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

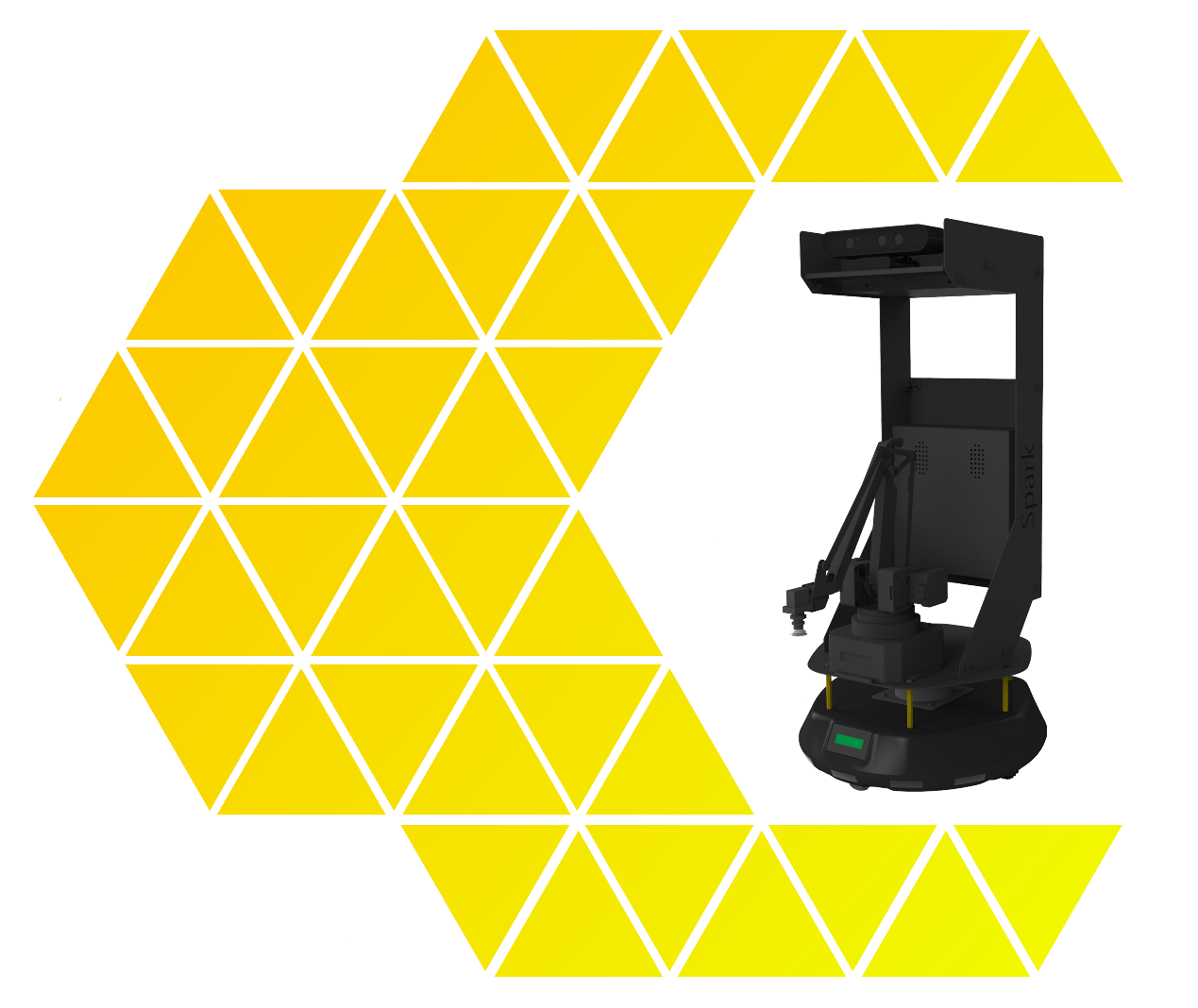
实验案例 – 教学参考手册

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

6 在线语音合成



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：在线语音合成 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 3](#_Toc486434499)

# 一、实验名称：在线语音合成

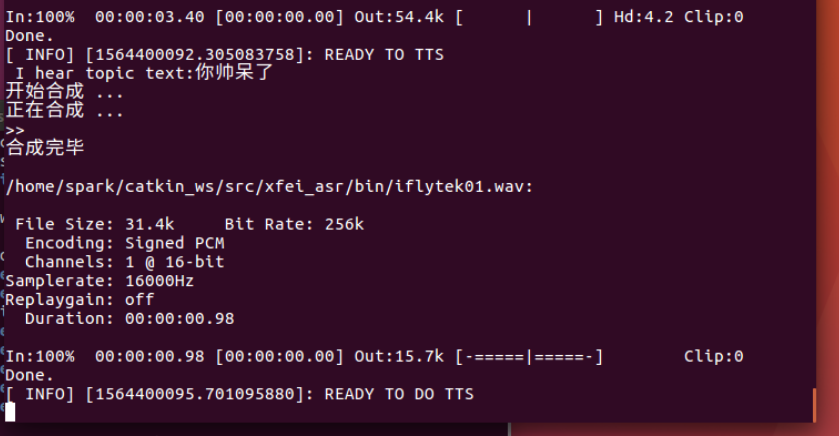
## 1、相关技能

能通过在线进行语音合成

## 2、相关知识点

* 开通讯飞云语音合成功能
* RESTFULL接口调用
* 参数调整及功能调试
* 如何播放语音

## 3、实现效果



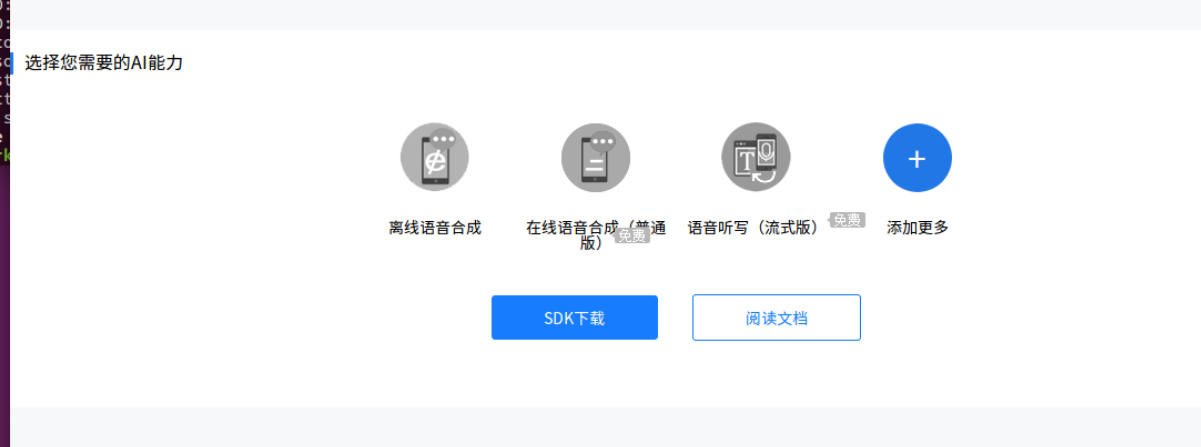
1 在线语音合成

## 实验要求

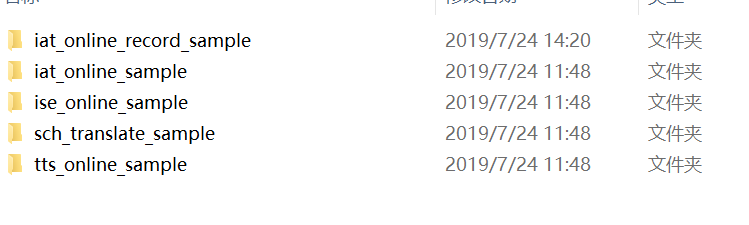
1. **本实验**要求**：**讯飞云帐号，开通语音合成功能，编写ROS应用节点。
   1. 讯飞云帐号，开通语音合成功能
   2. 编写ROS应用节点（见7.0答案）

## 5、实现思路

* 1. 讯飞云帐号，开通语音合成功能



5-2 下载SDK，并解压



5-3将科大 tts 应用到ros节点上

1)创建包

cd catkin\_ws/src

catkin\_creat\_pkg voice\_system roscpp rospy std\_msgs

将科大迅飞 包下 include目录下 的 头文件拷贝到 刚刚创建的包voice下的include内

msp\_cmn.h

msp\_errors.h

msp\_types.h

qise.h

qisr.h

qtts.h

2)创建节点文件

根据 tts.sample.c文件修改创建 kdxf\_ttf.cpp文件

主要就是订阅一个自定义话题上的文本消息 传给科大tts 播放wav文件

3）见源码（7.0答案）

## 6、验证与测试

1)修改CMakeLists 文件

共有三个部分：

a) 添加 头文件搜索 .h文件

include\_directories( #头文件路径

include #cpp文件自建的头文件

${catkin\_INCLUDE\_DIRS})

b) 添加编译源文件信息

add\_executable(xftts src/xf\_ttf.cpp)

c) 添加动态搜索库 .o文件 可执行文件名

target\_link\_libraries(xftts ${catkin\_LIBRARIES} -lmsc -lrt -ldl -lpthread ) #系统库 和外库（由原先的 make文件得来）

注意可能会提示找不到 -lmsc

可替换为完整路径:

/home/ewenwan/ewenwan/catkin\_ws/src/voice/libs/x64/libmsc.so #注意更换为自己的路径

2) 编译

cd catkin\_ws

catkin\_make

3) 依赖

play 播放wav文件 // 需要安装 sudo apt-get install sox

4) 实验

rosrun voice\_system xftts

发布话题 数据

rostopic pub -1 voice/xf\_tts\_topic std\_msgs/String "你帅呆了"

## 7、参考答案

/\*

\* 语音合成（Text To Speech，TTS）技术能够自动将任意文字实时转换为连续的

\* 自然语音，是一种能够在任何时间、任何地点，向任何人提供语音信息服务的

\* 高效便捷手段，非常符合信息时代海量数据、动态更新和个性化查询的需求。

\*/

#include "ros/ros.h" //ros系统头文件

#include "std\_msgs/String.h"

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include "qtts.h"

#include "msp\_cmn.h"

#include "msp\_errors.h"

const char\* filename = "/home/ewenwan/ME/music/music.wav";

const char\* playwavpath = "play /home/ewenwan/ME/music/music.wav";

/\* wav音频头部格式 \*/

typedef struct \_wave\_pcm\_hdr

{

char riff[4]; // = "RIFF" RIFF(Resource Interchange File Format) 资源交换 文件规范

int size\_8; // = FileSize - 8 从下个地址开始到文件尾的总字节数

char wave[4]; // = "WAVE" WAV文件标志（WAVE）

char fmt[4]; // = "fmt " 波形格式标志（fmt ），最后一位空格。

int fmt\_size; // = 下一个结构体的大小 : 16 过滤字节（一般为00000010H，16d），若为00000012H则说明数据头携带附加信息（见“附加信息”）。

short int format\_tag; // = PCM : 1 格式种类（值为1时，表示数据为线性PCM编码）

short int channels; // = 通道数 : 1 通道数，单声道为1，双声道为2

int samples\_per\_sec; // = 采样率 : 8000 | 6000 | 11025 | 16000

int avg\_bytes\_per\_sec; // = 每秒字节数 : samples\_per\_sec \* bits\_per\_sample / 8

short int block\_align; // = 每采样点字节数 : wBitsPerSample / 8

short int bits\_per\_sample; // = 量化比特数: 8 | 16

char data[4]; // = "data";

int data\_size; // = 纯数据长度 : FileSize - 44

} wave\_pcm\_hdr;

/\* 默认wav音频头部数据 \*/

wave\_pcm\_hdr default\_wav\_hdr =

{

{ 'R', 'I', 'F', 'F' }, //文件规范

0, //后面数据的大小（前面有4个字节）

{'W', 'A', 'V', 'E'}, //文件格式

{'f', 'm', 't', ' '}, //波形格式标志 最后一位 空

16, //过滤字节数

1, //格式种类（值为1时，表示数据为线性PCM编码）

1, //通道数，单声道为1，双声道为2

16000, //采样率 : 8000 | 6000 | 11025 | 16000

32000, //每秒字节数 : samples\_per\_sec \* bits\_per\_sample / 8

2, //每采样点字节数 : wBitsPerSample / 8

16, //量化比特数: 8 | 16

{'d', 'a', 't', 'a'}, //"data";

0 //纯音频数据长度 数据长度 : FileSize - 44 （前面有40个字节）

};

/\* 文本合成 函数 text, filename, session\_begin\_params \*/

int text\_to\_speech(const char\* src\_text, const char\* des\_path, const char\* params)

{

int ret = -1; //返回参数

FILE\* fp = NULL; //文件句柄 文件头

const char\* sessionID = NULL; //

unsigned int audio\_len = 0;

wave\_pcm\_hdr wav\_hdr = default\_wav\_hdr; //wav文件头

int synth\_status = MSP\_TTS\_FLAG\_STILL\_HAVE\_DATA;

if (NULL == src\_text || NULL == des\_path) //原文本空或者 生成的文件名路径为空 返回

{

printf("params is error!\n");

return ret;

}

fp = fopen(des\_path, "wb"); //二进制格式写入

if (NULL == fp)

{

printf("open %s error.\n", des\_path); //打开文件失败

return ret;

}

/\* 开始合成 合成语音参数 标志\*/

sessionID = QTTSSessionBegin(params, &ret);//开始一次语音合成，分配语音合成资源。

if (MSP\_SUCCESS != ret) //参数不合规范

{

printf("QTTSSessionBegin failed, error code: %d.\n", ret);

fclose(fp); //关闭文件

return ret;

}

// int MSPAPI QTTSTextPut ( const char \* sessionID, const char \* textString, unsigned int textLen, const char \* params )

// 由QTTSSessionBegin返回的句柄 字符串指针,指向待合成的文本字符串 合成文本长度,最大支持8192个字节（2730个汉字） 本次合成所用的参数，只对本次合成的文本有效。

ret = QTTSTextPut(sessionID, src\_text, (unsigned int)strlen(src\_text), NULL);//写入待合成的文本。

if (MSP\_SUCCESS != ret)

{

printf("QTTSTextPut failed, error code: %d.\n",ret);

QTTSSessionEnd(sessionID, "TextPutError"); //发生错误的话 就结束本次语音合成。

fclose(fp); //关闭文件

return ret;

}

printf("正在合成 ...\n");

//写入文件 头信息 大小 写入次数 文件名

fwrite(&wav\_hdr, sizeof(wav\_hdr) ,1, fp); //添加wav音频头，使用采样率为16000

while (1)

{

/\* 获取合成音频 \*/ // 由QTTSSessionBegin返回的句柄 合成音频长度,单位字节 合成音频状态 成功与否

const void\* data = QTTSAudioGet(sessionID, &audio\_len, &synth\_status, &ret); //获取合成音频。

if (MSP\_SUCCESS != ret) //为成功

break;

if (NULL != data) //合成的音频有内容

{

fwrite(data, audio\_len, 1, fp); //写入内容 长度 次数 文件名

wav\_hdr.data\_size += audio\_len; //总音频长度计算data\_size大小 用于记录头文件 size\_8 大小

}

if (MSP\_TTS\_FLAG\_DATA\_END == synth\_status) //合成的音频已经取完

break;

printf(">");

usleep(150\*1000); //防止频繁占用CPU

}

printf("\n");

if (MSP\_SUCCESS != ret) //获取音频未成功

{

printf("QTTSAudioGet failed, error code: %d.\n",ret);

QTTSSessionEnd(sessionID, "AudioGetError");//发生错误的话 就结束本次语音合成。

fclose(fp);

return ret;

}

/\* 修正wav文件头数据的大小 \*/

wav\_hdr.size\_8 += wav\_hdr.data\_size + (sizeof(wav\_hdr) - 8); //头文件长度+音频长度 -8

/\* 将修正过的数据写回文件头部,音频文件为wav格式 \*/

fseek(fp, 4, 0); //偏移4个字节 从第4个字节开始写入 size\_8

fwrite(&wav\_hdr.size\_8,sizeof(wav\_hdr.size\_8), 1, fp); //写入size\_8的值

fseek(fp, 40, 0); //偏移4个字节 //将文件指针偏移到存储data\_size值的位置

fwrite(&wav\_hdr.data\_size,sizeof(wav\_hdr.data\_size), 1, fp); //写入data\_size的值

fclose(fp); //关闭文件

fp = NULL;

/\* 合成完毕 \*/

ret = QTTSSessionEnd(sessionID, "Normal"); //结束本次语音合成。

if (MSP\_SUCCESS != ret)

{

printf("QTTSSessionEnd failed, error code: %d.\n",ret);

}

return ret;

}

/\*

make topic Text To Wave file

\*/

int makeTextToWave(const char\* text, const char\* filename)

{

int ret = MSP\_SUCCESS; //默认返回参数

/\* appid 应用ID。\*/

const char\* login\_params = "appid = 58dbcf6e, work\_dir = .";//登录参数,appid与msc库绑定,请勿随意改动

/\*

\* rdn: 合成音频数字发音方式 0：数值优先（车牌号报读）,1：完全数值(1000 一千),2：完全字符串，3：字符串优先。 默认为0

\* volume: 合成音频的音量 [0,100]，数值越大音量越大。默认为50

\* pitch: 合成音频的音调 [0,100]，数值越大音调越高。默认为50

\* speed: 合成音频对应的语速 [0,100]，数值越大语速越快。默认为50

\* voice\_name: 合成发音人 xiaoyan yanping jinger yufeng donaldduck babyxu nannan xiaoqian（东北话）

\* sample\_rate: 合成音频采样率

\* text\_encoding: 合成文本编码格式 GB2312；GBK；BIG5；UNICODE；GB18030；UTF8

\* background\_sound 合成音频中的背景音 0：无背景音乐 1：有背景音乐。默认为0

\* 详细参数说明请参阅《讯飞语音云MSC--API文档》

\*/

const char\* session\_begin\_params = "voice\_name = donaldduck, text\_encoding = utf8, sample\_rate = 16000, speed = 50, volume = 50, pitch = 50, rdn = 0";

//const char\* filename = "tts\_sample.wav"; //合成的语音文件名称

//const char\* text = "大家好，我叫小明，车牌号沪A1005，两套房子，五百万存款"; //合成文本

/\* 用户登录 int MSPAPI MSPLogin ( const char \* usr,const char \* pwd,const char \* params )\*/

ret = MSPLogin(NULL, NULL, login\_params);//第一个参数是用户名，第二个参数是密码，第三个参数是登录参数，用户名和密码可在http://www.xfyun.cn注册获取

if (MSP\_SUCCESS != ret)

{

printf("MSPLogin failed, error code: %d.\n", ret);

goto exit ;//登录失败，退出登录

}

/\* 文本合成 \*/

printf("开始合成 ...\n");

/\*要合成语音的文本 合成后的语音文件wav名字 合成语音的参数\*/

ret = text\_to\_speech(text, filename, session\_begin\_params);

if (MSP\_SUCCESS != ret)

{

printf("text\_to\_speech failed, error code: %d.\n", ret);

}

else{

printf("合成完毕\n");

}

exit:

MSPLogout(); //退出登录

//return 0;

}

/\*播放 wav文件\*/

void playWav()

{

system(playwavpath); // 需要安装 sudo apt-get install sox

}

/\*subcribe topic callbace function\*/

void chatterCallback(const std\_msgs::String::ConstPtr& msg)

{

//ROS\_INFO("I heard: [%s]", msg->data.c\_str());

//const char\* text=msg->data.c\_str();

std::cout << " I hear topic text:" << msg->data.c\_str() << std::endl;

makeTextToWave( msg->data.c\_str() , filename );

playWav();

ROS\_INFO("READY TO DO TTS");

}

int main(int argc, char\*\* argv)

{

const char\* start\_voice ="科大讯飞在线语音合成启动成功";

makeTextToWave( start\_voice , filename );

playWav();

ROS\_INFO("READY TO TTS");

ros::init(argc, argv, "tts"); //初始化ros系统 ，在roscore节点管理器注册节点

ros::NodeHandle nhd; //节点句柄

//节点创建一个发布者

//ros::Publisher chatter\_pub = nh.advertise<std\_msgs::String>("pub\_hello\_ros", 1000);

//创建一个订阅者sub 节点句柄 话题 缓存区 函数指针 &callbackfunc 得到

ros::Subscriber sub = nhd.subscribe("voice/tts", 100, &chatterCallback);

//ros::spin();

ros::Rate rate(5); //发布频率

while(ros::ok()){

ros::spinOnce(); //给ROS控制权 可以调用一次回调函数

rate.sleep();

}

return 0;

}