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

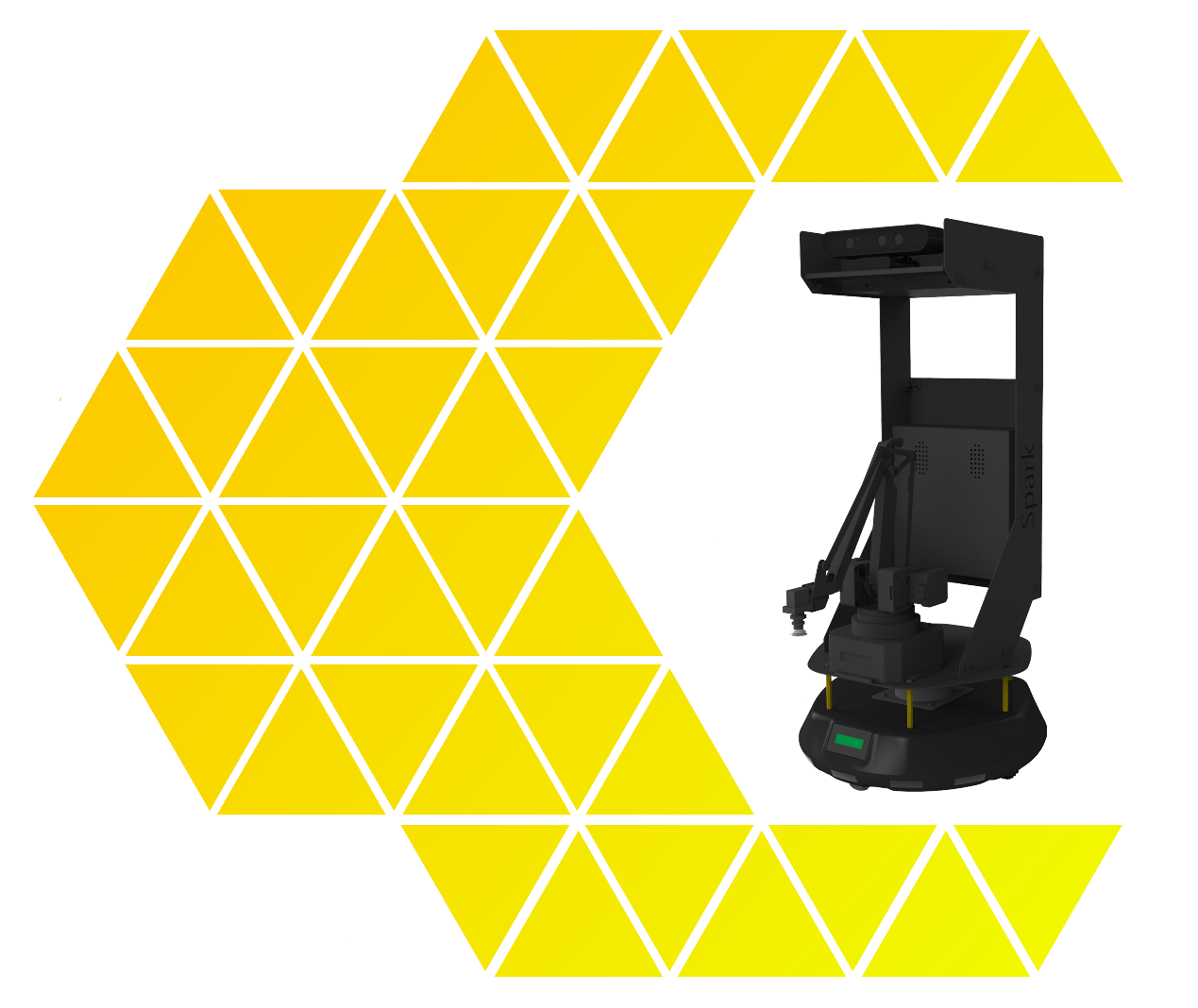
实验案例 – 教学参考手册

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

4 语义分析



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：语义分析 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 4](#_Toc486434498)

[7、参考答案 5](#_Toc486434499)

# 一、实验名称：**语音采集与处理**

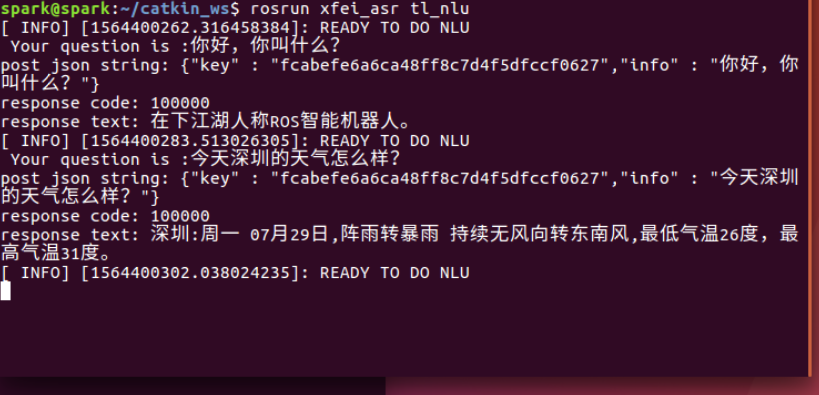
## 1、相关技能

* 可以通过接口获取音频流

## 2、相关知识点

* 语义理解相关概念
* 如何添加技能插件
* 调用参数调整
* JSON格式处理

## 3、实现效果



1 使用图灵机器人进行对话

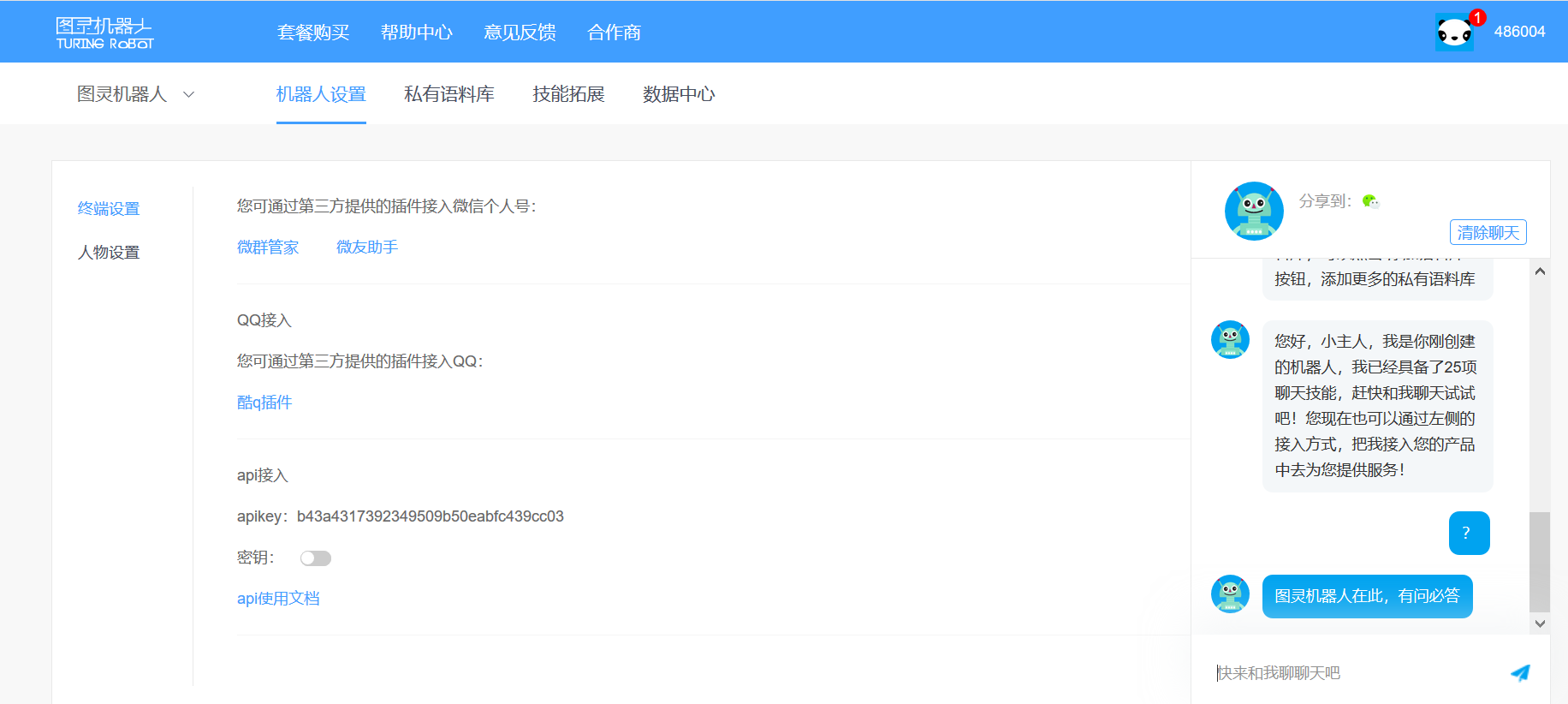
## 4、实验要求

**本实验要求：注册图灵机器人账号，记录下apikey,编写ros应用，实现订阅语音内容，以JSON格式发给图灵机器人。**

* 1. 注册图灵机器人账号，并记录下apikey
  2. 获取语音内容
  3. 以JSON格式发送语音内容
  4. 获取返回的数据

## 5、实现思路

* 1. 注册图灵机器人账号
     1. 记录apikey



* 1. 安装依赖库

|  |
| --- |
| sudo apt-get install libcurl3 libcurl4-openssl-dev //curl httppose访问  sudo apt-get install libjsoncpp1 libjsoncpp-dev //json字符串 |

* 1. 启动ROS语音识别节点

|  |
| --- |
| **+ 提示：**  rosrun xfei\_asr xf\_asr |

* 1. 编写ROS语义分析节点并启动

5-3.1编写ROS语义分析节点

|  |
| --- |
| **+ 提示：**  源码见7.0答案，  修改自己的apikey  roscd catkin\_ws  catkin\_make  source ~/catkin\_ws/devel/setup.bash |

5-3.2启动ROS语义分析节点

|  |
| --- |
| **+ 提示：**  rosrun xfei\_asr tl\_nlu |

## 6、验证与测试

1)修改CMakeLists 文件

# NLU

add\_executable(tlnlu src/tl\_nlu.cpp)

target\_link\_libraries(tlnlu ${catkin\_LIBRARIES} -lcurl -ljsoncpp) #系统库 和外库 curl jsoncpp

2) 编译

cd catkin\_ws

catkin\_make

3) 实验

roscore

rosrun voice\_system xftts // 文本转换成语言

rosrun voice\_system tlnlu // 语意理解

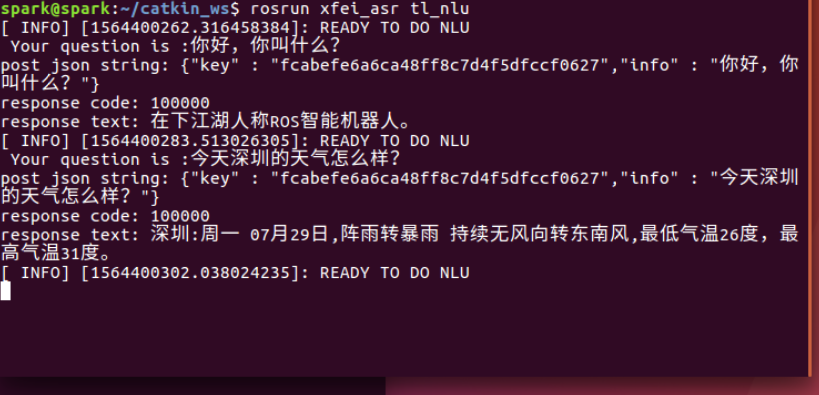
向主题发消息,会收到回复

|  |
| --- |
| **+ 提示：**  rostopic pub -1 /voice/tl\_nlu\_topic std\_msgs/String "给我说一个绕口令吧" |

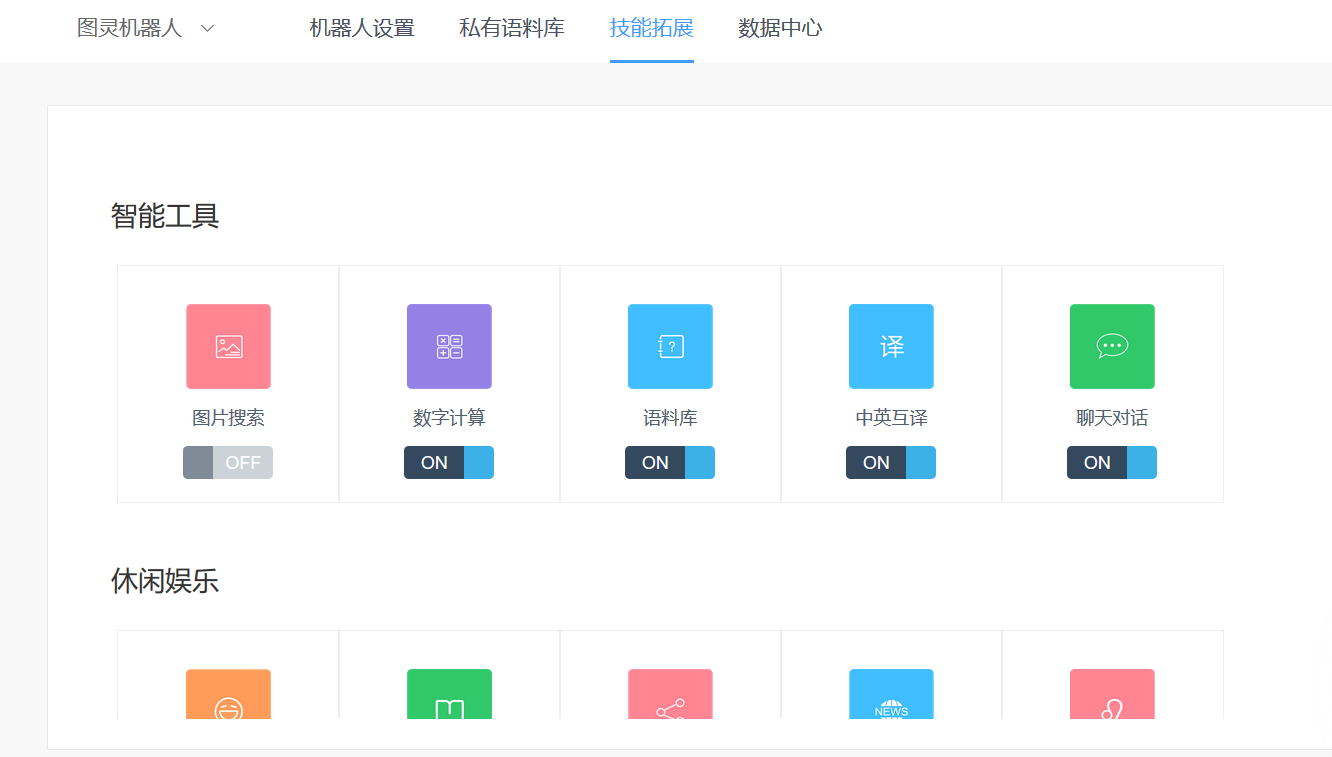
对着麦克风讲话

|  |
| --- |
| **比如讲：**  深圳的天气怎么样？ |

会收到智能的回复



我们还可以添加私有预料与技能拓展



## 7、参考答案

图灵 NLU 在线语意理解

\*/

#include "ros/ros.h" //ros系统头文件

#include "std\_msgs/String.h"

#include <sstream>

#include <jsoncpp/json/json.h> //json字符串

#include <curl/curl.h> //curl http访问

#include <string>

#include <exception>

using namespace std;

//全局变量

int flag = 0;

string result;

/\*解析图灵服务器返回的json字符串\*/

int parseJsonResonse(string input\_str)

{

Json::Value root;

Json::Reader reader;

bool parsingSuccessful = reader.parse(input\_str, root);

if( !parsingSuccessful )

{

std::cout << "Fail to parse the response data" << std::endl;

return -1;

}

const Json::Value code = root["code"];//文本类型标识吗

const Json::Value text = root["text"];//返回的结果

result = text.asString(); //返回文本内容

flag = 1;

std::cout << "response code: " << code << std::endl;

std::cout << "response text: " << result << std::endl;

return 0;

}

/\*将接收到的返回数据写如内存\*/

int writer(char \*data, size\_t size, size\_t number, string \*writerData)

{

if ( writerData == NULL)

{

return -1;

}

unsigned long len = size \* number;

writerData->append(data, len);

return len;

}

/\*HTTP 请求\*/

int HttpPostRequest(string input\_str)

{

string buffer;

std::string strJson = "{" ;

strJson += "\"key\" : \"fcabefe6a6ca48ff8c7d4f5dfccf0627\",";

strJson += "\"info\" : ";

strJson += "\"";

strJson += input\_str;

strJson += "\"";

strJson += "}";

std::cout << "post json string: " << strJson << std::endl;

try

{

CURL \*pCurl = NULL;

CURLcode res; //返回状态

curl\_global\_init(CURL\_GLOBAL\_ALL); //初始化 pCurl

pCurl = curl\_easy\_init(); //头

if( NULL != pCurl)

{

curl\_easy\_setopt(pCurl, CURLOPT\_TIMEOUT,5); //延迟时间

curl\_easy\_setopt(pCurl, CURLOPT\_URL,"http://www.tuling123.com/openapi/api");

// http头 // 设置http发送的内容类型为JSON

curl\_slist \*plist = curl\_slist\_append(NULL, "Content-Type:application/json;charset=UTF-8");

curl\_easy\_setopt(pCurl, CURLOPT\_HTTPHEADER, plist);

// 设置要POST的JSON数据

curl\_easy\_setopt(pCurl, CURLOPT\_POSTFIELDS,strJson.c\_str());

curl\_easy\_setopt(pCurl, CURLOPT\_WRITEFUNCTION, &writer);

curl\_easy\_setopt(pCurl, CURLOPT\_WRITEDATA, &buffer);

//执行http请求

res = curl\_easy\_perform(pCurl);

//检查错误

if (res != CURLE\_OK)

{

printf("curl\_easy\_perform failed:%s\n", curl\_easy\_strerror(res));

}

curl\_easy\_cleanup(pCurl); //清除当前http请求

}

curl\_global\_cleanup(); //全部清除

}

catch (std::exception &ex)

{

printf("!!! curl exception: %s.\n", ex.what());

}

if(buffer.empty())

{

std::cout << "!!! ERROR The Tuling server response NULL" << std::endl;

}

else

{

parseJsonResonse(buffer);

}

return 0;

}

/\*

\* 当voice/tl\_nlu 话题有消息时，调用HttpPostRequest向图灵服务器发送内容，返回结果。

\*/

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

{

std::cout << " Your question is :" << msg->data << std::endl;

HttpPostRequest(msg->data);

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

}

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

{

ros::init(argc, argv, "tl\_nlu\_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/tl\_nlu\_topic", 20, &nluCallback);

//节点创建一个发布者

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

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

//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);

flag = 0 ;

}

ros::spinOnce(); //给ROS控制权 可以调用一次回调函数

rate.sleep();

}

return 0;

}