搭建以展品讲解为主要内容的语音对话系统

2020212185张扬

任务

基本功能：

1. 最能够展示机器人语音对话能力 **已实现**
2. 自然语言理解（识别句子的意图和槽），从而实现有针对性的问答 **已实现**
3. 对话状态跟踪，记录任务进展，依据对话状态给出回复。从而实现参观顺序的引导 **已实现**

扩展功能：

1. 可打断，机器人说话过程中可以被打断，转换话题 **已实现**
2. 基于产品信息的问答，（可通过机器阅读理解功能实现）**已实现**

提交内容：1. 代码（代码需能够执行）2. 实验报告 3. 功能展示所需的录音或录像

特点：实现ChatGPT回复

语音转文字和文字转语音

查阅 `leju` robot官网，得到 `S2T` 类和 `T2S` 类，对其进行重写如下：

S2T (Speech-to-Text)

```
class S2T():
    def __init__(self):
        my_museum=museum()
        rospy.Subscriber("/aiui/nlp", String, my_museum.voice_guide)
        rospy.spin()                        # 控制 ROS 系统中的消息循环

    def nlp_callback(self, msg):
        rospy.loginfo(msg.data)             # 用 ros 的 log 输出结果
```

我重写了官网名为S2T（Speech-to-Text）的类。

1. `class S2T()`：
    - 定义了一个名为S2T的类，表示语音到文本的转换。
2. `my_museum = museum()`
    - 创建了一个名为 `my_museum` 的 `museum` 类（由我编写）的实例，将其赋值给 `my_museum` 变量。
3. `rospy.Subscriber("/aiui/nlp", String, my_museum.voice_guide)`
    - 创建了一个ROS的订阅者（**Subscriber**），订阅了名为 `"/aiui/nlp"` 的主题（Topic）。当有新消息发布到该主题时，会调用 `my_museum.voice_guide` 方法来处理消息。
4. `rospy.spin()`
    - 控制ROS系统中的消息循环。它会一直运行，直到程序被终止。
5. `def nlp_callback(self, msg):`
    - 定义了一个名为nlp_callback的方法，它接收一个名为msg的参数。

6. `rospy.loginfo(msg.data)`
    - 使用ROS的日志功能，在控制台输出msg.data的内容。msg.data是传入nlp_callback方法的消息数据。

总体来说，创建一个S2T类的实例，该实例订阅ROS主题"/aiui/nlp"上的消息，并将收到的消息传递给my_museum.voice_guide方法进行处理。

同时，通过使用 `rospy.spin()` 来保持ROS系统的消息循环运行，以便接收和处理新的消息。当收到新的消息时，消息的数据将被记录到ROS的日志中。

T2S(Text-to-Speech)

```
class T2S():
    def __init__(self):
        self.tts_param = {                # 定义待转文字及合成的参数
            'text': '你好，我是鲁班',
            'vcn': 'qige',
            'speed': 50,
            'pitch': 5,
            'volume': 20
        }

    def tts(self,text):
        self.tts_param["text"]=text

 rospy.wait_for_service("/aiui/text_to_speak_multiple_options", timeout=2)   # 等待服务可用。超时时间这里设置为 2s，默认会一直等待，超时会抛出 rospy.ROSException 异常
        tts_client = rospy.ServiceProxy("/aiui/text_to_speak_multiple_options", textToSpeakMultipleOptions)     # 创建 ros 服务客户端
        tts_client(self.tts_param['text'], self.tts_param['vcn'], self.tts_param['speed'], self.tts_param['pitch'], self.tts_param['volume'])
      # 客户端发起请求，参数与该服务的类型定义一一对应
```

这段代码的作用是创建一个T2S(Text-to-Speech)类的实例，该实例包含了待转换的文本和合成参数。

通过调用tts方法，可以动态修改待转换的文本。

然后，使用ROS的wait_for_service函数等待名为 `"/aiui/text_to_speak_multiple_options"` 的服务可用，并创建一个ROS服务代理 `tts_client` 。

最后，通过调用tts_client服务代理并传递相应的参数，向 `"/aiui/text_to_speak_multiple_options"` 服务发起请求，实现文本到语音的转换。

自写museum类

初始化

```
def __init__(self):
        # 0:待选择 1:导览和导游 2:chatGPT
        self.mode=0

        # 解说词
        self.t1="第一件是《星夜》。梵高的经典之作，这幅画充满了梦幻般的色彩和运动感，描绘了一个星空下的风景，令人陶醉其中。"
        self.t2="第二件是《蓝色时期》。毕加索的代表作之一，这幅画以蓝色调为主，表达了艺术家内心深处的孤独和忧伤。"
        self.t3="第三件是《蒙克的呐喊》。爱德华·蒙克的标志性作品，表现了一个扭曲而恐怖的人物尖叫的场景，彰显了内心的痛苦和绝望"

        # 用于和chatGPT交互，初始化socket，连接到我的笔记本电脑
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

1. `self.mode=0` ：这是一个类成员变量，用于表示当前模式。0代表待选择状态，1代表导览和导游模式，2代表chatGPT模式。

2. `self.t1`，`self.t2`，`self.t3`：这些变量包含了一些解说词的文本。它们描述了三幅艺术作品，《星夜》、《蓝色时期》和《蒙克的呐喊》。每个变量存储了一段文本描述。
3. `self.sock`:代码创建了一个套接字（socket）对象，用于与我的笔记本电脑进行交互。它使用了TCP/IP协议（`socket.AF_INET`）并采用流式传输（`socket.SOCK_STREAM`）。该套接字将连接到笔记本电脑的IP地址。

connect_socket函数

```
def connect_socket(self):
    # 创建 socket 对象
    host = '192.168.163.1'
    port = 20055
    # 连接服务，指定主机和端口
    self.sock.connect((host, port))

    # 检测socket连接
    info=self.sock.recv(1024).decode('utf-8')
    rospy.loginfo(info)
```

`connect_socket`的方法用于建立套接字连接并检测连接状态。

1. `host = '192.168.163.1'`：将目标主机的IP地址存储在`host`变量中。在这个例子中，目标主机的IP地址是`192.168.163.1`。
2. `port = 20055`：将目标主机的端口号存储在`port`变量中。在这个例子中，目标主机的端口号是`20055`。
3. `self.sock.connect((host, port))`：创建一个套接字对象，并通过`connect`方法连接到指定的主机和端口。在这个例子中，通过`self.sock`套接字对象连接到`192.168.163.1`的`20055`端口。
4. `info=self.sock.recv(1024).decode('utf-8')`：使用套接字对象`self.sock`接收来自服务器的响应数据。
   `self.sock.recv(1024)`从套接字接收最多1024个字节的数据，并使用UTF-8解码为字符串格式。
5. `rospy.loginfo(info)`：记录连接状态信息。在这个例子中，使用`rospy.loginfo`方法将接收到的连接状态信息记录下来。

总之，`connect_socket`方法创建套接字连接并检测连接状态。方法内部通过套接字对象实现与目标主机的连接，然后接收服务器的连接状态信息并记录下来。

待选择模式

```
if self.mode==0:
    # 待选择模式
    if "智能" in text:
        # 进入chatGPT模式
        self.mode=2
        reply="好的，正在接入"

    elif "导览" not in text and "导游"not in text:
        # 未识别到导览和导游
        reply="抱歉，我不太理解，请再说一次"

    else:
        # 进入导览模式
        reply="好的，欢迎来到2020212185的展馆，这里有三幅精选艺术作品：《星夜》、《蓝色时期》和《蒙克的呐喊》。我们从哪个展品开始？"
        self.mode=1
```

使用一个条件语句，根据`self.mode`的值执行不同的逻辑。

- 如果`self.mode`等于`0`，表示当前处于**待选择模式**。
  - 如果用户的输入中包含"智能"，则将模式切换为chatGPT模式（`self.mode`设置为2），并回复"**好的，正在接入**"。
  - 如果用户的输入中既不包含"导览"也不包含"导游"，则回复"**抱歉，我不太理解，请再说一次**"。
  - 如果用户的输入中包含"导览"或"导游"，则将模式切换为导览模式（`self.mode`设置为1），回复"**好的，欢迎来到2020212185的展馆，这里有三幅精选艺术作品：《星夜》、《蓝色时期》和《蒙克的呐喊》。我们从哪个展品开始？**"。

导览模式

```
elif self.mode==1:
    # 导览模式
    reply="好的，我们开始导览，"
    s1=self.t1
    s2=self.t2
    s3=self.t3

    order=[] # 讲解顺序
    if "1" in text or "一"in text or "星夜" in text:
        order=[s1,s2,s3]
    elif "2" in text or "二"in text or "蓝色时期" in text:
        order=[s2,s1,s3]
    elif "3" in text or "三"in text or "蒙克的呐喊" in text:
        order=[s3,s1,s2]
    else:
        order=[s1,s2,s3]

        for stuff in order:
            reply=reply+stuff

            reply=reply+",好的，导览完成，谢谢"
            self.mode=0
```

如果 `self.mode` 等于1，表示当前处于**导览模式**。

- 回复"**好的，我们开始导览，**"。
- 初始化变量 `s1` 、 `s2` 和 `s3` ，它们分别存储三个艺术作品的描述文本。
- 初始化 `order` 列表，用于**存储讲解的顺序**。根据用户的输入，确定讲解顺序：
  - 如果用户的输入包含"1"、"一"或"星夜"，则讲解顺序为 `s1` 、 `s2` 、 `s3` 。
  - 如果用户的输入包含"2"、"二"或"蓝色时期"，则讲解顺序为 `s2` 、 `s1` 、 `s3` 。

    - 如果用户的输入包含"3"、"三"或"蒙克的呐喊"，则讲解顺序为 `s3`、`s1`、`s2`。
    - 如果用户的输入不包含以上关键词，则讲解顺序为 `s1`、`s2`、`s3`。
- 然后，根据确定的讲解顺序进行导览：
    - 对于每个艺术作品的描述文本，在回复中加入该文本（`reply=reply+stuff`）。
    - 最后，回复"好的，导览完成，谢谢"，并将模式切换回**待选择模式**（`self.mode` 设置为`0`）。

chatgpt模式

```
else:
    # chatGPT模式
    reply=self.intell_reply(text)
    rospy.loginfo("GPT已回复! ")
```

如果 `self.mode` 等于2，表示当前处于**导览模式**。

- 调用 `self.intell_reply` ,获取 `text` 的chatGPT回复
- 使用 `rospy.loginfo` 打印"GPT已回复! "

intell_reply()函数

```
def intell_reply(self,text):
    # 发送数据
    self.sock.send(text.encode('utf-8'))
    # 接收数据
    gpt_reply=self.sock.recv(1024).decode('utf-8')

    return gpt_reply
```

`intell_reply` 方法，用于与chatGPT模型进行交互。以下是代码的解释：

1. `self.sock.send(text.encode('utf-8'))`：通过套接字对象 `self.sock`，将文本编码为UTF-8格式并发送给我的笔记本电脑。`text.encode('utf-8')` 将输入文本转换为**字节流**并使用 `UTF-8` 编码。

2. `gpt_reply = self.sock.recv(1024).decode('utf-8')`：使用套接字对象`self.sock`接收来自我的笔记本电脑的响应数据。`self.sock.recv(1024)`从套接字接收最多1024个字节的数据，并使用UTF-8解码为字符串格式。
3. `return gpt_reply`：返回chatGPT模型的响应作为输出。

`intell_reply`方法用于向chatGPT模型发送文本并接收其响应。方法内部通过套接字对象实现数据的发送和接收。发送时，将输入文本编码为字节流并发送；接收时，从套接字接收数据，并解码为字符串。最后，将chatGPT模型的响应作为输出返回。

运行效果见视频