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

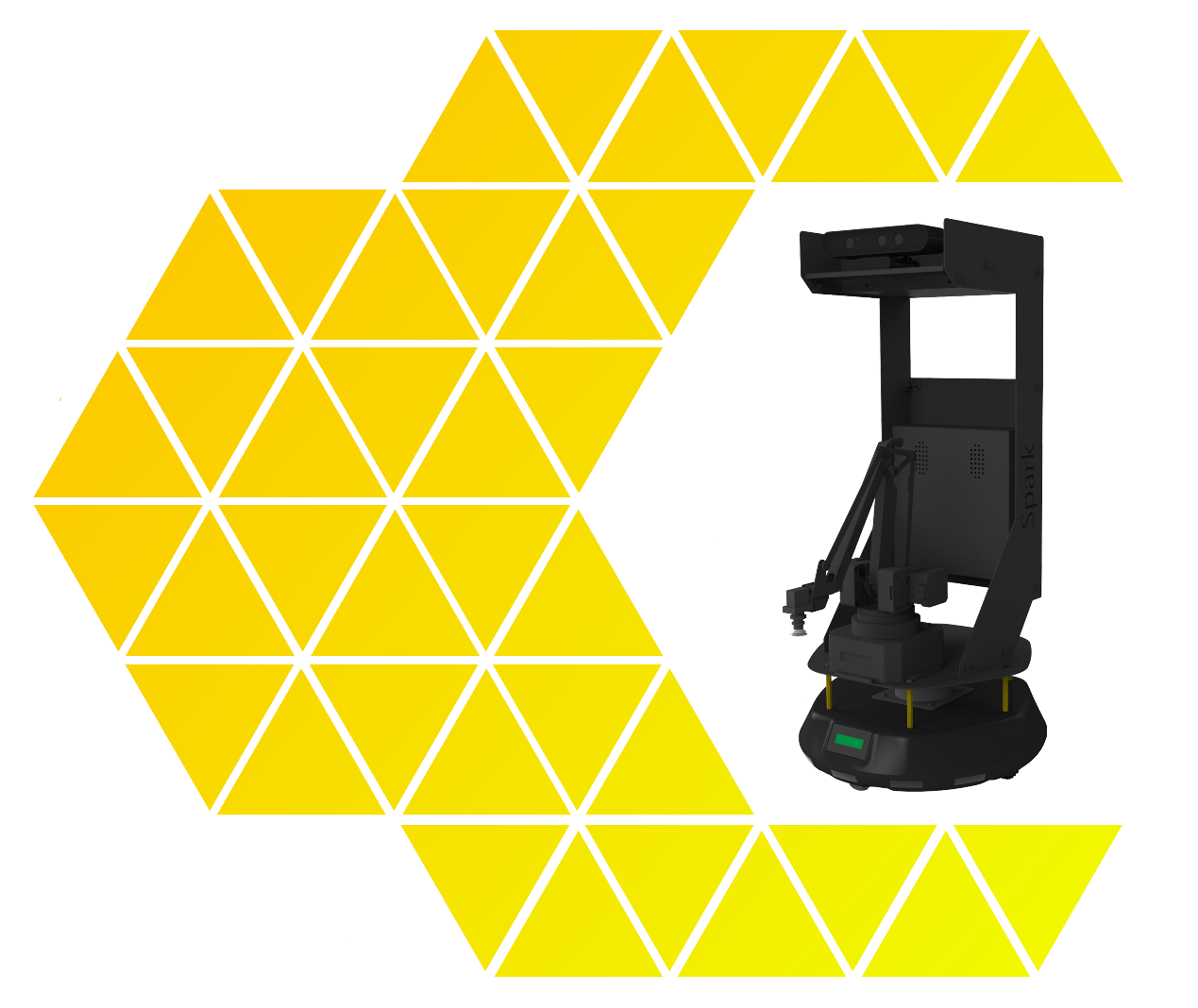
实验案例 – 教学参考手册

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

3 在线语音识别



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：在线语音识别 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 3](#_Toc486434498)

[7、参考答案 3](#_Toc486434499)

# 一、实验名称：**在线语音识别**

## 1、相关技能

能够在线进行语音识别

## 2、相关知识点

开通讯飞云语音功能

RESTFULL接口调用

参数调整及功能调试

* 参数调整及功能调试

## 3、实现效果



1 查看语音识别结果

## 4、实验要求

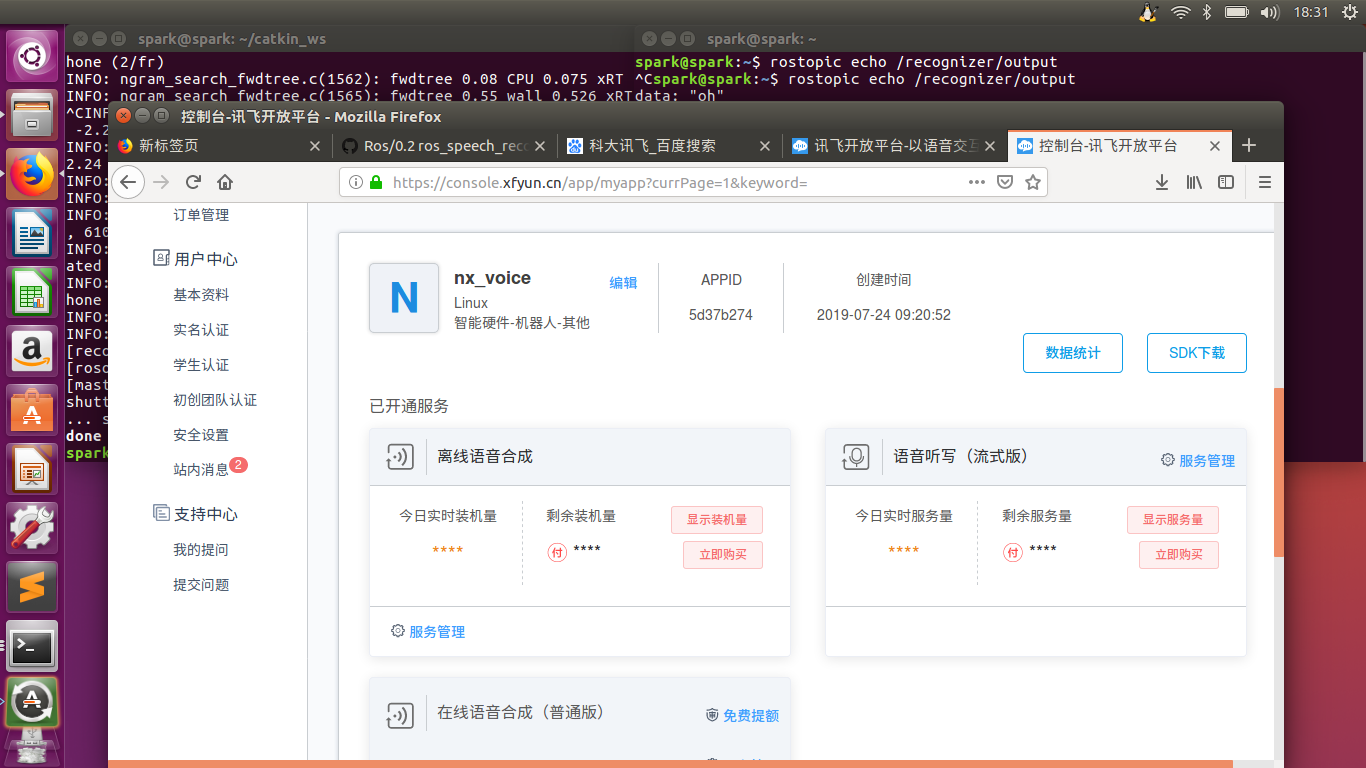
**本实验要求：注册科大讯飞账号，下载语音听写JDK包，编写ROS源程序，实现在线语音识别。**

* 1. 注册科大讯飞账号并下载JDK包
  2. 将科大的包应用到ros节点
  3. 实现在线语音识别

## 5、实现思路

* 1. 注册科大讯飞账号并下载JDK包

|  |
| --- |
| **+ 提示：**  **科大讯飞开放平台 https://www.xfyun.cn/**   1. 注册账号 2. 下载语音听写JDK包   记住应用的APPID |



* 1. 将科大讯飞的包应用到ros节点

|  |
| --- |
| **+ 提示：**  1)创建包  cd catkin\_ws/src  catkin\_creat\_pkg voice\_tx 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  示例目录下：  /voice/samples/ita\_record\_sample  .c源文件 拷贝到 包的源文件下 catkin\_ws/src/voice\_tx/src  .h源文件 拷贝到 包的包含文件下 catkin\_ws/src/voice\_tx/include  2)创建节点文件  根据 ita\_record\_sample.c文件修改创建 xf\_asr.cpp文件  （参考7.0答案）    3)编写CMakeLists.txt文件 |
| **(参考7.0答案)** |
| **4）编译**  **cd ~/catkin\_ws/**  **catkin\_make -DCAKIN\_** |
| **Source ~/catkin\_ws/devel/setup.bash** |

## 6、验证与测试

6.1 测试在线语音识别

6.1.1 终端运行node节点

|  |
| --- |
| **+ 提示：**  **roscore**  rosrun voice\_tx xf\_asr |

6.1.2 往voice\_asr\_topic主题发布启动命令

|  |
| --- |
| **+ 提示：**  rostopic pub -1 voice/xf\_asr\_topic std\_msgs/Int32 1 |

6.1.3 对着麦克风讲话，会识别出来



## 7、参考答案

/\*

\* 语音听写(iFly Auto Transform)技术能够实时地将语音转换成对应的文字。

\* voice Activity Detection 语音活动检测 一句话是否说完了

\*/

#include<ros/ros.h>

#include<std\_msgs/String.h>

#include<std\_msgs/Int32.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include "qisr.h"

#include "msp\_cmn.h"

#include "msp\_errors.h"

#include "speech\_recognizer.h"

#define FRAME\_LEN 640

#define BUFFER\_SIZE 4096

#define ASRFLAG 1 //控制命令

using namespace std; //命名空间

//全局变量

bool flag = false; //发布语音识别结果话题 标志

bool recorder\_Flag = true; //可录音标志，开始麦克风录音并上传识别

string result = ""; //识别结果字符串

/\* 上传用户字典\*/

static int upload\_userwords()

{

char\* userwords = NULL; //

size\_t len = 0; //

size\_t read\_len = 0; //

FILE\* fp = NULL; //

int ret = -1; //

fp = fopen("userwords.txt", "rb"); //打开用户字典文本

if (NULL == fp)

{

printf("\nopen [userwords.txt] failed! \n"); //打开失败

goto upload\_exit;

}

fseek(fp, 0, SEEK\_END); //到文件结尾

len = ftell(fp); //偏移长度 字节 相当于文件大小

fseek(fp, 0, SEEK\_SET); //到文件头

userwords = (char\*)malloc(len + 1); //申请文件字节+1 大小的缓存区

if (NULL == userwords)

{

printf("\nout of memory! \n"); //缓存区申请失败

goto upload\_exit;

}

read\_len = fread((void\*)userwords, 1, len, fp); //读取文件写入到 缓存区

if (read\_len != len)

{

printf("\nread [userwords.txt] failed!\n"); //文件应该相等

goto upload\_exit;

}

userwords[len] = '\0'; //添加结束符号

//用户数据上传 数据名称字符串 待上传数据缓冲区的起始地址。 数据长度(如果是字符串，则不包含'\0')。

//params[in] "sub = uup,dtt = userword" 上传用户词表 iat业务 UTF-8 编码

MSPUploadData("userwords", userwords, len, "sub = uup, dtt = userword", &ret); //上传

if (MSP\_SUCCESS != ret)

{

printf("\nMSPUploadData failed ! errorCode: %d \n", ret);//上传失败

goto upload\_exit;

}

//退出

upload\_exit:

if (NULL != fp)

{

fclose(fp); //关闭文件

fp = NULL;

}

if (NULL != userwords)

{

free(userwords);//释放缓存区

userwords = NULL;

}

return ret;

}

/\*打印结果\*/

static void show\_result(char \*str, char is\_over)

{

printf("\rResult: [ %s ]", str);

if(is\_over)

putchar('\n');

string s(str);

result = s; //得到语音识别结果

flag = true; //设置发布话题为真

}

//全局变量

static char \*g\_result = NULL;

static unsigned int g\_buffersize = BUFFER\_SIZE;

//对结果采取的措施

void on\_result(const char \*result, char is\_last)

{

if (result) {

size\_t left = g\_buffersize - 1 - strlen(g\_result);

size\_t size = strlen(result);

if (left < size) {

g\_result = (char\*)realloc(g\_result, g\_buffersize + BUFFER\_SIZE);

if (g\_result)

g\_buffersize += BUFFER\_SIZE;

else {

printf("mem alloc failed\n");

return;

}

}

strncat(g\_result, result, size);

show\_result(g\_result, is\_last);

}

}

/\*一段新的语句开始\*/

void on\_speech\_begin()

{

if (g\_result)

{

free(g\_result);

}

g\_result = (char\*)malloc(BUFFER\_SIZE);

g\_buffersize = BUFFER\_SIZE;

memset(g\_result, 0, g\_buffersize);

printf("Start Listening...\n");

}

/\* 说话结束\*/

void on\_speech\_end(int reason)

{

if (reason == END\_REASON\_VAD\_DETECT) //已经检测到 语句vad断点了 就是说了一句话了

{

printf("\nSpeaking done \n");

recorder\_Flag = false; //录音标志 录音结束

}

else

printf("\nRecognizer error %d\n", reason);

}

/\* 从麦克风识别语音\*/

static void demo\_mic(const char\* session\_begin\_params)

{

int errcode;

struct speech\_rec iat;

struct speech\_rec\_notifier recnotifier = {

on\_result,

on\_speech\_begin,

on\_speech\_end

};

errcode = sr\_init(&iat, session\_begin\_params, SR\_MIC, &recnotifier);

if (errcode) {

printf("speech recognizer init failed\n");

return;

}

errcode = sr\_start\_listening(&iat);

if (errcode) {

printf("start listen failed %d\n", errcode);

}

while(recorder\_Flag){ //当可录音标志为真时 开始录音并识别

sleep(1);

}

errcode = sr\_stop\_listening(&iat);

if (errcode) {

printf("stop listening failed %d\n", errcode);

}

sr\_uninit(&iat);

}

int asrToText()

{

int ret = MSP\_SUCCESS;

int upload\_on = 1; /\* whether upload the user word \*/

/\* login params, please do keep the appid correct \*/

const char\* login\_params = "appid = 58dbcf6e, work\_dir = ."; //登陆参数

int aud\_src = 0; /\* from mic or file \*/

/\*

\* See "iFlytek MSC Reference Manual"

\* 识别参数

\*/

const char\* session\_begin\_params =

"sub = iat, domain = iat, language = zh\_cn, "

"accent = mandarin, sample\_rate = 16000, "

"result\_type = plain, result\_encoding = utf8";

/\* Login first. the 1st arg is username, the 2nd arg is password

\* just set them as NULL. the 3rd arg is login paramertes

\* 登陆

\* \*/

ret = MSPLogin(NULL, NULL, login\_params);

if (MSP\_SUCCESS != ret) {

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

goto exit; // login fail, exit the program

}

/\*字典上传？\*/

/\*

printf("Want to upload the user words ? \n0: No.\n1: Yes\n");

scanf("%d", &upload\_on);

if (upload\_on)

{

printf("Uploading the user words ...\n");

ret = upload\_userwords();

if (MSP\_SUCCESS != ret)

goto exit;

printf("Uploaded successfully\n");

}

\*/

/\*声音来源 麦克分或 wav文件\*/

/\*

printf("Where the audio comes from?\n0: From a audio file.\n1: From microphone.\n");

scanf("%d", &aud\_src);//键盘输入

if(aud\_src != 0) {

printf("Demo recognizing the speech from microphone\n");

printf("Speak in 15 seconds\n");

demo\_mic(session\_begin\_params);

printf("15 sec passed\n");

} else {

printf("Demo recgonizing the speech from a recorded audio file\n");

demo\_file("wav/iflytek02.wav", session\_begin\_params); //选者wav文件

}

\*/

demo\_mic(session\_begin\_params);

exit:

MSPLogout(); // Logout...

}

/\*

\* 根据发布的话题来修改录音标志

\*/

void asrCallBack(const std\_msgs::Int32::ConstPtr &msg)

{

ROS\_INFO\_STREAM("Topic is Subscriber, now starting voice recognition");

if(msg->data == ASRFLAG) //话题收到相应的 语音识别激活标志

{

asrToText();

}

}

/\* main thread: start/stop record ; query the result of recgonization.

\* record thread: record callback(data write)

\* helper thread: ui(keystroke detection)

\*/

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

{

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

ros::NodeHandle nhd; //节点句柄

//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/xf\_asr\_topic", 20, &asrCallBack);

//节点创建一个发布者

ros::Publisher pub = nhd.advertise<std\_msgs::String>("voice/tl\_nlu\_topic", 20);

ROS\_INFO("please publish the kinds of std\_msgs/Int32 1 to the topic voice/xf\_asr\_topic");

ROS\_INFO("waitting for the running sign of voice recognition");

//ros::spin();

ros::Rate rate(10); //频率

while(ros::ok()){

if(flag) //成功获取到返回数据

{

std\_msgs::String msg;

msg.data = result;

pub.publish(msg);

recorder\_Flag = true; //可开始录音识别

flag = false ; //发布识别结果 话题 标志 为否

}

ros::spinOnce(); //给ROS控制权 可以调用一次回调函数

rate.sleep();

}

return 0;;

}

###CMakeLists.txt文件####

cmake\_minimum\_required(VERSION 2.8.3)

project(voice\_system)

## Add support for C++11, supported in ROS Kinetic and newer

# add\_definitions(-std=c++11)

## Find catkin macros and libraries

## if COMPONENTS list like find\_package(catkin REQUIRED COMPONENTS xyz)

## is used, also find other catkin packages

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

std\_msgs

)

## System dependencies are found with CMake's conventions

# find\_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro ensures

## modules and global scripts declared therein get installed

## See http://ros.org/doc/api/catkin/html/user\_guide/setup\_dot\_py.html

# catkin\_python\_setup()

################################################

## Declare ROS messages, services and actions ##

################################################

## To declare and build messages, services or actions from within this

## package, follow these steps:

## \* Let MSG\_DEP\_SET be the set of packages whose message types you use in

## your messages/services/actions (e.g. std\_msgs, actionlib\_msgs, ...).

## \* In the file package.xml:

## \* add a build\_depend tag for "message\_generation"

## \* add a build\_depend and a run\_depend tag for each package in MSG\_DEP\_SET

## \* If MSG\_DEP\_SET isn't empty the following dependency has been pulled in

## but can be declared for certainty nonetheless:

## \* add a run\_depend tag for "message\_runtime"

## \* In this file (CMakeLists.txt):

## \* add "message\_generation" and every package in MSG\_DEP\_SET to

## find\_package(catkin REQUIRED COMPONENTS ...)

## \* add "message\_runtime" and every package in MSG\_DEP\_SET to

## catkin\_package(CATKIN\_DEPENDS ...)

## \* uncomment the add\_\*\_files sections below as needed

## and list every .msg/.srv/.action file to be processed

## \* uncomment the generate\_messages entry below

## \* add every package in MSG\_DEP\_SET to generate\_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder

# add\_message\_files(

# FILES

# Message1.msg

# Message2.msg

# )

## Generate services in the 'srv' folder

# add\_service\_files(

# FILES

# Service1.srv

# Service2.srv

# )

## Generate actions in the 'action' folder

# add\_action\_files(

# FILES

# Action1.action

# Action2.action

# )

## Generate added messages and services with any dependencies listed here

# generate\_messages(

# DEPENDENCIES

# std\_msgs

# )

################################################

## Declare ROS dynamic reconfigure parameters ##

################################################

## To declare and build dynamic reconfigure parameters within this

## package, follow these steps:

## \* In the file package.xml:

## \* add a build\_depend and a run\_depend tag for "dynamic\_reconfigure"

## \* In this file (CMakeLists.txt):

## \* add "dynamic\_reconfigure" to

## find\_package(catkin REQUIRED COMPONENTS ...)

## \* uncomment the "generate\_dynamic\_reconfigure\_options" section below

## and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder

# generate\_dynamic\_reconfigure\_options(

# cfg/DynReconf1.cfg

# cfg/DynReconf2.cfg

# )

###################################

## catkin specific configuration ##

###################################

## The catkin\_package macro generates cmake config files for your package

## Declare things to be passed to dependent projects

## INCLUDE\_DIRS: uncomment this if you package contains header files

## LIBRARIES: libraries you create in this project that dependent projects also need

## CATKIN\_DEPENDS: catkin\_packages dependent projects also need

## DEPENDS: system dependencies of this project that dependent projects also need

catkin\_package(

# INCLUDE\_DIRS include

# LIBRARIES voice\_system

# CATKIN\_DEPENDS roscpp rospy std\_msgs

# DEPENDS system\_lib

)

###########

## Build ##

###########

## Specify additional locations of header files

## Your package locations should be listed before other locations

# include\_directories(include)

include\_directories( #头文件路径

include #cpp文件自建的头文件

${catkin\_INCLUDE\_DIRS}

)

## Declare a C++ library

# add\_library(${PROJECT\_NAME}

# src/${PROJECT\_NAME}/voice\_system.cpp

# )

## Add cmake target dependencies of the library

## as an example, code may need to be generated before libraries

## either from message generation or dynamic reconfigure

# add\_dependencies(${PROJECT\_NAME} ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

## Declare a C++ executable

## With catkin\_make all packages are built within a single CMake context

## The recommended prefix ensures that target names across packages don't collide

# add\_executable(${PROJECT\_NAME}\_node src/voice\_system\_node.cpp)

## Rename C++ executable without prefix

## The above recommended prefix causes long target names, the following renames the

## target back to the shorter version for ease of user use

## e.g. "rosrun someones\_pkg node" instead of "rosrun someones\_pkg someones\_pkg\_node"

# set\_target\_properties(${PROJECT\_NAME}\_node PROPERTIES OUTPUT\_NAME node PREFIX "")

## Add cmake target dependencies of the executable

## same as for the library above

# add\_dependencies(${PROJECT\_NAME}\_node ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

## Specify libraries to link a library or executable target against

# target\_link\_libraries(${PROJECT\_NAME}\_node

# ${catkin\_LIBRARIES}

# )

# TTS

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

target\_link\_libraries(

xftts

${catkin\_LIBRARIES}

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

-lrt

-ldl

-lpthread ) #系统库 和外库 msc rt dl pthread

#############

## Install ##

#############

# all install targets should use catkin DESTINATION variables

# See http://ros.org/doc/api/catkin/html/adv\_user\_guide/variables.html

## Mark executable scripts (Python etc.) for installation

## in contrast to setup.py, you can choose the destination

# install(PROGRAMS

# scripts/my\_python\_script

# DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

# )

## Mark executables and/or libraries for installation

# install(TARGETS ${PROJECT\_NAME} ${PROJECT\_NAME}\_node

# ARCHIVE DESTINATION ${CATKIN\_PACKAGE\_LIB\_DESTINATION}

# LIBRARY DESTINATION ${CATKIN\_PACKAGE\_LIB\_DESTINATION}

# RUNTIME DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

# )

## Mark cpp header files for installation

# install(DIRECTORY include/${PROJECT\_NAME}/

# DESTINATION ${CATKIN\_PACKAGE\_INCLUDE\_DESTINATION}

# FILES\_MATCHING PATTERN "\*.h"

# PATTERN ".svn" EXCLUDE

# )

## Mark other files for installation (e.g. launch and bag files, etc.)

# install(FILES

# # myfile1

# # myfile2

# DESTINATION ${CATKIN\_PACKAGE\_SHARE\_DESTINATION}

# )

#############

## Testing ##

#############

## Add gtest based cpp test target and link libraries

# catkin\_add\_gtest(${PROJECT\_NAME}-test test/test\_voice\_system.cpp)

# if(TARGET ${PROJECT\_NAME}-test)

# target\_link\_libraries(${PROJECT\_NAME}-test ${PROJECT\_NAME})

# endif()

## Add folders to be run by python nosetests

# catkin\_add\_nosetests(test)