任务三 最终导游模块

综合应用所掌握的信息，设计导游机器人，不限制开发环境。

测试环境：至少包含3~6个目标展品，展品内容不限制（可利用任务二作业）

能够接收用户询问，想看哪幅图片/展品。理解用户意图。对展品介绍顺序灵活实际的规划。（10分）

能够针对展品回答用户提问。随时可以被打断。对话内容自然丰富。（10分）

（可利用任务一作业）

能够在给定场地环境中，引领用户走到图片附近，抬手指向展品。（20分）

能够通过传感器（如声呐及脚前方的按钮）躲避障碍物防止跌倒。（20分）（新增任务）

能够通过视觉分辨不同用户、不同展品。（20分）

能够通过检测声音传来的方向，将头转向用户，讲解时面向用户。（20分）

· 作业提交：　· 时间：6月9日

· 提交内容：　1、代码，代码需能够执行 2、实验报告 3、功能、展示所需的录像或者录音

04智能机器人导游系统

一、实验目的与实验要求

设计博物馆导游机器人，能够在3-6个展品间进行导览。

1. 规划导览：
    a. 询问用户、并理解用户意图
    b. 根据用户要求规划展品的导览顺序
2. 展品导航：参照任务1的要求
    a. 走到目标展品位置，要求机器人具备避障能力
    b. 手指向目标展品
    c. 导航过程中可以打断，或者更改目标
3. 语音介绍展品：参照任务2的要求

a. 理解并回答用户问题

b. 对话过程支持打断

4. 视觉识别：

a. 识别用户并跟踪

b. 识别展品，识别结果应该与实物、介绍内容匹配

5. 其他功能：

a. 头转向声源方向，头面向用户进行讲解

任务分工

人员	学号	分工
佟欣阳	2020212097	目标检测、代码整合、语音识别、视频剪辑
林亦旸	2020212142	路径规划、目标检测、语音识别
粟思悦	2020212222	目标检测、运动控制、语音识别
李云川	2020212234	路径规划、运动控制、报告撰写
尹真真	2020212068	协助代码测试，报告撰写
黄莫寒	2020212033	代码调试分析，报告撰写

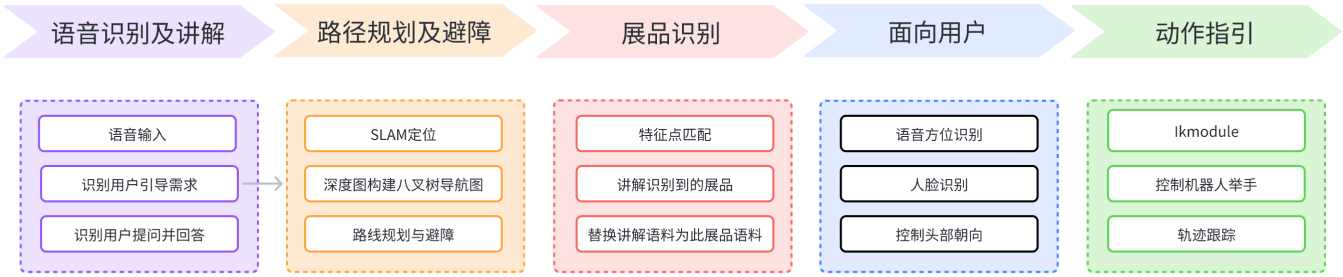
二、模块介绍

2.1 模块实现情况

本实验实现了以下模块：

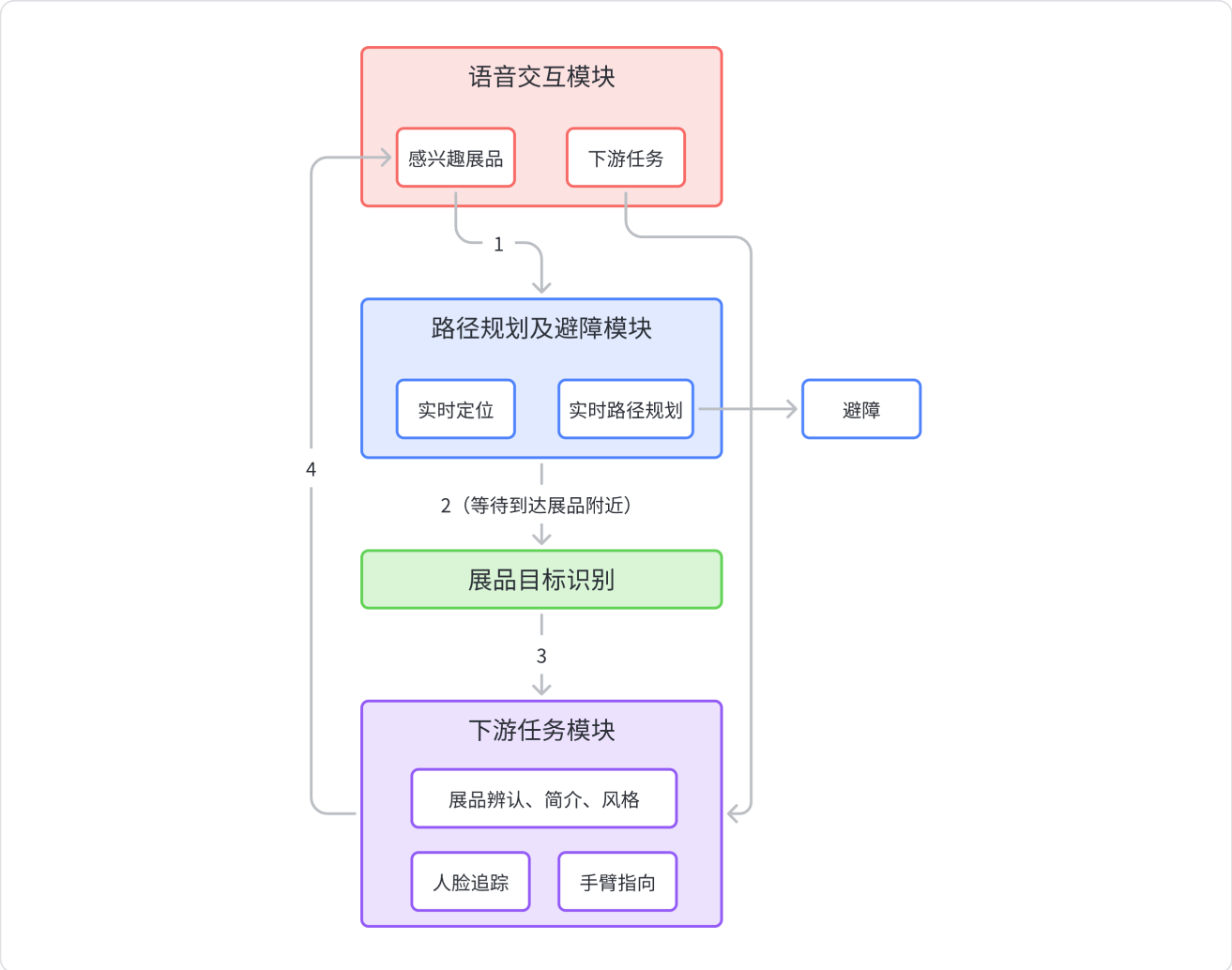
1. 语言识别及讲解模块。此模块用于接收用户对于展品的提问和引导需求，以调用后续模块进行展品讲解或进行展位引导。
2. 路径规划及避障模块。此模块使用视觉SLAM算法对机器人定位，并调用D435i相机的深度图对场景建立八叉树导航图。模块基于视觉定位和导航图，完成了机器人到展位的路径规划和避障功能。
3. 展品识别模块。此模块通过特征匹配，识别当前所在的展品。以调用对应的讲解语料。
4. 面向用户模块。此模块通过人脸检测和声音位置方向，让机器人在讲解或回答问题过程中始终面向用户。
5. 动作指引模块。此模块让机器人跟着规划的轨迹带领用户走到展品面前，并控制机器人抬手指向展品，实现动作指引。

模块间的使用关系如下图所示：



2.2 运行流程

用户向机器人发出语音指令，说明用户想要参观的展品。机器人解析用户感兴趣展品后开始引导用户前往。此时机器人进行实时定位与实时路径规划。到达展品附近后，机器人进行展品识别，确认展品信息，抬手指示到达展品附近。之后用户可以开始提问本展品信息，包括简介、风格等，也可以要求机器人前往下一个展品。

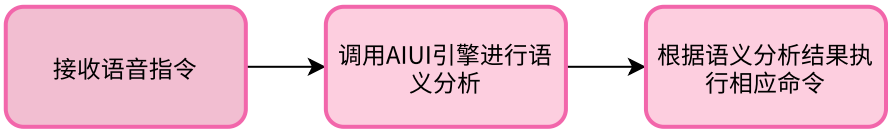


三、系统实现方法

3.1 语音识别及讲解

3.1.1 功能简述

这一部分实现了一个基于AIUI技术的语音识别模块。主要流程如下图所示：



该模块主要由voice_control.py文件实现。下面对代码的不同模块进行详细介绍：

1. import模块：
   - rospy和rospkg：ROS相关模块，用来实例化ROS节点和获取ROS包路径。
   - os、sys和subprocess：Python自带的系统模块，用于调用操作系统的命令和运行子进程。
   - std_msgs：ROS标准消息格式，用于定义消息类型。
   - time：提供sleep函数用于暂停程序。
   - random：提供生成随机数的函数。
2. motion.motionControl模块：
   - ResetBodyhub：机器人姿态初始化，重置机器人的身体状态。
   - SetBodyhubTo_setStatus：设置机器人的身体状态。
3. AIUI_node模块：
   - EventListener类：继承自AIUI的事件监听器，重写了语义结果事件和语音合成事件的回调函数。根据语义分析结果执行相应的命令或回答用户的问题。
   - AIUINode类：继承自ROS节点，用于启动AIUI引擎，并将EventListner对象作为参数传递给AIUI引擎，注册回调函数。
4. VoiceControlConfig.json：
   - 本地自定义技能的配置文件，包含每个技能的语义解析和回答等内容。

3.1.2 关键算法

在本模块中，语音识别与响应主要通过EventListener类实现。

Eventlistener类是一个继承了AIUI Python SDK中的EventListener基类并进行了重构的类。其中包含了主要的语音交互逻辑，定义了NLP和TTS事件的回调函数，负责将语音识别结果转化为对应的回答文本并经过TTS转为语音播放，同时还可执行一些指令（如启动运行某个节点或停止某个节点）。

这个类具体实现了以下方法和属性：

- 属性:
  - `ex_status`：当前所处的展品状态。
  - `question_num`：当前展品问答中问题的编号。
  - `question_list`：保存一些问答问题的列表。
  - `museum_ex`：保存展品名称的列表。
  - `next`：下一个展品的命令文本。
  - `museum_num`：当前正在浏览的展品在展品名称列表中的位置。
- 方法:
  - `__init__(self, skills_dict, debug=False)`：类的初始化方法，设置自定义技能和调试标志。
  - `EventNLP(self, event)`：重写父类的语义结果事件回调函数，获取识别的语义结果并根据结果进行回答和执行指令。

    在这一函数中，对用户发起的指令分为了三大类：`展览馆相关`、`前进相关`、`其他`

    如果匹配到了展览馆的相关命令，如**"这是什么"**、**展览名称**、**问题**等，则进行相应的处理和回答。其中，如果是"这是什么"命令，则需要订阅话题"/detected_exhibition"获取当前展览名称，并根据展览名称执行相应操作；如果是问题，则根据问题编号选择对应的回答。

    如果匹配到了前往展位的命令，则随机从预设的回答中选择一个进行回答。如果该命令中包含cmd_type类型，则需要根据类型进行相应操作，如运行指定路径的python程序。

    如果是其他常规命令，则也根据预设的回答随机选择一个进行回答，并根据是否包含cmd_type类型进行相应操作。
  - `EventTTS(self, event)`：重写父类文本转语音事件回调函数，获取语音数据并播放。具体代码思路如下：
    - 从event中获取一段语音数据，存储在buffer中。
    - 如果正在播放语音，则停止当前的播放。然后将语音数据追加写入一个文件中。
    - 判断是否接收完成语音合成的结果，如果接收完整，则开始播放语音文件。

    其中，语音数据保存的位置为**self.tts_pcm_file**，在实现中是通过文件读写来实现的。播放语音使用了**self.player.play**方法，该方法主要利用了pygame库来实现语音的播放。
  - `detect_callback(self,data)`：展品识别的回调函数。

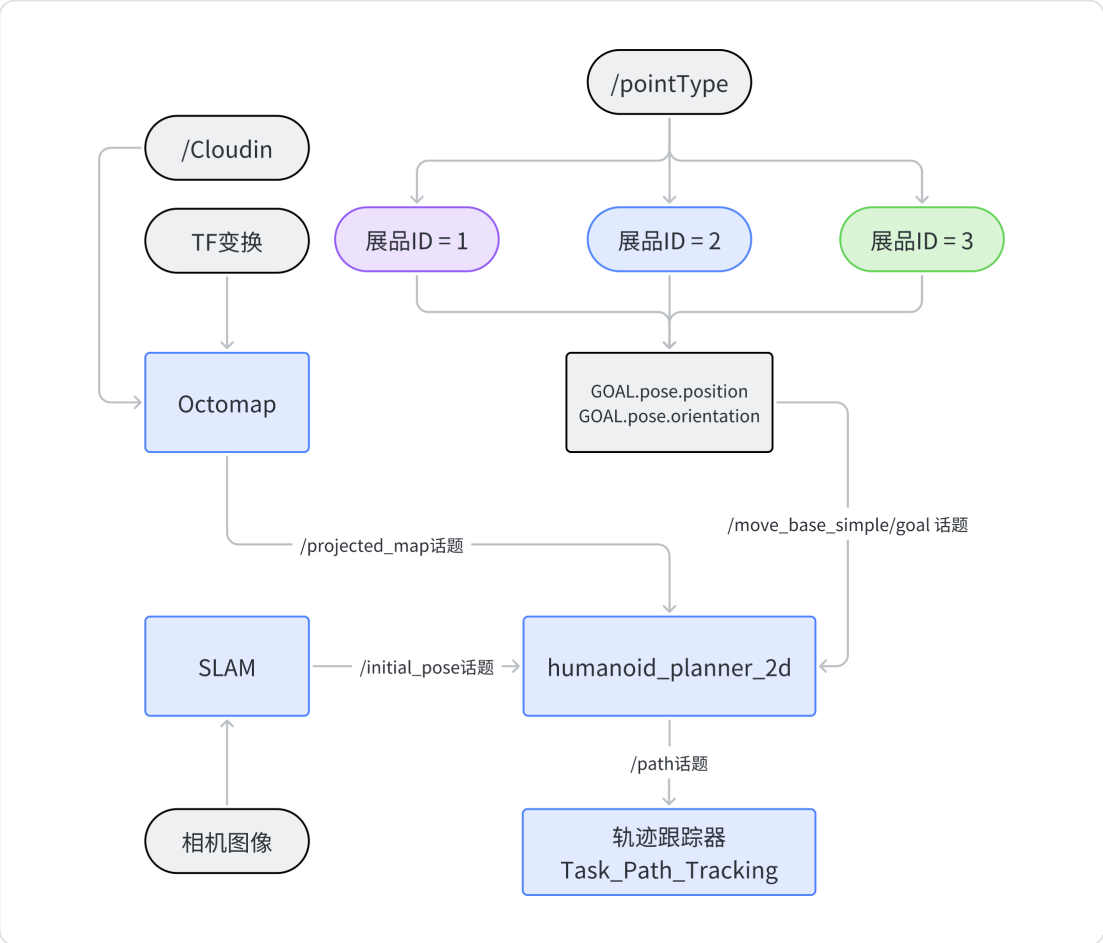
3.2 路径规划及避障

3.2.1 功能简述

这一模块以用户语音指定的目标展品作为输入，自动解析得到展品对应的目标坐标和实时规划路径，将局部的目标坐标输出到下层的运动控制模块。

3.2.2 关键算法

模块实现的依赖关系图：

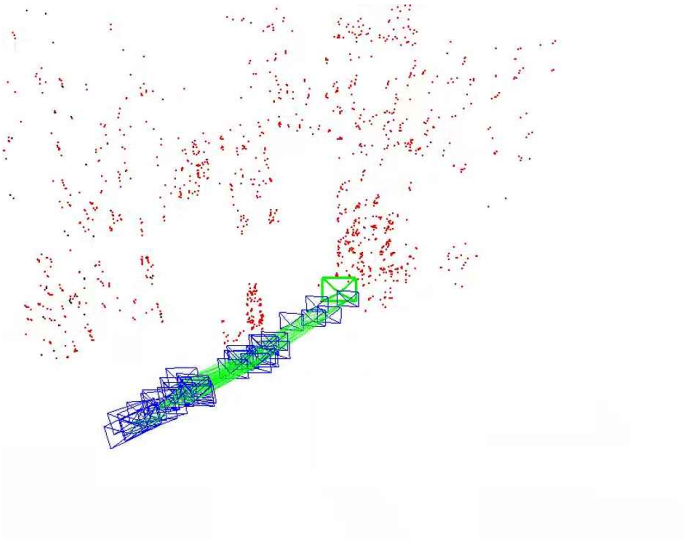


自定义一个服务端 `pointType` ，用来指定当前目标展品的id号及位置

```
#客户端
s = rospy.Service("/pointType",SrvTLSuint,callback)

#服务端
rospy.wait_for_service("/pointType")
changeid = rospy.ServiceProxy("/pointType",SrvTLSuint)
changeid(1)

#目标坐标赋值
if point == 1:
    GOAL.pose.position.x = 0.2
```

```
    GOAL.pose.position.y = -1
    GOAL.pose.position.z = 0
    GOAL.pose.orientation.x = 0
    GOAL.pose.orientation.y = 0
    GOAL.pose.orientation.z = -0.38
    GOAL.pose.orientation.w = 0.92
```

运行ORB_SLAM2，得到机器人的实时定位：



同时启动Octomap，用稠密点云进行建图。为了使地图更具实用性，增加了按一定频率更新地图的功能，移动物体的占据栅格不会一直保留，进而可以用于路径规划。

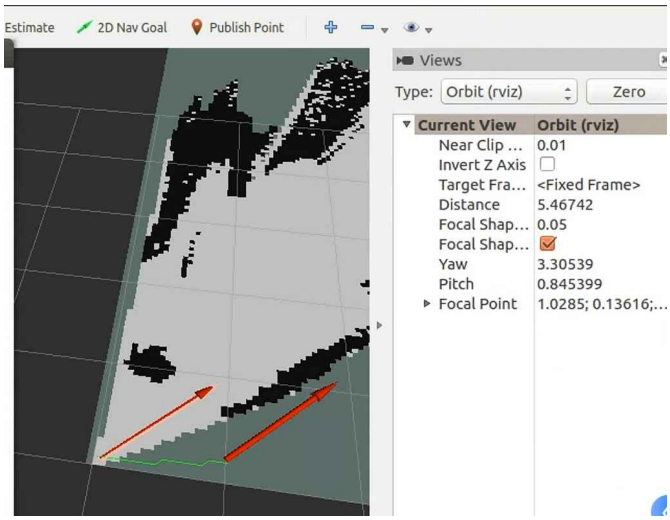
使用 `humanoid_navigation` 功能包进行轨迹规划和避障的开发。其工作原理为：收到地图、机器人坐标、目标点后，使用全局路径规划SBPL软件搜索规划点。地图、机器人坐标、目标点更新后会进行重规划，这样就可以在不预先建图的情况下完成规划功能。

对于避障功能的实现，可以利用膨胀八叉树地图的方法生成机器人的安全工作空间。在安全工作空间中进行全局路径规划。这里需要设定机器人半径，将八叉树地图中的障碍区域向外拓展一个机器人半径的大小。对于不能到达或有碰撞风险的目标点，功能包会给出提示。其计算方法如下：

```
int GridLocal[map_->getInfo().width][map_->getInfo().height] = {0};
  for(unsigned int j = 0; j < map_->getInfo().height; ++j)
    for(unsigned int i = 0; i < map_->getInfo().width; ++i)
        GridLocal[i][j]=0;

  for(unsigned int j = 0; j < map_->getInfo().height; ++j){
    for(unsigned int i = 0; i < map_->getInfo().width; ++i){
      if (map_->isOccupiedAtCell(i,j)) {
        GridLocal[i][j]=OBSTACLE_COST;
        // 计算阴影（弱栅格）
        for(int k = 1; k <= SHADOW_RADIUS; k++) {
          if((i-k >= 0) && (i-k < map_->getInfo().width) && (map_->isOccupiedAtC
```

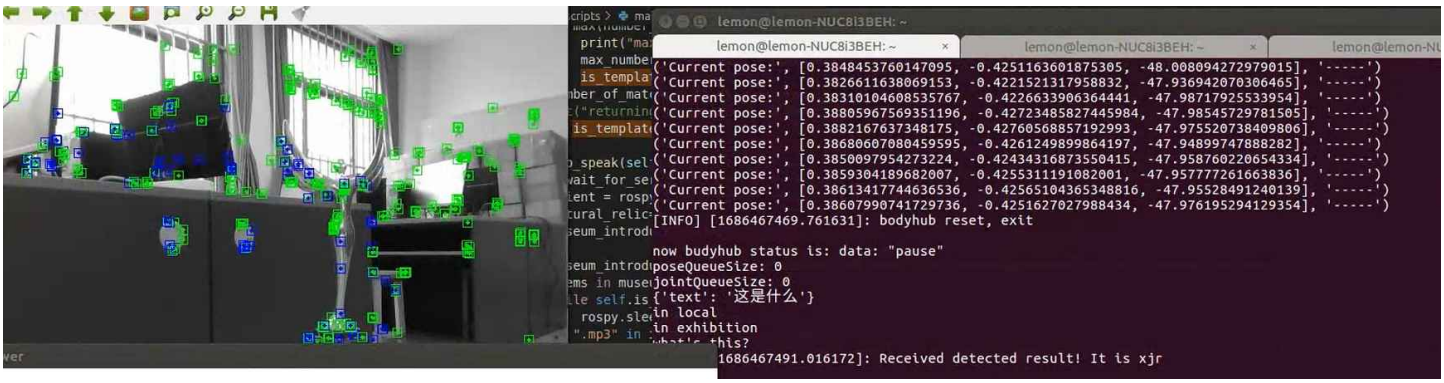
```
            int temp = GridLocal[i-k][j];
            temp = SHADOW_RADIUS - k;
            if(GridLocal[i-k][j] < temp)
                GridLocal[i-k][j] = temp;
          }

          if((i+k >= 0) && (i+k < map_->getInfo().width) && (map_->isOccupiedAtC
            int temp = GridLocal[i+k][j];
            temp = SHADOW_RADIUS - k;
            if(GridLocal[i+k][j] < temp)
                GridLocal[i+k][j] = temp;
          }

          if((j-k >= 0) && (j-k < map_->getInfo().width) && (map_->isOccupiedAtC
            int temp = GridLocal[i][j-k];
            temp = SHADOW_RADIUS - k;
            if(GridLocal[i][j-k] < temp)
                GridLocal[i][j-k] = temp;
          }

          if((j+k >= 0) && (j+k < map_->getInfo().width) && (map_->isOccupiedAtC
            int temp = GridLocal[i][j+k];
            temp = SHADOW_RADIUS - k;
            if(GridLocal[i][j+k] < temp)
                GridLocal[i][j+k] = temp;
          }
        }

      }
    }
  }

  for(unsigned int j = 0; j < map_->getInfo().height; ++j){
    for(unsigned int i = 0; i < map_->getInfo().width; ++i){
      planner_environment_->UpdateCost(i,j,GridLocal[i][j]);
    }
  }
```

启动ROS包humanoid_planner_2d，得到前往展品的最佳全局路径，以 `/path` 话题发布，地图和路径的可视化效果如下：



由于采用**全局**和**实时**的基于视觉的路径规划，避障功能在不断更新计算最佳全局路径的过程中已经实现。

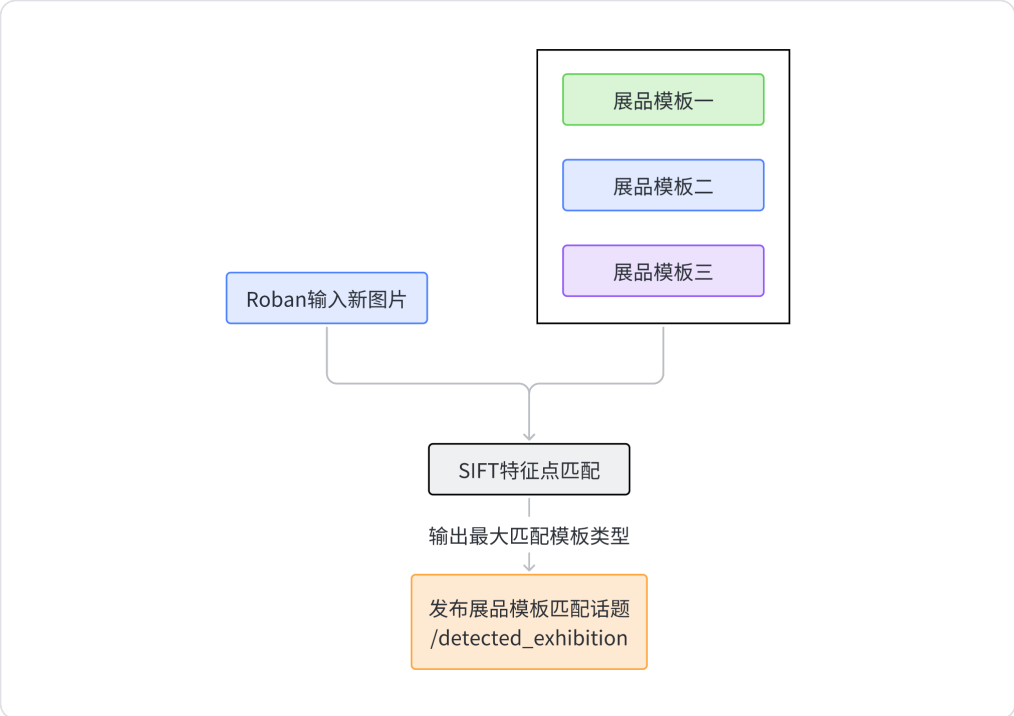
3.3 展品识别



3.3.1 功能简述

此模块用于运行时的展品视觉识别。它使用预先准备的展品模板图片，在运行时与相机图片匹配SIFT特征，满足匹配点数量大于某个阈值且是各展品匹配中匹配点数量最多的，则匹配判定成功并发布 `/detected_exhibition` 话题。

3.3.2 关键算法



该模块主要定义了FeatureMatch类和MuseumDocent类。前者用于特征匹配，后者综合整体图像识别流程。

1. **FeatureMatch**

这个类实现了基于SIFT算法的图像特征提取与匹配。类初始化时需要传入一个待匹配的图像 `pending_img` 。

其中， `feature_extraction(self, model_img)` 函数用于提取待匹配图像和参考图像的SIFT特征点，并进行特征匹配。函数参数 `model_img` 是参考图像的路径。首先将参考图像转换为灰度图像，然后使用 `cv.xfeatures2d.SIFT_create()` 函数构建了一个SIFT对象 `sift` 。接着使用 `sift.detectAndCompute()` 函数分别提取待匹配图像和参考图像的特征点和特征描述符。这里使用FLANN算法进行匹配，具体实现使用 `cv.FlannBasedMatcher()` 函数构建一个FLANN匹配器 `flann` ，并使用 `flann.knnMatch()` 函数对两幅图像的特征描述符进行匹配。然后根据设定的匹配门限对匹配结果进行筛选得到好的匹配点，并返回匹配点个数。

2. **MuseumDocent**

这个类实现了机器人识别图像全流程。机器人会通过摄像头获取当前画面并进行图像特征提取，在提取过程中会将当前画面与预先存储好的模板画面进行比对，找到最匹配的模板画面。该类中的关键函数结构如下：

a. **get_match_result(self)**：在预设的模板画面中，找到与当前画面最匹配的那一张图片。函数首先初始化了一个列表number_of_matching_to_single_template，存储每个待匹配模板图片与原始图像的匹配数量。然后通过遍历参数 `elements_and_results_of_matching` 字典中的元素和内容，对每个元素的所有模板图片进行匹配，将匹配结果存储到 `number_of_matching_to_single_template` 列表中，并记录最大匹配数量以及所匹配的模板图片索引。最后将匹配最佳的模板图片的索引返回。

b. **text_to_speak(self, cultural_relic)**：将文本信息通过语音合成技术转化为音频文件，并播放出来。

函数的参数 `cultural_relic` 表示文物的索引，如果传入-1，则会停止运行。首先使用 `rospy.wait_for_service` 函数等待/text_to_speak服务可用，并使用 `rospy.ServiceProxy` 函数创建一个服务代理 `tts_client` 。

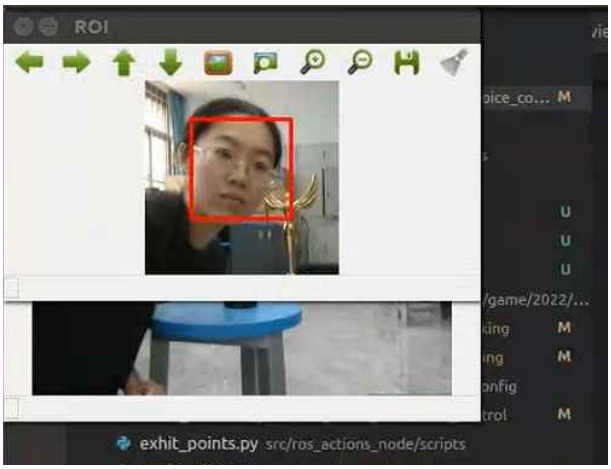
根据传入的文物索引值 `cultural_relic` 从 `self.elements_and_results_of_matching` 字典中获取该文物的介绍文本 `museum_introduce_text` ，并遍历其中的每一条文本。如果文本中包含".mp3"或".wav"，则使用 `subprocess.Popen` 函数以子进程的形式播放音频文件，其中使用 `ffplay` 命令播放音频，并通过参数 `-nodisp` 、 `-loglevel quiet` 和 `-autoexit` 设置不显示播放器窗口、输出日志信息的等级为"quiet"、播放结束后自动退出。

在播放音频的过程中，使用 `player_process.poll` 函数检查子进程是否已经结束，如果未结束则暂停0.1秒后继续检查。如果文本不是音频文件，则使用 `tts_client` 函数将文本转换为语音并播放。播放结束之前，将 `self.is_play_end` 设置为False，表示正在播放中。最后使用while循环等待播放结束，即 `self.is_play_end` 变为True，才会继续执行下一条文本的播放。

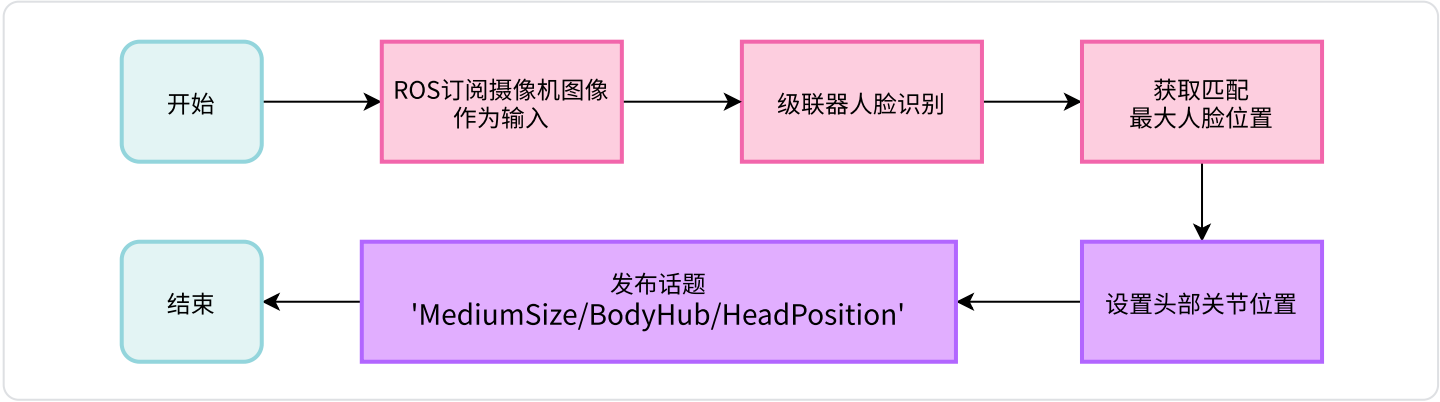
3.4 面向用户

3.4.1 功能简述

面向用户功能实现的功能是人脸跟踪，即通过头部关节控制器控制机器人头部的转动角度，使得机器人始终朝向人脸所在的方向。



3.4.2 算法步骤



人脸检测和跟踪，基于OpenCV库的人脸识别程序，使用了多线程技术，程序通过订阅ROS中的摄像头图像数据，使用级联分类器检测人脸，并锁定最大的人脸，最后结果输出人脸中心位置用于控制头部关节转动。

3.4.3 关键算法

1. **Face_detecter**

这个类实现了机器人人脸识别全流程。机器人会通过摄像头获取当前画面并使用级联器锁定最大人脸，如果没有检测到人脸则进行模板匹配。该类中的关键函数结构如下：

a. **run(self)**：类循环执行 `run` 函数，该函数有双层循环结构。

函数从摄像机图像队列 `QUEUE_IMG` 中获取帧，外循环使用 `cv` 库级联器检测并锁定最大人脸（感兴趣区域）。直到检测到人脸后进入内循环。

内循环对感兴趣区域进行进一步检测，如果没有检测到人脸，则使用 `cv2.matchTemplate` 人脸模板匹配。如果内循环匹配失败，则说明该区域不是人脸，故退出到外循环重新检测感兴趣区域。

```
    def run(self):
        rate = rospy.Rate(100)
        while not rospy.core.is_shutdown_requested() and not self.Face.fou
            # 外循环用于检测人脸并锁定最大的人脸
            time.sleep(0.01)
            if not QUEUE_IMG.empty():
                frame = QUEUE_IMG.get()
            else:
                continue
            self.detectFaceAllSizes(frame)# 检测人脸并找到并锁定最大的人脸

            while not rospy.core.is_shutdown_requested() and self.Face.fou
                # 如果找到人脸并锁定之后，进入内循环
                rate.sleep()
                if not self.Face.face_template.any():
                    continue
                if not QUEUE_IMG.empty():
```

```
                    frame = QUEUE_IMG.get()
                else:
                    continue
                self.detectFaceAroundRoi(frame)# 在感兴趣区域即上一次检测到人
                # 如果没有检测到人脸，则使用模板匹配查找
                if self.Face.template_matching_running:
                    self.detectFacesTemplateMatching(frame)
```

b. **detectFaceAllSizes(self, frame)**：外循环核心函数，检测各种大小的人脸，筛选最大的人脸并存储为模板。

   对于输入的 `frame` 帧，使用 `cv` 库级联器算法 `face_detector.detectMultiScale` 检测人脸区域，对于检测的结果，计算人脸区域大小并筛选最大人脸，最后将人脸区域存入模板中。

c. **detectFaceAroundRoi(self, frame)**：内循环核心函数，在感兴趣区域即上一次检测到人脸的区域附近进行检测。

   方法与外循环类似。

2. **tracking() 函数**

该函数是用于实现人脸追踪的关键函数。其输入是当前人脸在画面上的位置信息 `face_pos`，另外还接受期望人脸位置信息 `desired_pos`，默认为画面中心点(320, 240)。输出是舵机角度执行器位置 `actuator_pos`。

舵机角度通过计算目标位置与当前人脸位置之间的偏差实现。如果偏差超过了预设的阈值，则进入 PID 控制器进行计算并输出控制器的控制量。

3. **Head_Joint_Controller**

这个类用于控制头部关节的转动，使用PID控制保证头部转动的流畅性和准确性。该类的关键类函数如下：

a. **set_head_servo(self, angles)**：通过'MediumSize/BodyHub/HeadPosition'话题发布头部关节动作信息，bodyhub接收到消息后转动头部。

3.5 动作指引

3.5.1 功能简述

让机器人跟着规划的轨迹带领用户走到展品面前，并控制机器人抬手指向展品，实现动作指引。

3.5.2 关键算法

由于路径规划模块是实时更新路径点，因此机器人只需要跟踪路径规划模块发布的轨迹中的下一个点位，就能一路导航到最后的目标点。在这个过程中需要保持机器人的朝向沿着轨迹方向，以保证机器

人能看到前方障碍物。

需要注意的是，轨迹规划模块并不会发布路径点的偏航角，模块固定发布的是终点的偏航角。因此对于偏航角的控制需要做特殊处理。当发布的轨迹点序列只有一个点时，认为机器人在走向终点，此时跟踪终点偏航角。其余情况下，计算当前位置和目标运动点的连线的角度，让机器人朝向该角度行进。

抬手这一部分主要基于IkmoduleSim类实现。先调用toInitPoses方法进行机器人位姿初始化，如果初始化成功，则调用body_motion方法对机器人的某一部分进行控制。该方法有三个参数，第一个参数为机器人的末端位置，第二个参数为设定的末端的位姿值，第三个参数为插值次数。在本实验的实现中，由于只需要调动机器人抬手指向展品，因此对其右胳膊进行控制，并设定好了其末端位姿。

```
if self.toInitPoses():
    self.body_motion([C.RArm_x,C.RArm_z], [0.3,0.4])
    time.sleep(2)
    self.reset()
```

四、运行方式

```
#启动roban节点
./start.sh

#开语音识别节点，等待语音输入
python ros_demo_case/voice_control/voice_control.py


#开detect结果topic,在"/detected_exhibition"话题下发布String类型展品检测结果
python src/ros_museum_docent/scripts/main.py

#开目标点"/pointType"服务，进行SrvTLSuint类型交互
python src/ros_actions_node/scripts/exhibits_points.py

# 开SLAM节点
先进行nav_dev的source
rosrun SLAM RGBD true false
roslaunch humanoid_planner_2d humanoid_planner_2d.launch
roslaunch SLAM octo.launch
```

一些用到的服务

```
rosservice call /MediumSize/ActPackageExec/StateJump 6 reset  #更改6号ID下状态机状
```

```
2 rosservice call /MediumSize/BodyHub/GetMasterID  #查看控制的ID号
```

五、总结

5.1 优势

1. **路径规划**

给定展品的位置信息，机器人可以自主规划路径行进到展品位置。当展品位置发生移动时只需修改展品的位置信息，适应性强，方便移植。

2. **避障功能**

在机器人行进到展品位置的过程中，如果遇到障碍物，机器人能够检测到障碍并重新规划路径。保证了机器人在运行过程中的安全性，避免与障碍物相撞造成不必要的损失，且灵活性好，当场馆内物品位置发生变化时不必重新修改代码逻辑。

3. **人脸识别和语音交互**

机器人可通过人脸识别技术检测到用户所在位置，并在讲解的过程中面向用户。还可以根据语音的方向将头转向用户，增强了机器人的交互性，提高用户体验。

4. **支持打断**

在机器人行进以及讲解的过程中，通过语音指令可以打断机器人当前的行为，支持更改目的地、更换问题等功能，使得机器人在运行的过程中灵活性更好，更方便进行导览。

5.2 问题与不足

1. **语音模块识别不准确**

对于语速过快以及吐字不清的情况，语音模块会对结果识别的不够准确。对于语音模块识别不准确的问题，可能需要进一步调整其参数和优化模型，以提高准确性。此外，还可以考虑增加语音训练数据集，以提高模型的泛化能力。

2. **镜头晃动导致路径规划不准**

路径规划在进行时容易受到镜头晃动等因素的影响，导致点飘的情况。为解决这个问题，可以考虑改进路径规划算法，采用更加鲁棒的方法来消除干扰，或者在设备上采用更稳定的支架，以保证镜头稳定性。

3. **状态机修改困难**

关于状态机修改困难的问题，可以考虑将原有的状态机分解成更小的模块，分别进行设计和编写，并在必要时进行接口的统一。同时，在开发过程中可以采用模块化的方式组织代码，以便于后续的维护和扩展。

5.3 实验总结

我们使用ROS作为开发环境，使用语音识别和语音合成技术实现与用户的交互，使用机器人视觉技术实现人脸跟踪和展品识别，使用SLAM技术实现导航和障碍物避免和导航。

在实验中，我们将机器人放置在展馆中，用户可以通过语音询问机器人想看哪幅图片或展品，机器人会根据用户的意图规划展品介绍顺序，并引领用户走到展品附近，抬手指向展品，同时通过语音回答用户的问题。在导航过程中，机器人会通过视觉技术识别展品和用户，并通过传感器避免障碍物和跌倒。

通过实验，我们发现机器人的导游功能可以帮助用户更好地了解展品，同时也提高了机器人的交互性和智能化程度。但是，在实验中我们也发现了一些问题，比如机器人的语音识别还需要进一步优化，机器人的导航稳定性还需要提高等。我们将继续改进机器人的功能和性能，以提高机器人的实用性。