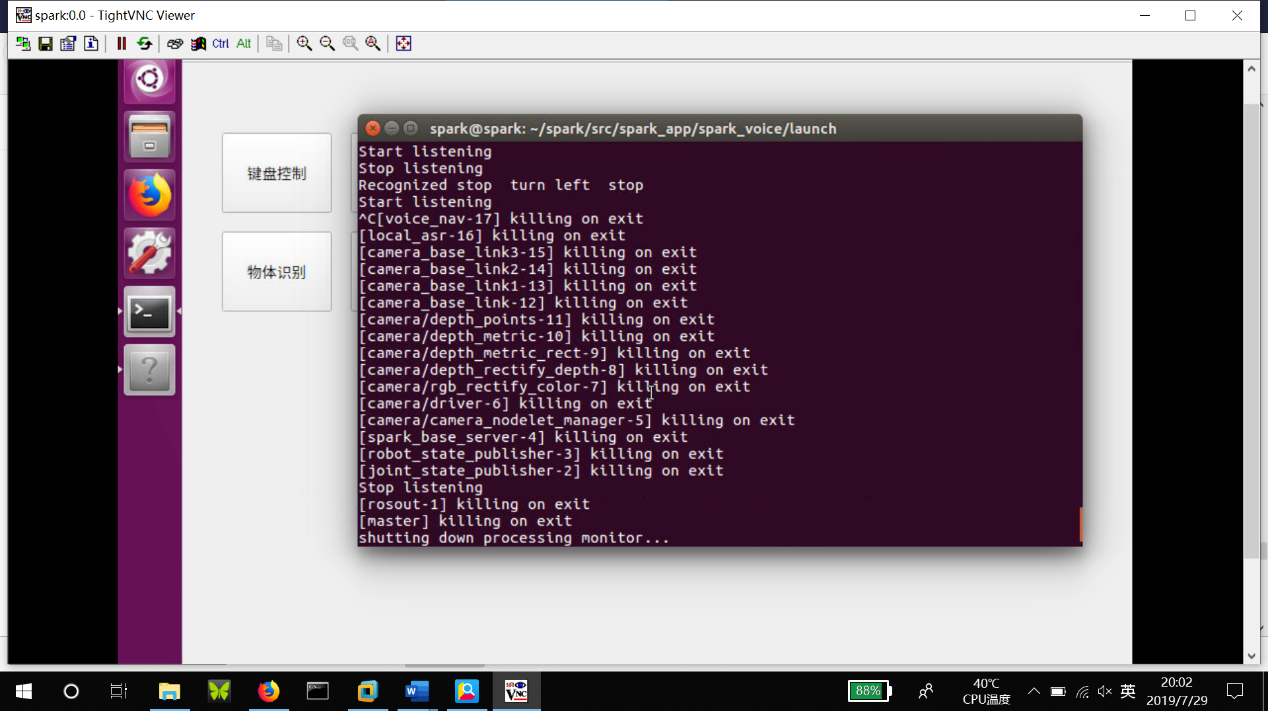
5-3 安装move\_base

sudo apt-get install ros-kinetic-move-base

## 6、验证与测试

启动语音控制,当识别语音后，就能看到小车移动了

|  |
| --- |
| **+ 提示：**  $ roslaunch spark\_voice voice\_nav.launch |



## 7、参考答案

##voice\_nav.launch

<launch>

<!--spark底盘驱动，机器人描述,底盘,相机-->

<include file="$(find

spark\_bringup)/launch/driver\_bringup.launch"/>

<!--spark键盘控制 “wsad”分别代表“前后左右”-->

<node pkg="spark\_voice" type="local\_asr.py" name="local\_asr" />

<node pkg="spark\_voice" type="voice\_nav.py" name="voice\_nav" />

</launch>

##local\_asr.py

import rospy

from std\_msgs.msg import String

import logging

import time

from threading import Thread, Event

from lib import Microphone

def main():

logging.basicConfig(level=logging.INFO)

quit\_event = Event()

mic = Microphone(quit\_event=quit\_event)

pub = rospy.Publisher('voice/stt', String, queue\_size=10)

rospy.init\_node('local\_asr', anonymous=True)

#rate = rospy.Rate(10) # 10hz

while not rospy.is\_shutdown():

# if mic.wakeup('alexa'):

# print('Wake up\n')

data = mic.listen()

text = mic.recognize(data)

hello\_str = "REC: %s" % text

rospy.loginfo(hello\_str)

if text:

pub.publish(text)

#rate.sleep()

if \_\_name\_\_ == '\_\_main\_\_':

try:

main()

except rospy.ROSInterruptException:

pass

##voice\_nav.py

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import rospy

from geometry\_msgs.msg import Twist

from std\_msgs.msg import String

from math import copysign

class VoiceNav:

def \_\_init\_\_(self):

rospy.init\_node('voice\_nav')

rospy.on\_shutdown(self.cleanup)

# Set a number of parameters affecting the robot's speed

self.max\_speed = rospy.get\_param("~max\_speed", 0.4)

self.max\_angular\_speed = rospy.get\_param("~max\_angular\_speed", 1.5)

self.speed = rospy.get\_param("~start\_speed", 0.1)

self.angular\_speed = rospy.get\_param("~start\_angular\_speed", 0.5)

self.linear\_increment = rospy.get\_param("~linear\_increment", 0.05)

self.angular\_increment = rospy.get\_param("~angular\_increment", 0.4)

# We don't have to run the script very fast

self.rate = rospy.get\_param("~rate", 5)

r = rospy.Rate(self.rate)

self.STEPS = rospy.get\_param("~steps", 20)

self.stp\_counts=0

# A flag to determine whether or not voice control is paused

self.paused = False

# Initialize the Twist message we will publish.

self.cmd\_vel = Twist()

# Publish the Twist message to the cmd\_vel topic

self.cmd\_vel\_pub = rospy.Publisher('cmd\_vel', Twist, queue\_size=5)

# Subscribe to the /voice/stt topic to receive voice commands.

rospy.Subscriber('/voice/stt', String, self.speech\_callback)

# A mapping from keywords or phrases to commands

self.keywords\_to\_command = {'stop': ['stop','停止'],

'forward': ['forward','前进'],

'backward': ['backward','后退'],

'turn left': ['turn left','左转'],

'turn right': ['turn right','右转']}

rospy.loginfo("Ready to receive voice commands")

# We have to keep publishing the cmd\_vel message if we want the robot to keep moving.

while not rospy.is\_shutdown():

self.stp\_counts+=1

if(self.stp\_counts>=self.STEPS):

self.cmd\_vel = Twist()

self.stp\_counts=0

self.cmd\_vel\_pub.publish(self.cmd\_vel)

r.sleep()

def get\_command(self, data):

# Attempt to match the recognized word or phrase to the

# keywords\_to\_command dictionary and return the appropriate

# command

for (command, keywords) in self.keywords\_to\_command.iteritems():

for word in keywords:

if data.find(word) > -1:

return command

def speech\_callback(self, msg):

# Get the motion command from the recognized phrase

command = self.get\_command(msg.data)

self.stp\_counts=0

# Log the command to the screen

rospy.loginfo("Command: " + str(command))

# If the user has asked to pause/continue voice control,

# set the flag accordingly

if command == 'pause':

self.paused = True

elif command == 'continue':

self.paused = False

# If voice control is paused, simply return without

# performing any action

if self.paused:

return

# The list of if-then statements should be fairly

# self-explanatory

if command == 'forward':

self.cmd\_vel.linear.x = self.speed

self.cmd\_vel.angular.z = 0

elif command == 'rotate left':

self.cmd\_vel.linear.x = 0

self.cmd\_vel.angular.z = self.angular\_speed

elif command == 'rotate right':

self.cmd\_vel.linear.x = 0

self.cmd\_vel.angular.z = -self.angular\_speed

elif command == 'turn left':

if self.cmd\_vel.linear.x != 0:

self.cmd\_vel.angular.z += self.angular\_increment

else:

self.cmd\_vel.angular.z = self.angular\_speed

elif command == 'turn right':

if self.cmd\_vel.linear.x != 0:

self.cmd\_vel.angular.z -= self.angular\_increment

else:

self.cmd\_vel.angular.z = -self.angular\_speed

elif command == 'backward':

self.cmd\_vel.linear.x = -self.speed

self.cmd\_vel.angular.z = 0

elif command == 'stop':

# Stop the robot! Publish a Twist message consisting of all zeros.

self.cmd\_vel = Twist()

elif command == 'faster':

self.speed += self.linear\_increment

self.angular\_speed += self.angular\_increment

if self.cmd\_vel.linear.x != 0:

self.cmd\_vel.linear.x += copysign(self.linear\_increment, self.cmd\_vel.linear.x)

if self.cmd\_vel.angular.z != 0:

self.cmd\_vel.angular.z += copysign(self.angular\_increment, self.cmd\_vel.angular.z)

elif command == 'slower':

self.speed -= self.linear\_increment

self.angular\_speed -= self.angular\_increment

if self.cmd\_vel.linear.x != 0:

self.cmd\_vel.linear.x -= copysign(self.linear\_increment, self.cmd\_vel.linear.x)

if self.cmd\_vel.angular.z != 0:

self.cmd\_vel.angular.z -= copysign(self.angular\_increment, self.cmd\_vel.angular.z)

elif command in ['quarter', 'half', 'full']:

if command == 'quarter':

self.speed = copysign(self.max\_speed / 4, self.speed)

elif command == 'half':

self.speed = copysign(self.max\_speed / 2, self.speed)

elif command == 'full':

self.speed = copysign(self.max\_speed, self.speed)

if self.cmd\_vel.linear.x != 0:

self.cmd\_vel.linear.x = copysign(self.speed, self.cmd\_vel.linear.x)

if self.cmd\_vel.angular.z != 0:

self.cmd\_vel.angular.z = copysign(self.angular\_speed, self.cmd\_vel.angular.z)

else:

return

self.cmd\_vel.linear.x = min(self.max\_speed, max(-self.max\_speed, self.cmd\_vel.linear.x))

self.cmd\_vel.angular.z = min(self.max\_angular\_speed, max(-self.max\_angular\_speed, self.cmd\_vel.angular.z))

def cleanup(self):

# When shutting down be sure to stop the robot!

twist = Twist()

self.cmd\_vel\_pub.publish(twist)

rospy.sleep(1)

if \_\_name\_\_=="\_\_main\_\_":

try:

VoiceNav()

rospy.spin()

except rospy.ROSInterruptException:

rospy.loginfo("Voice navigation terminated.")