**【仓库分拣机器人】**

**软件设计说明书**

**【SDD203】**

**【version 2.0】**

分工说明

|  |  |  |
| --- | --- | --- |
| 小组名称 | White Givers | |
| 学号 | 姓名 | 本文档中主要承担的工作内容 |
| 17373342 | 邓力友 | 主编，前两节、体系结构设计、详细设计、需求可追踪性说明编写 |
| 17373344 | 杨华晟 | 接口设计、详细设计编写 |
| 17373332 | 刘冬 | 应用场景、数据库设计、详细设计编写 |
| 77086002 | 萨扎尔 | 运行与开发环境编写 |

版本变更历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 主要编制人 | 审核人 | 版本说明 |
| 0.4 | 4.19 | 邓力友 | 邓力友 | 添加体系结构设计内容 |
| 0.7 | 4.19 | 杨华晟 | 邓力友 | 添加接口设计、详细设计移动控制模块内容 |
| 0.9 | 4.19 | 刘冬 | 邓力友 | 添加数据库设计内容 |
| 0.95 | 4.20 | 萨扎尔 | 邓力友 | 添加运行与开发环境内容 |
| 1.0 | 4.20 | 邓力友 | 杨华晟 | 第一版 |
| 1.1 | 4.22 | 刘冬 | 邓力友 | 修改详细设计类图，应用需求 |
| 2.0 | 4.23 | 邓力友 | 杨华晟 | 修改需求追踪性说明内容 |
| 3.0 | 5.21 | 邓力友 | 杨华晟 | 添加各模块用例图和说明 |

目录

[1 范围 4](#_Toc38579902)

[1.1 项目概述 4](#_Toc38579903)

[1.1.1 系统背景 4](#_Toc38579904)

[1.1.2 主要功能性需求 4](#_Toc38579905)

[1.1.3 非功能性需求 5](#_Toc38579906)

[1.1.4 应用场景 5](#_Toc38579907)

[1.2 文档概述 5](#_Toc38579908)

[1.2.1 文档用途 5](#_Toc38579909)

[1.2.2 内容组织 5](#_Toc38579910)

[1.3 术语和缩略词 5](#_Toc38579911)

[1.4 引用文档 6](#_Toc38579912)

[2. 需求概述 6](#_Toc38579913)

[2.1 业务需求 6](#_Toc38579914)

[2.1.1 障碍识别与路径规划 6](#_Toc38579915)

[2.1.2 目标物体识别与抓取 6](#_Toc38579916)

[2.2 数据需求 6](#_Toc38579917)

[2.3 功能需求 7](#_Toc38579918)

[2.3.1 语音识别 7](#_Toc38579919)

[2.3.2 地图建立 8](#_Toc38579920)

[2.3.3 路径规划 8](#_Toc38579921)

[2.3.4 目标物体识别抓取 9](#_Toc38579922)

[2.4 非功能需求 9](#_Toc38579923)

[2.4.1 性能指标 9](#_Toc38579924)

[2.4.2 系统可用性 10](#_Toc38579925)

[2.4.3 系统可移植性 10](#_Toc38579926)

[2.4.4 系统安全性 10](#_Toc38579927)

[2.4.5 系统效率 11](#_Toc38579928)

[3. 数据库设计 11](#_Toc38579929)

[3.1 E-R图 11](#_Toc38579930)

[3.2 数据库关系模式 11](#_Toc38579931)

[3.2.1 用户信息(Users) 12](#_Toc38579932)

[3.2.2 地图信息(Maps) 12](#_Toc38579933)

[3.2.3 路径信息(Routes) 12](#_Toc38579934)

[3.2.4 导航信息(Navigation) 13](#_Toc38579935)

[3.2.5 语音关键词(Characteristic Words) 13](#_Toc38579936)

[4. 体系结构设计 14](#_Toc38579937)

[4.1 总体结构 14](#_Toc38579938)

[4.2 软件体系结构 15](#_Toc38579939)

[4.3 硬件体系结构 17](#_Toc38579940)

[4.4 技术体系结构 18](#_Toc38579941)

[4.5 支撑体系结构 18](#_Toc38579942)

[5. 接口设计 19](#_Toc38579943)

[5.2.1 运动底盘 22](#_Toc38579944)

[6. 详细设计 26](#_Toc38579945)

[6.1 移动控制模块 26](#_Toc38579946)

[6.1.1 MoveControl类 27](#_Toc38579947)

[6.1.2 ObstacleDetect类 27](#_Toc38579948)

[6.1.3 Radar类 27](#_Toc38579949)

[6.2 地图构建模块 28](#_Toc38579950)

[6.2.1 Map Control类 28](#_Toc38579951)

[6.2.2 Map类 28](#_Toc38579952)

[6.2.3 Radar类 29](#_Toc38579953)

[6.2.4 Exception类 29](#_Toc38579954)

[6.3 自主巡航模块 29](#_Toc38579955)

[6.3.1 Path Control类 29](#_Toc38579956)

[6.3.2 Path Move Control类 30](#_Toc38579957)

[6.3.3 Path类 30](#_Toc38579958)

[6.4 机械臂控制模块 30](#_Toc38579959)

[6.5 图像识别模块 31](#_Toc38579960)

[6.6 目标物抓取模块 32](#_Toc38579961)

[7. 运行与开发环境 32](#_Toc38579962)

[7.1 运行环境 32](#_Toc38579963)

[7.2 开发环境 33](#_Toc38579964)

[8. 需求可追踪性说明 34](#_Toc38579965)

[8.1 功能需求 34](#_Toc38579966)

[8.1.1 语音识别 34](#_Toc38579967)

[8.1.2 地图建立 34](#_Toc38579968)

[8.1.3 路径规划 35](#_Toc38579969)

[8.1.4 目标物体识别抓取 36](#_Toc38579970)

[8.2 非功能需求 36](#_Toc38579971)

[8.2.1 性能指标 36](#_Toc38579972)

[8.2.2 系统可用性 37](#_Toc38579973)

[8.2.3 系统可移植性 37](#_Toc38579974)

[8.2.4 系统安全性 37](#_Toc38579975)

[8.2.5 系统效率 38](#_Toc38579976)

# 1 范围

## 项目概述

### 系统背景

目前，嵌入式系统已经深入到我们生活中的各个方面。小到录音笔，mp3，

大到汽车，飞机，上面都搭载着各式各样的嵌入式系统。嵌入式系统因为其适应硬件的独特软件系统，在功耗、成本、体积、可靠性、处理能力上都有着过人的表现而在各种工作类型清晰的设备上为人所青睐。

机器人同样也收到人们的欢迎。现在各式各样的机器人层出不穷，在不同的岗位上帮助人们完成着一些重复性高，危险性高，技术要求精细的工作。而机器人的自主导航是一大类机器人的核心问题。已经日趋成熟的扫地机器人，擦玻璃机器人，包括正在研究的自动驾驶汽车，对于路径的识别与规划都是重中之重。

在此基础上，我们将对有着嵌入式系统的机器人进行编码与设计。

### 主要功能性需求

我们设计的机器人的主要功能需求有：

* 机器人的主动控制 ；
* 机器人利用传感器实时建立环境地图 ；
* 机器人根据地图和自身的位置信息实现动态路径规划及导航控制；
* 物体的识别与自动抓取；

### 非功能性需求

* 性能需求：响应时间短、短时间内吞吐量大、资源利用率高；
* 安全性：保密性、防泄漏、权限控制、防攻击；
* 可维护性与可扩展性：模块性、可复用性、易分析性；
* 可靠性：易恢复性、容错性、成熟性；
* 易用性：易学习性、易操作性、用户错误防御机制、用户界面美观；

### 应用场景

本机器人计划应用场景为仓库。快递物流量如此之大的今天，货物的分拣工作必不可少，但始终是由人类完成。我们希望设计出的机器人能够自主在待分拣货物堆中巡航，并最终通过对货物体积的大小的识别与自主的路径导航，在非人工干预下自主完成对货物的分拣。

## 文档概述

### 文档用途

本文档为软件设计说明文档，用于对机器人进行具体的功能与结构设计。

### 内容组织

本文档包含项目概述及文档概述、需求概述、数据库设计、体系结构设计、接口设计、详细设计、运行和开发环境和需求可追踪性说明。

## 术语和缩略词

专业术语/缩略词有：

ROS：机器人操作系统

RoboWare Studio：基于VS Code的一款专门为ROS设计的编辑器

Kinect2：3D体感摄像机

## 引用文档

参考的文档有：

* 《SDP-软件开发计划 203》
* 《SRS-软件需求规格说明书 203》
* 《启智ROS版\_开发手册\_20181109》
* 《启智ROS机器人》

# 需求概述

## 业务需求

### 障碍识别与路径规划

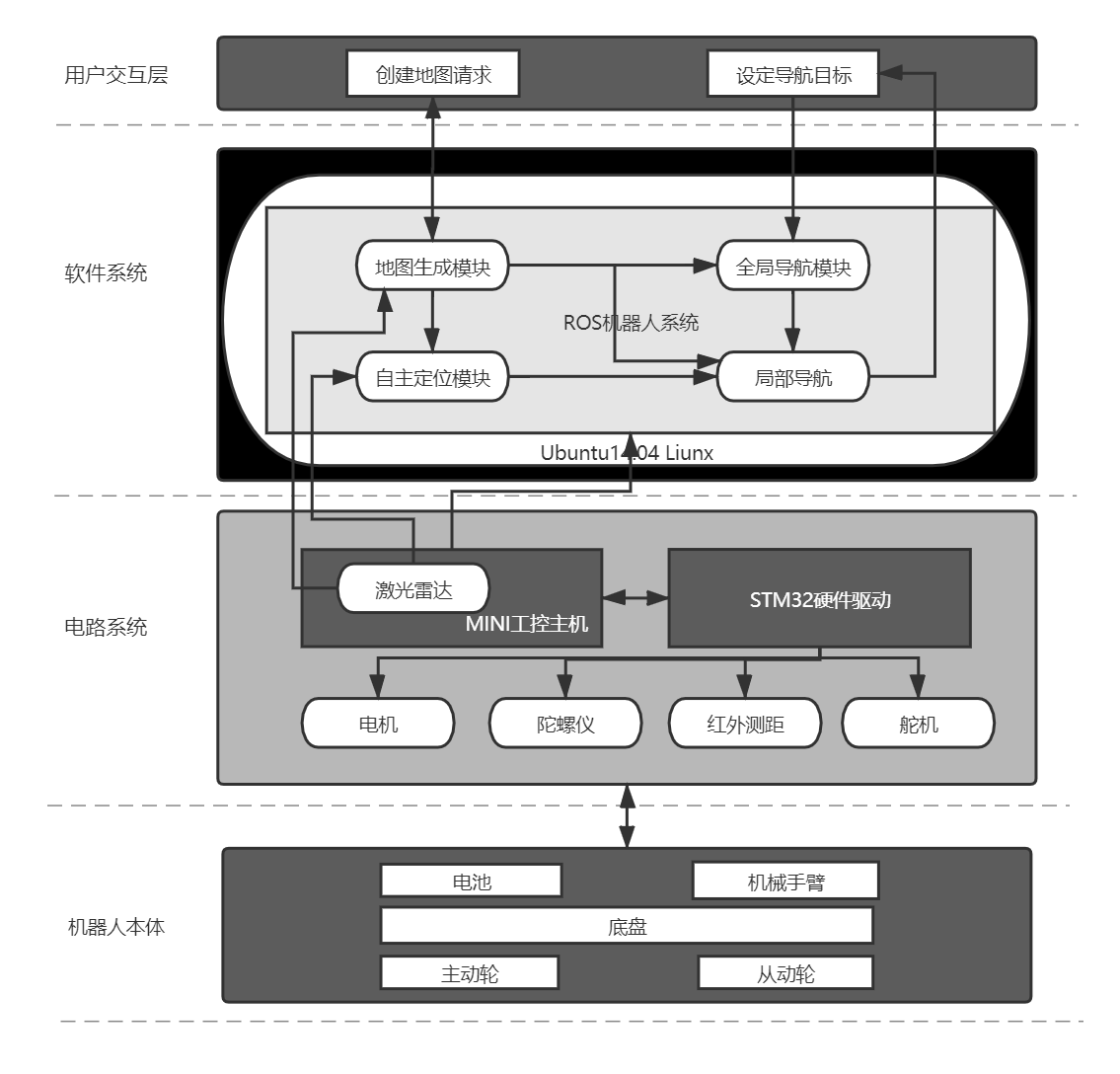
管理员可以通过手动设定目的地，使得机器人去往指定地点。机器人在移动过程中需确认自己所在位置、根据当前位置与目的地进行实时规划、识别当前路径中的障碍物、躲避障碍物后再次进行路径规划。

### 目标物体识别与抓取

管理员在设定好机器人待抓取的物体之后，机器人需通过图像识别出该物体，并确认自己与该物体的相对位置。移动至可抓取的位置并将机械臂移动至抓取动作起始位置后，开始抓取动作。抓取后识别可放置物体地点，执行前述操作的逆操作，将物体放回。

## 数据需求

ROS机器人的基本架构如下所示



**图2-2 基本架构图**

## 功能需求

### 语音识别

**主要参与者：**用户

**目标：** 识别用户的语音输入指令，从中解析出目标物体和目标地点（可无）

**前置条件：**机器人正常

**启动：**用户操作正确

**场景：**

1. 用户发出指令
2. 机器人收到并解析指令
3. 指令解析正确则执行，否则提示用户重新输入指令

**优先级：**中

**何时可用：**用户发出语音指令时

**使用频率：**高

**次要参与者：**无

### 地图建立

**主要参与者：**管理员

**目标：** 对指定区域进行扫描并建立地图

**前置条件：**机器人开机且可以正常运行

**启动：** 机器人在指定区域运动并扫描周边环境，建立地图

**场景：**

1. 管理员下达地图建立指令，并指定区域。
2. 机器人对指定区域进行遍历，并不断用激光雷达扫描周边环境
3. 在扫描过程中建立三维地图，对检测到的障碍物位置进行标记
4. 将地图存储在数据库中

**优先级：**高

**何时可用：**机器人转移到未知场所时

**使用频率：**低

**次要参与者：**激光雷达

### 路径规划

**主要参与者：**用户

**目标：** 打开/关闭机器人

**前置条件：**地图建立完毕

**启动：**用户操作正确

**场景：**

1. 从用户指令中解析出目标地与目标物
2. 对目标地进行路径规划
3. 机器人开始运动
4. 运动过程中检测障碍，若出现新的障碍则重新进行路径规划并修改地图
5. 到达指定目标点

**优先级：**高

**何时可用：**机器人关闭/开启时

**使用频率：**中

**次要参与者：**机器人运动模块，激光雷达

### 目标物体识别抓取

**主要参与者：**用户

**目标：** 机器人在路径规划结束后成功识别出用户指定物体的位置并抓取目标物

**前置条件：**路径规划成功

**启动：**路径规划完毕后启动

**场景：**

1. 从用户指令中解析出目标地与目标物
2. 对目标地进行路径规划
3. 到达目标地
4. 检测目标物体
5. 完成抓取

**优先级：**高

**何时可用：**机器人开机时

**使用频率：**中

**次要参与者：**机械臂

## 非功能需求

### 性能指标

* 机器人的主动控制响应时间应在0.5-1s内；
* 当机器人移动过程中在其激光雷达范围内突然检测到有障碍物接近，则应在1-1.5s内减速至停止；
* 在机器人行驶过程中，应当保持在0.15m/s左右的速度匀速行驶，在没有障碍物的情形下减少变速运动；
* 对于已知起点和终点位置的简单环境，机器人路径规划应在100s之内完成起点至终点的行进；对于未知的复杂环境，机器人应在环境感知过程中缓慢接近终点，预计在400s之内完成行进；
* 当目标物出现在机器人感知识别的范围内时，机器人应在1-3s内选择合适的路径动态接近目标物；
* 当机器人抓取目标物时，应该确保抓取目标物行进的过程中不掉落，能够适应最少0.2m/s的行进速度；

### 系统可用性

* 用户能够正常操作控制系统，确保起点、终点的合理规划，如果出现起点和终点不可到达的情况，系统应及时提醒用户，并且不进行任何动作；
* 系统操作应当简洁，确保每个用户学习后能够成功上手；

### 系统可移植性

* 机器人每项功能代码应分为独立模块，代码耦合度不应过大；
* 在确保代码的独立性后，新增功能时保证原有代码的可用性；
* 提供适当的接口，保证BUG的修复不影响其他功能；

### 系统安全性

* 启智ROS机器人是室内机器人，在此环境之外运行可能会损坏机器人；
* 启智ROS机器人的工作平面需要能够承载不小于40kg的重量。如果表面太软，则机器人可能卡住，运动受阻。启智ROS机器人原则上在水平平面上工作，坡道坡度不大于15度， 坡道倾斜度过大可能导致倾覆；
* 启智ROS机器人不具备防水功能，在任何情况下，启智ROS机器人都不应该与雨水，雾，地面积水以及任何其他液体接触，否则可能导致电路和机构损坏；
* 启智ROS机器人设计工作温度为15°C到35°C之间，使用中务必远离明火和其他热源；

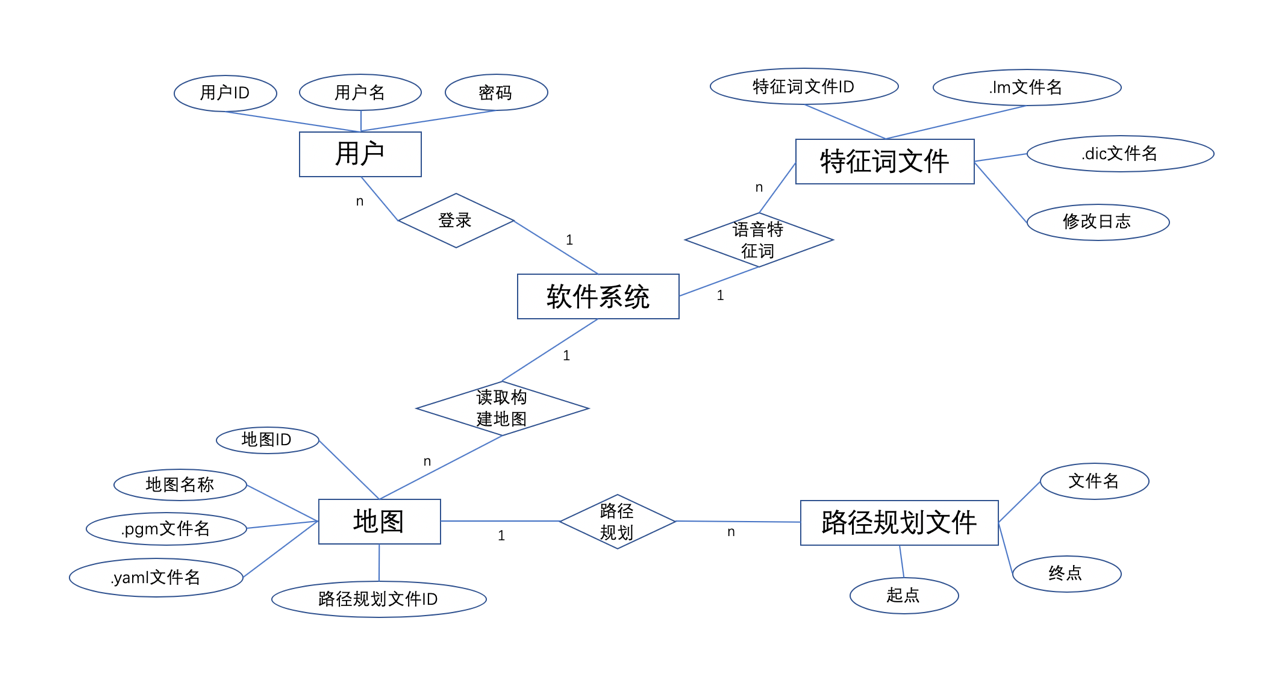
### 系统效率

* 在用户设定好起点和终点时，系统应当在5s内规划出一条最优路径，并能在1s内从静止加速到0.1m/s，即将抵达终点时，应保证机器人在0.2m左右开始减速，同时准备伸展机械臂，在抵达终点的同时，能在5s以内快速抓取目标物，返回起点；

# 数据库设计

本系统主要对三方面的数据进行存储，分别为用户相关内容，地图相关内容和语音特征词相关内容。

## E-R图

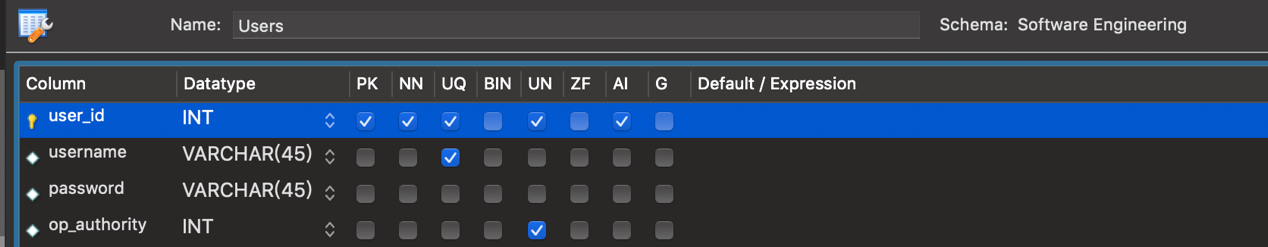


**图3-1 E-R图**

## 数据库关系模式

已对关系模式进行规范化，使其达到第三范式要求。

### 用户信息(Users)



**表3-2-1 用户信息表**

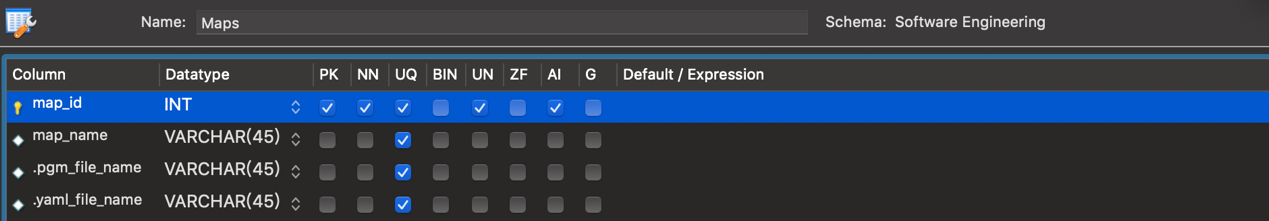
user\_id：用户id，主键

user\_name：用户名

password：密码

op\_authority：操作权限

### 地图信息(Maps)



**表3-2-2 地图信息表**

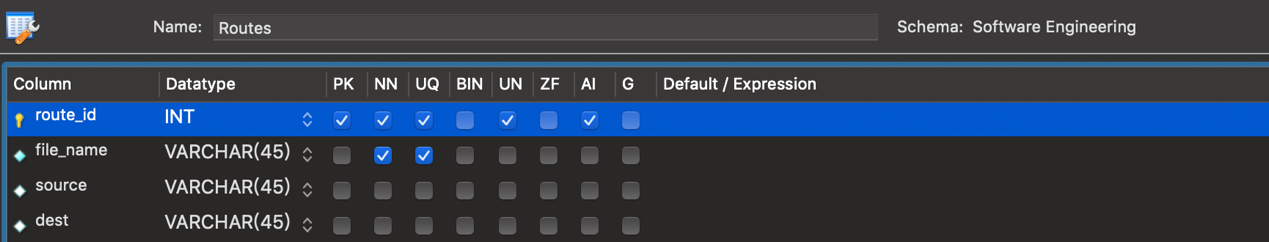
map\_id：地图id，主键

map\_name：地图名称

.pgm\_file\_name：pgm文件名

.yaml\_file\_name：yaml文件名

### 路径信息(Routes)



**表3-2-3 路径信息表**

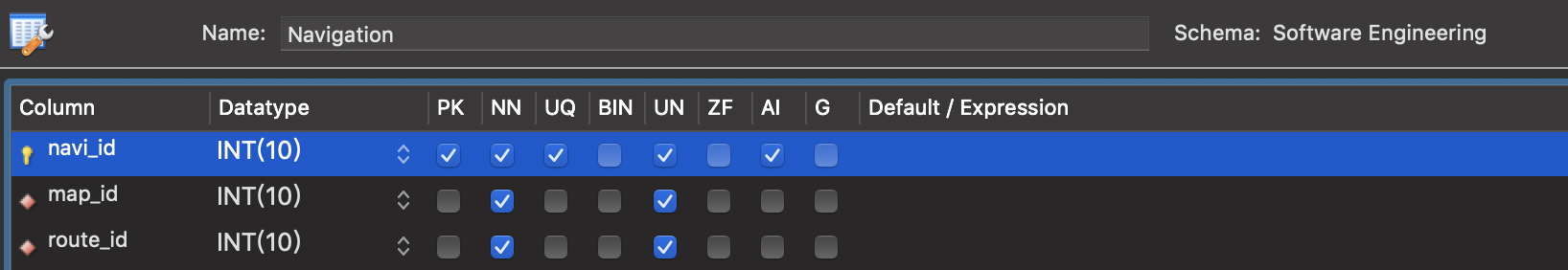
route\_id：路径id，主键

file\_name：路径文件名

source：起点

dest：终点

### 导航信息(Navigation)



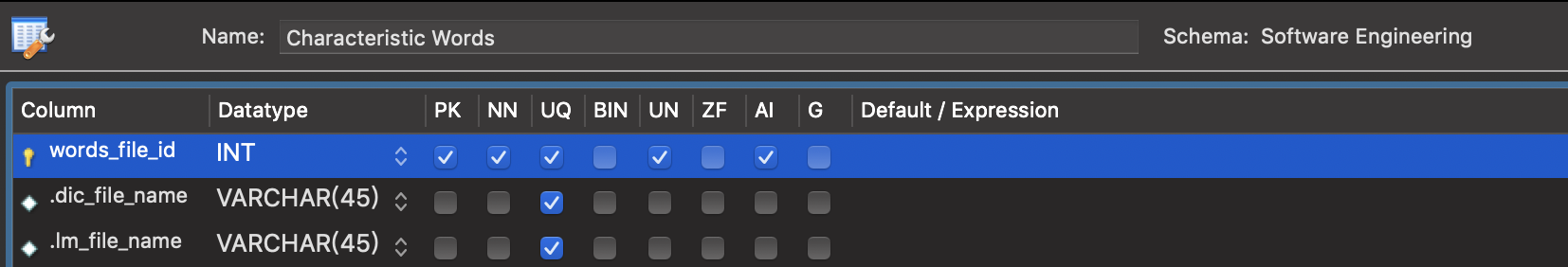
**表3-2-4 导航信息表**

navi\_id：地图-路线对应id，主键

map\_id：所依托于的地图，指向Maps

route\_id：路线id，指向Routes

### 语音关键词(Characteristic Words)



**表3-2-5 关键词表**

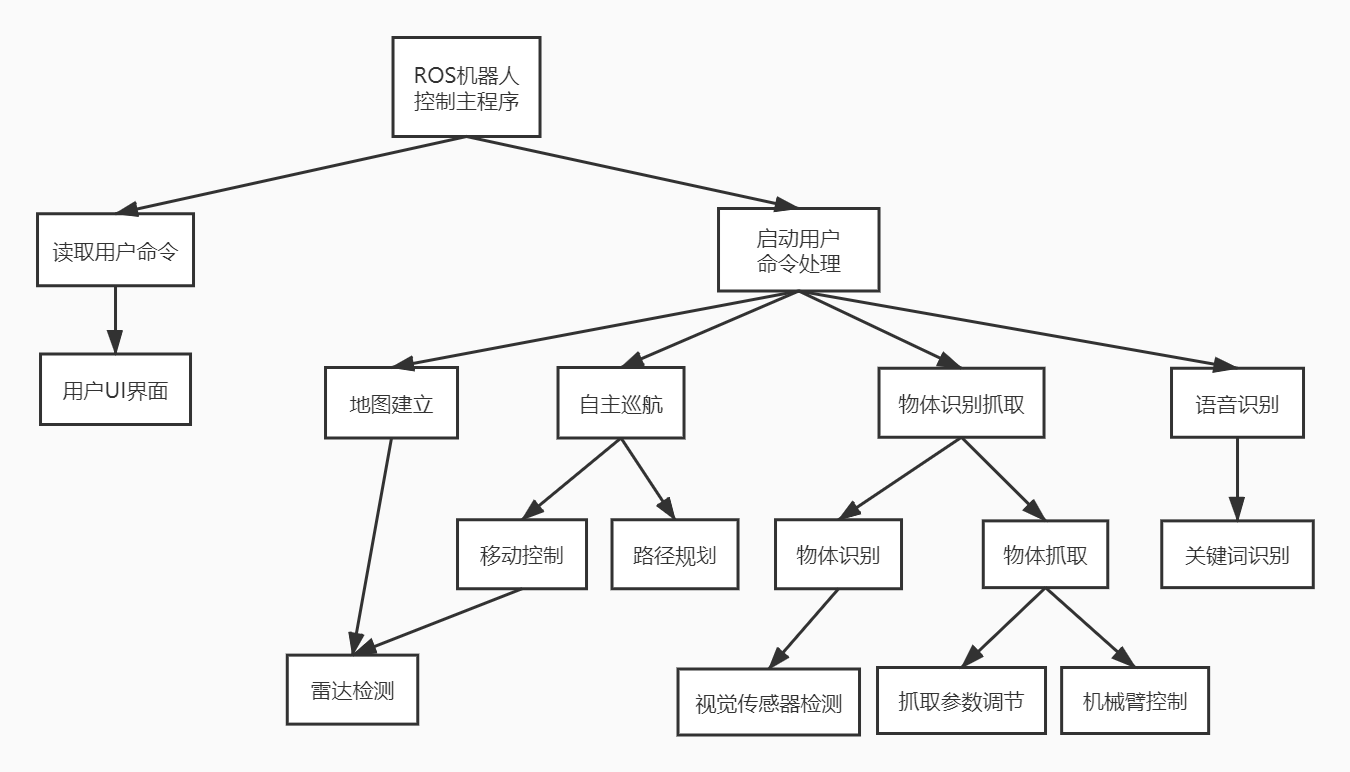
word\_file\_id：.corpus文件id，主键

.dic\_file\_name：对应的.dic文件名

.lm\_file\_name：对应的.lm文件名

# 体系结构设计

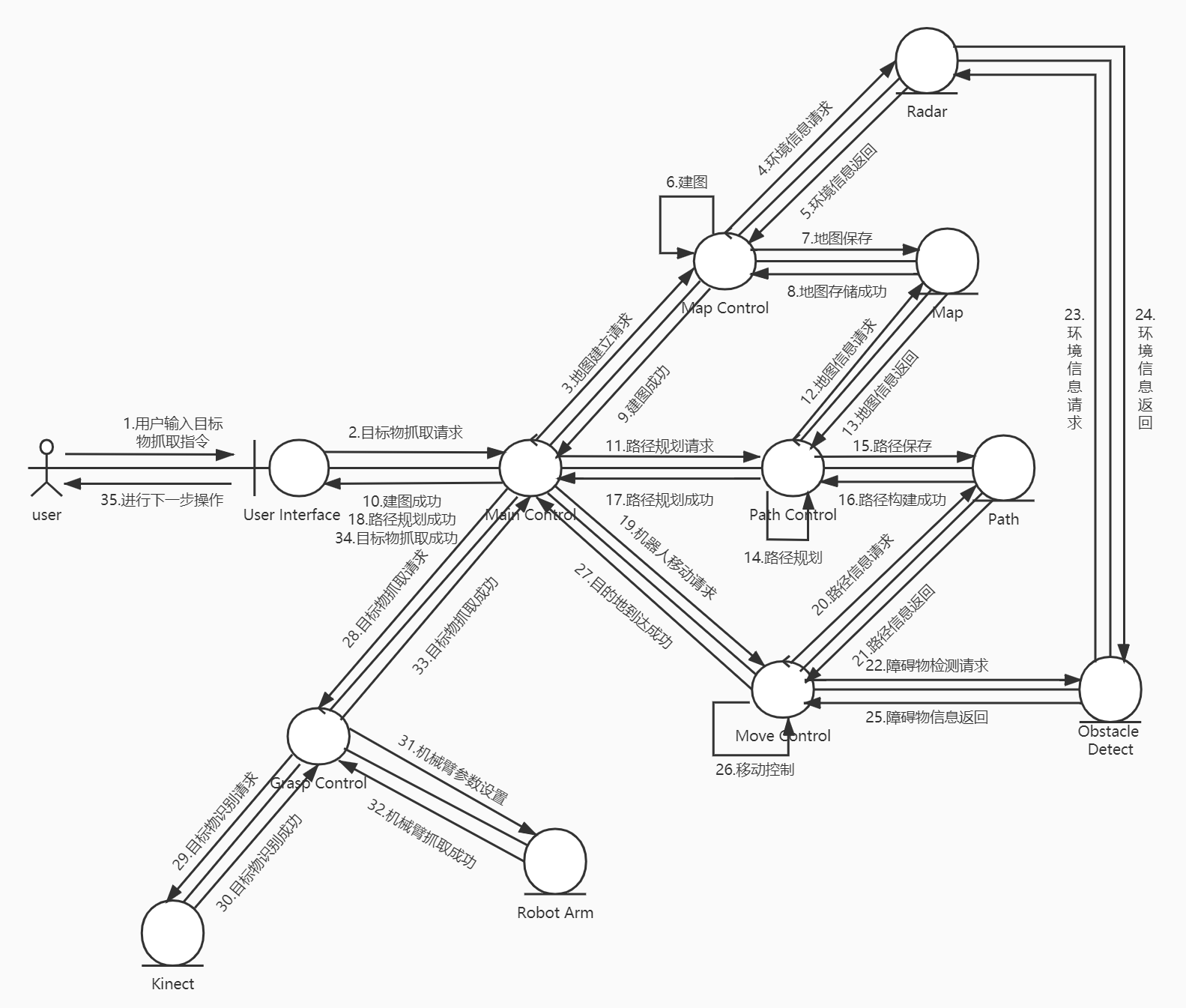
## 总体结构



**图4-1 调用-返回结构图**

用户通过UI界面选择机器人执行功能，并设置相应参数。其中，机器人功能有地图建立、自主巡航、物体识别抓取和语音识别。地图建立功能模块依赖于雷达检测等传感器模块。自主巡航模块需要实现移动控制和路径规划。其中，移动控制模块中的避障和目的地检测功能也同样依赖于雷达检测模块，需要实现机器人底盘控制；路径规划模块则是对创建好的地图进行路径规划，将生成的路径传送给移动控制模块中。到达目的地后，机器人的物体识别抓取模块中，物体识别部分依赖于Kinect视觉传感器检测并且识别出目标物体，而物体抓取部分则需要通过参数调节使机器人到达一个合适的位置，再利用机械臂完成目标物体的抓取。语音识别模块中，依赖于PocketSpinx软件包进行语音关键词识别，从而对语音指令进行解析。

## 软件体系结构



**图4-2 软件协作图**

在抓取目标物体的用例中，软件协作图总共抽象出12个类，其中：

控制类：

Main Control类负责机器人各模块间的总体调度控制；

Map Control类负责机器人地图建立模块控制；

Path Control类负责机器人自主巡航模块控制；

Move Control类负责机器人移动控制；

Grasp Control类负责机器人目标物抓取模块控制；

实体类：

Radar类负责保存雷达检测参数信息；

Map类负责存储构建好的地图信息；

Path类负责存储规划好的路径信息；

Obstacle Detect类负责获取和判断障碍物信息；

Kinect类负责存储图像信息；

Robot Arm类负责存储机械臂的参数信息；

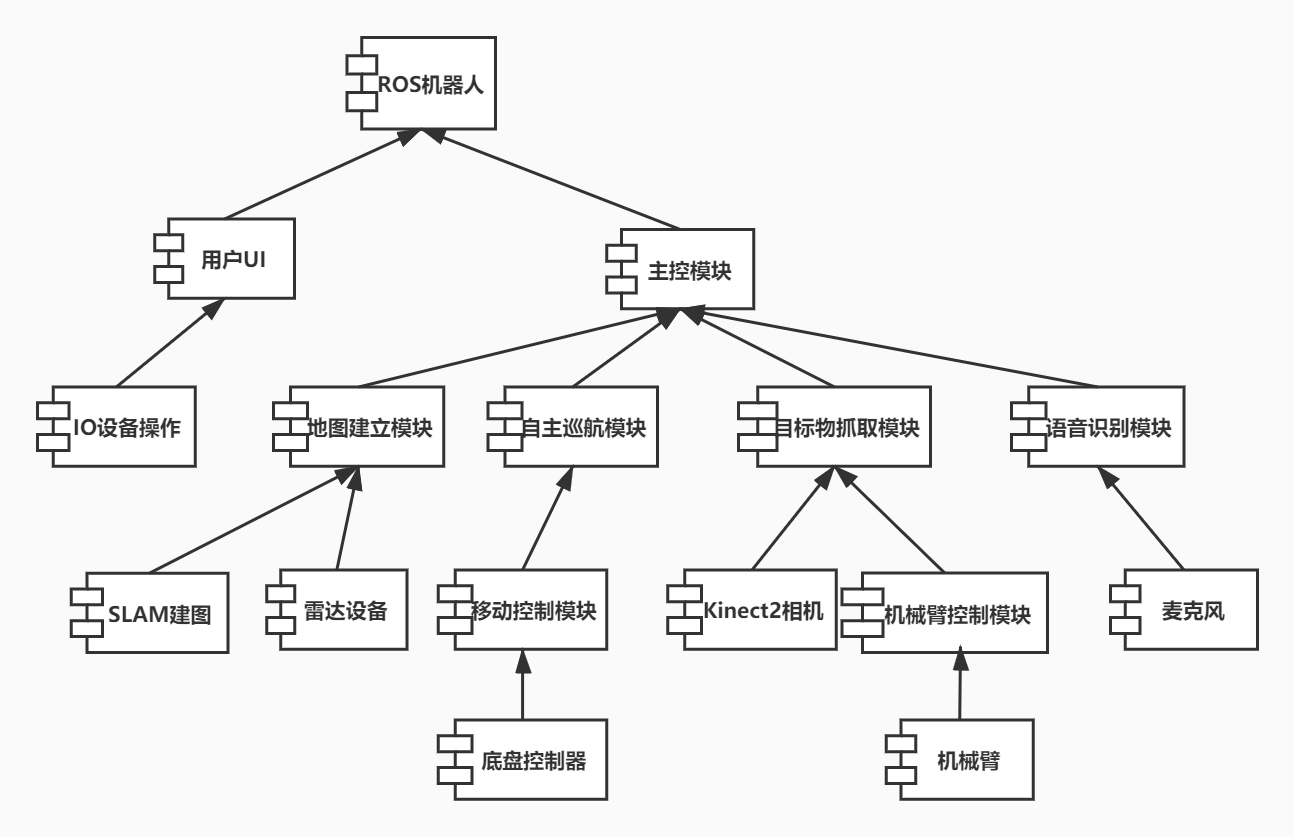
边界类：

User interface类用于实现用户与系统的交互；

抓取目标物体用例整体流程如下：

首先，系统启动进入准备状态后，用户输入目标物抓取指令，并且设置相关参数；User Interface收到指令后，向Main Control发送目标物抓取请求；Main Control接收目标物抓取请求后，将依次向Map Control、Path Control、Move Control和Grasp Control发送具体功能执行请求。在Map Control模块中，首先向Radar请求获得周围环境信息，利用环境信息进行建图，将地图保存到Map中，在Map可以对其进行存储和维护，最后Map Control向Main Control返回建图成功信息。在Path Control模块中，首先向Map请求获得以构建好的地图信息，利用地图信息和Main Control得到的指令信息进行路径规划，将路径保存到Path类中，最后向Main Control返回路径规划成功信息。在Move Control模块中，首先向Path中获得规划的路径信息，再向Obstacle Detect请求进行障碍物检测，Obstacle Detect对通过向Radar请求得到的信息判断周围是否有障碍物，如果路径中存在障碍物，机器人就会及时停止转入故障处理；否则，机器人会沿着路径进行移动，到达目的地后向Main Control返回目的地到达成功信息。在Grasp Control模块中，首先向Kinect请求进行目标物识别，如果相机周围没有出现目标物，机器人会停止执行任务，报告任务异常；否则，Kinect向Grasp Control返回目标物识别成功和目标物形状信息。Grasp Control经计算向Robot Arm发送机械臂参数进行设置，收到Robot Arm返回的抓取成功信息后向Main Control返回目标物抓取成功信息。在地图建立、路径规划和目标物抓取等三个模块每个模块成功执行后，Main Control都会向用户返回该模块执行成功信息，如果机器人中途停止任务，用户能够知道在哪个环节出了问题。最后，用户界面等待用户进行下一步操作。

## 硬件体系结构

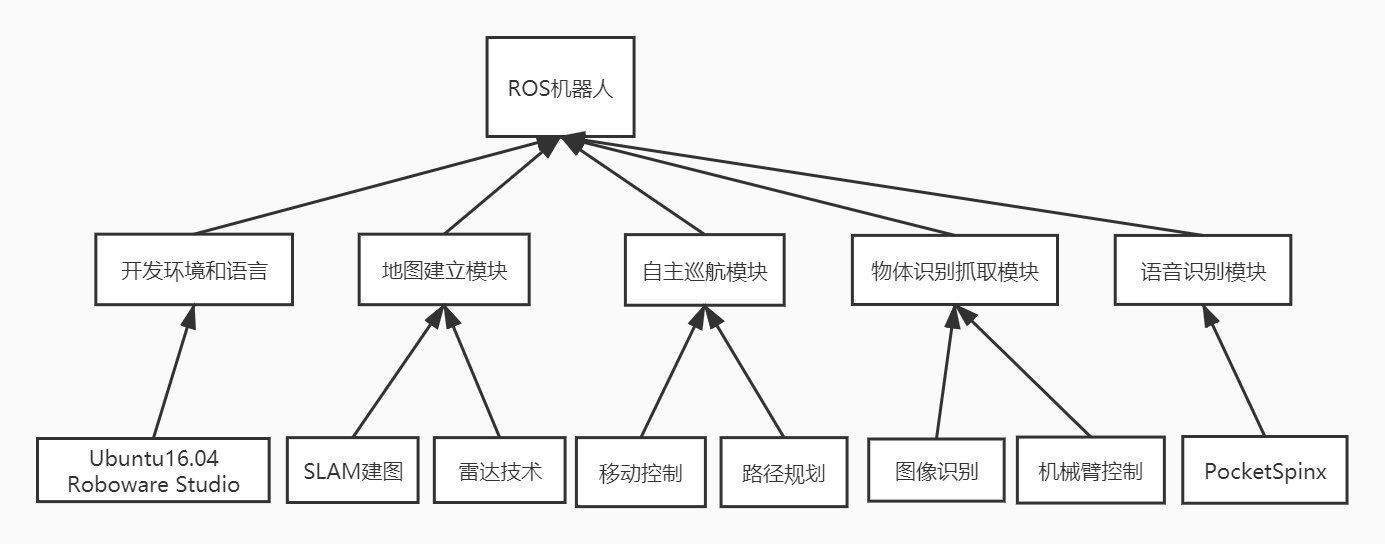


**图4-3 系统构件图(含硬件)**

整个系统的硬件层包含IO设备、激光雷达、底盘控制器、Kinect2相机、机械臂和麦克风等。

构件图表示了软件各种构件的依赖关系，同时也包含了硬软件之间的依赖关系。其中，用户UI经过IO设备接口实现用户指令读取；地图建立模块经过激光雷达设备接口获得周围环境信息；移动控制模块经过底盘控制器参数设置完成机器人的移动；目标物抓取模块需要通过Kinect2相机信息进行目标物识别；机械臂控制模块经过机械臂参数设置对机械臂进行控制；语音识别模块经过麦克风接口读取语音信息。

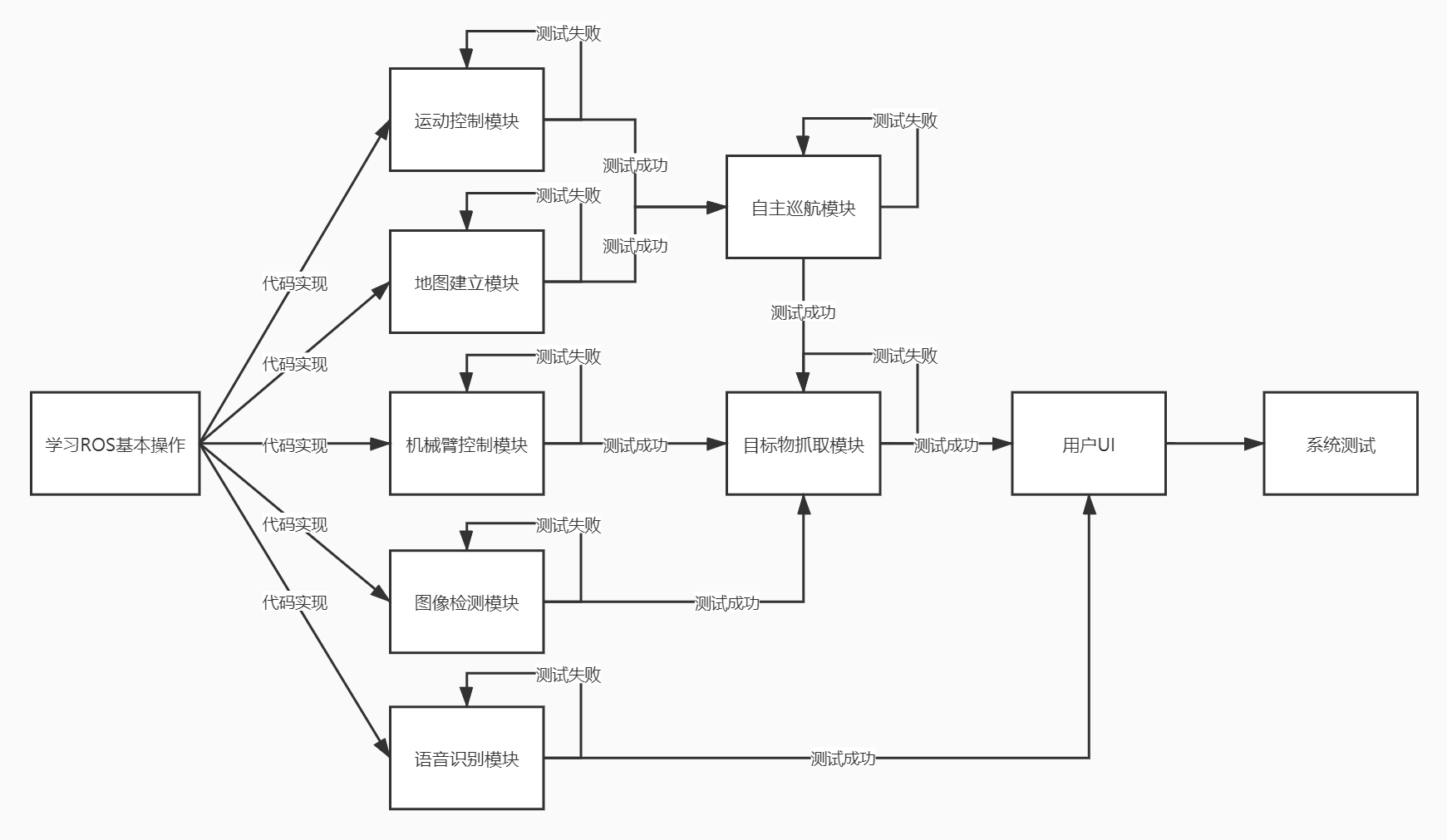
## 技术体系结构



**图4-4 技术体系结构图**

开发整个系统的过程需要很多的技术支持。首先，开发人员需要在Ubuntu 16.04操作系统上进行操作，在Roboware Studio编辑器中编写代码进行软件开发。在代码实现过程中，地图建立模块需要用到SLAM算法和雷达检测技术；自主巡航模块需要用到移动控制技术和路径规划算法；物体识别抓取模块需要用到图像识别技术和机械臂控制技术；而语音识别模块需要用到PocketSpinx等软件包的关键词识别技术。

## 支撑体系结构



**图4-5 开发流程图**

在机器人开发过程中，首先需要统一学习ROS基本操作，然后并发实现运动控制、地图建立、机械臂控制、图像检测和语音识别等五个简单模块，并且可以对每个独立模块进行单独测试。在完成5个基本模块后，需要多人完成自主巡航模块，完成路径规划和避障功能测试。接着再实现目标物抓取模块，完成ROS机器人目标物体识别与抓取。接着，编写用户UI界面，实现用户之间的交互过程。最后，对系统进行整体测试，对代码进行调整，完成软件开发。

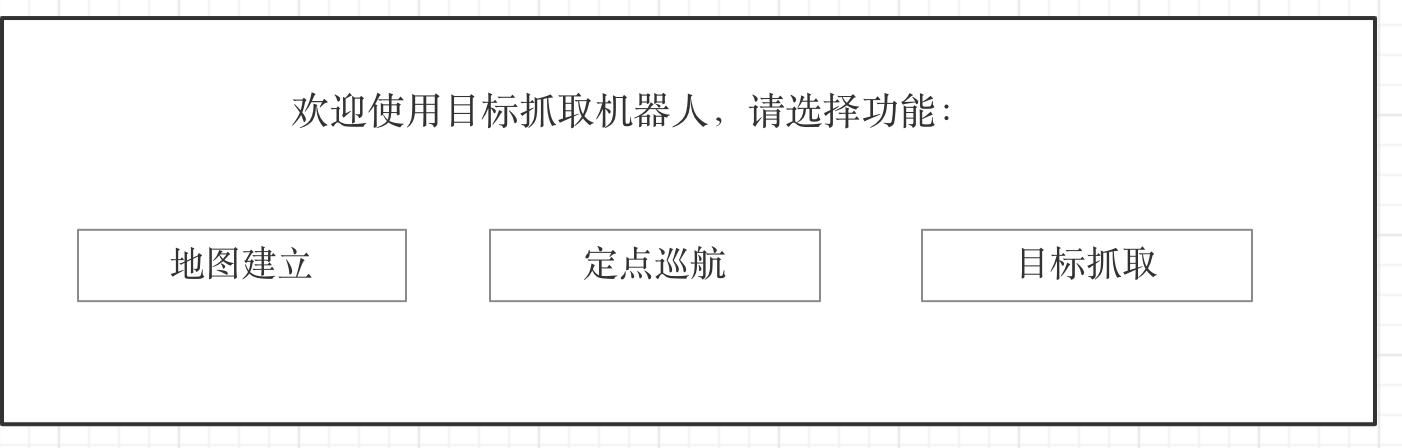
# 接口设计

**5.1 图形用户界面接口设计**

考虑到我们的机器人便于不具有计算机背景的人群使用，故我们选取了图形用户界面而非命令行界面作为系统用户界面。通过图形化用户界面中的按钮，输入框等元件，用户可以通过鼠标键盘或者触摸屏的形式和机器人进行交互。

**5.1.1 开机初始界面设计**

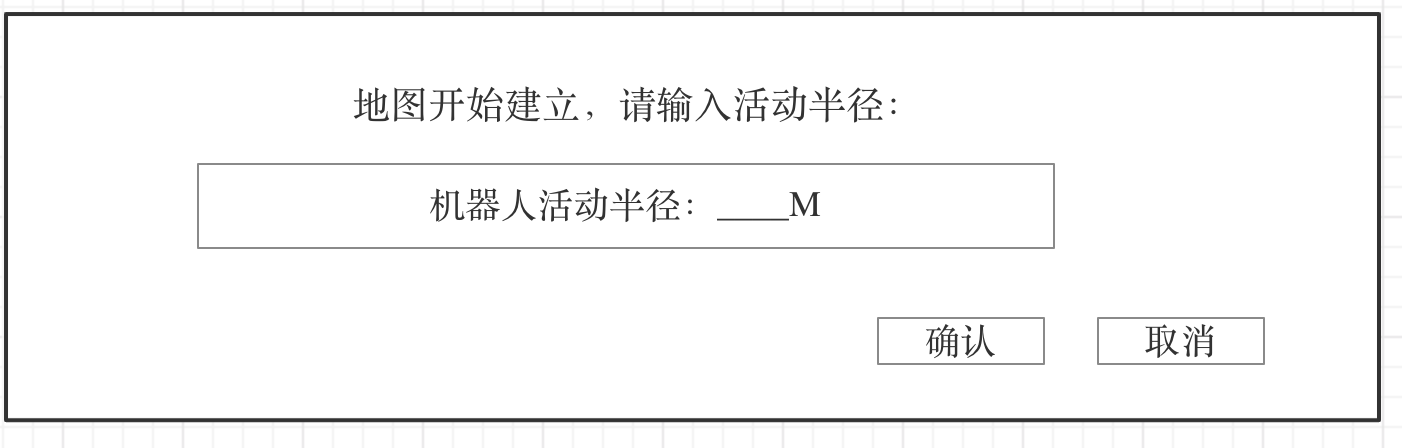
机器人启动后，开机界面如下图所示。用户可以通过鼠标或触摸屏任意选取一个功能。选取之后用户界面则跳转至指定功能的用户界面。



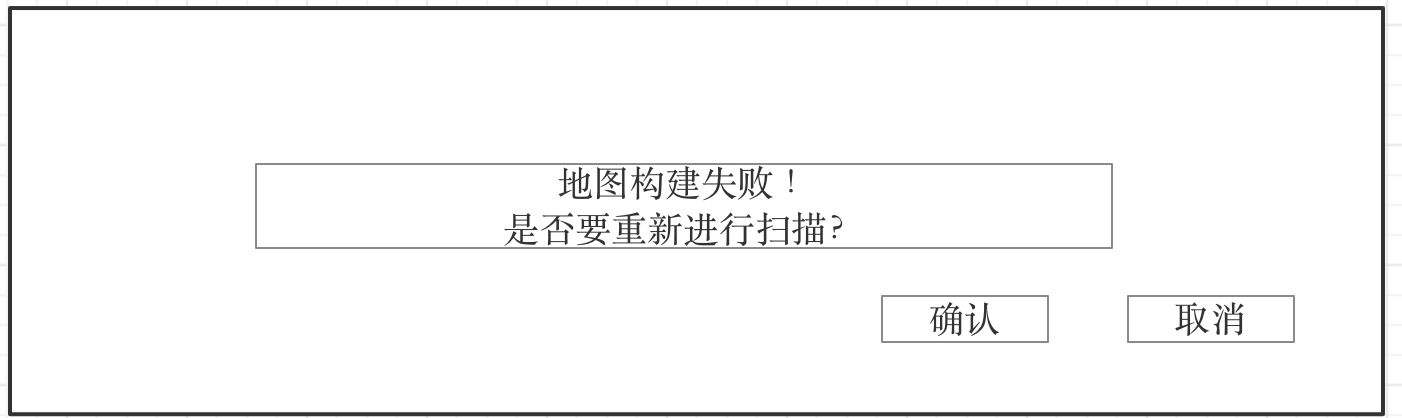
**图 5-1-1 开机初始界面**

**5.1.2 地图建立功能用户界面设计**

机器人开机后，用户选择地图建立后即可进入本界面。

**图 5-1-2.1 地图建立界面 1**

用户可手动输入活动半径并确认输入，也可通过语音输入并由机器人通过语音识别功能，解析出指令。之后机器人即开始通过运动，利用激光雷达对周边环境进行扫描并构建地图。



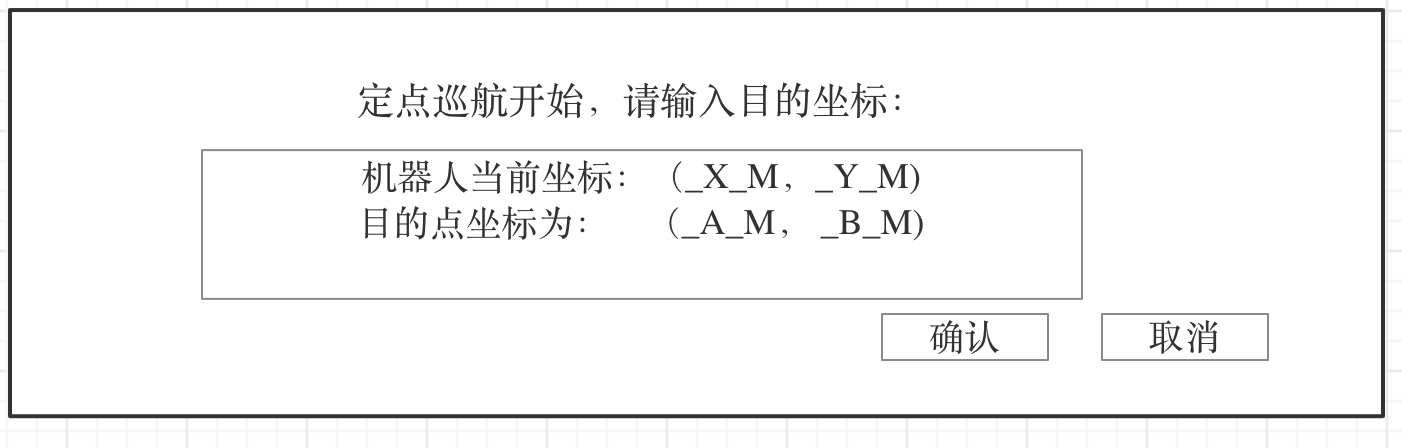
**图 5-1-2.2 地图建立界面 2**

构建失败后选择确认则进行重新扫描，返回地图功能建立初始界面并重新输入活动半径。选择取消则返回开机初始用户界面。

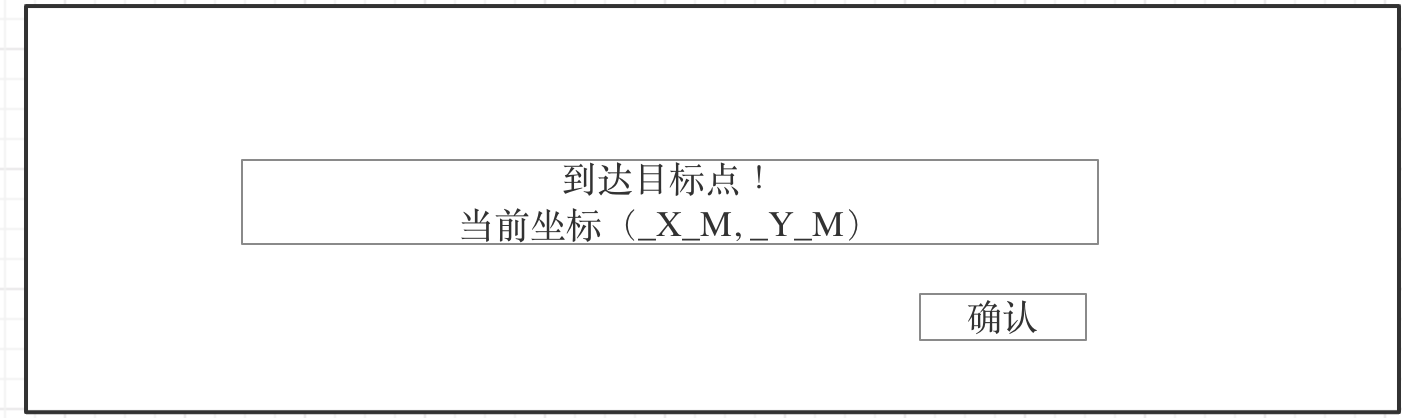
**5.1.3 定点巡航功能用户界面设计**

机器人开机后，用户选择定点巡航后即可进入本界面。

用户可手动输入目的地点的坐标并确认输入，也可通过语音输入并由机器人通过语音识别功能，解析出指令。之后机器人即可向指定目标地点运动。

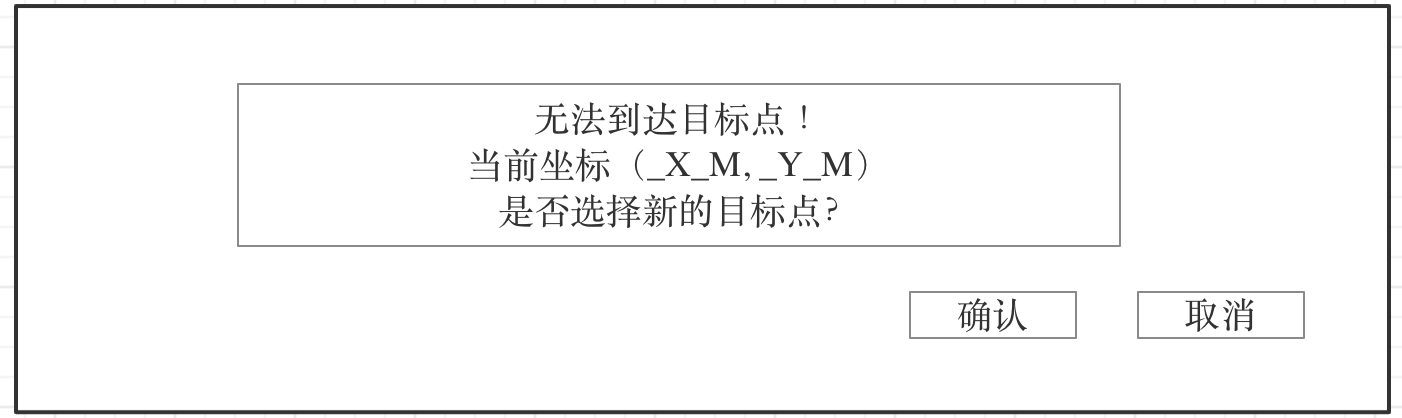
**图 5-1-3.1 定点巡航界面 1**

机器人当前坐标在运动过程中实时变化。到达目的点后反馈运动结果。



**图 5-1-3.2 定点巡航界面 2**

成功后用户选择确认，从而返回开机初始界面。

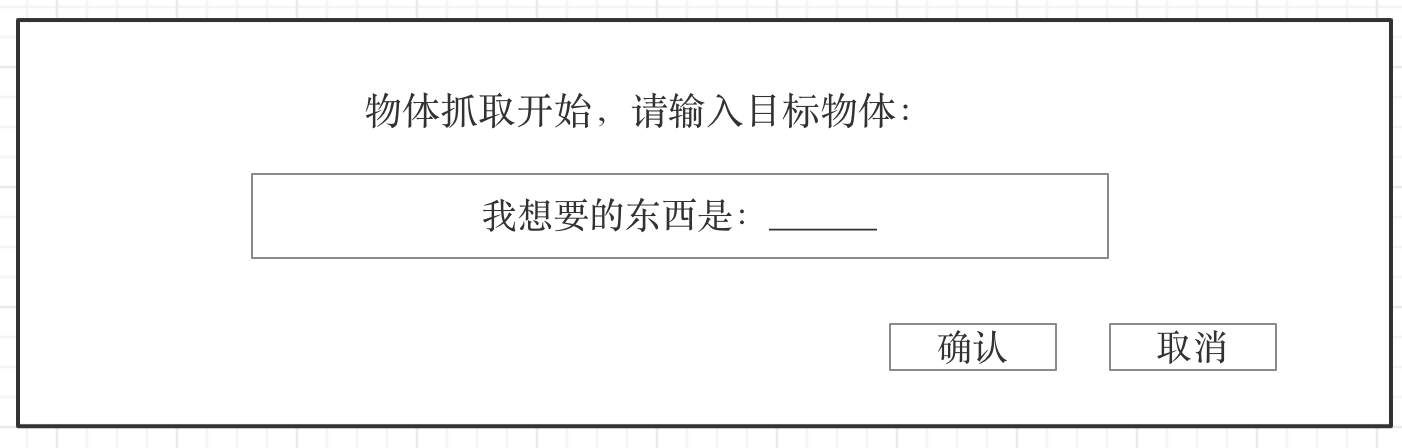


**图 5-1-3.3 定点巡航界面 3**

若无法到达目标点，则可以让用户选择新的目标地点或选择返回初始界面进行其他操作。

**5.1.4 物体抓取功能用户界面设计**

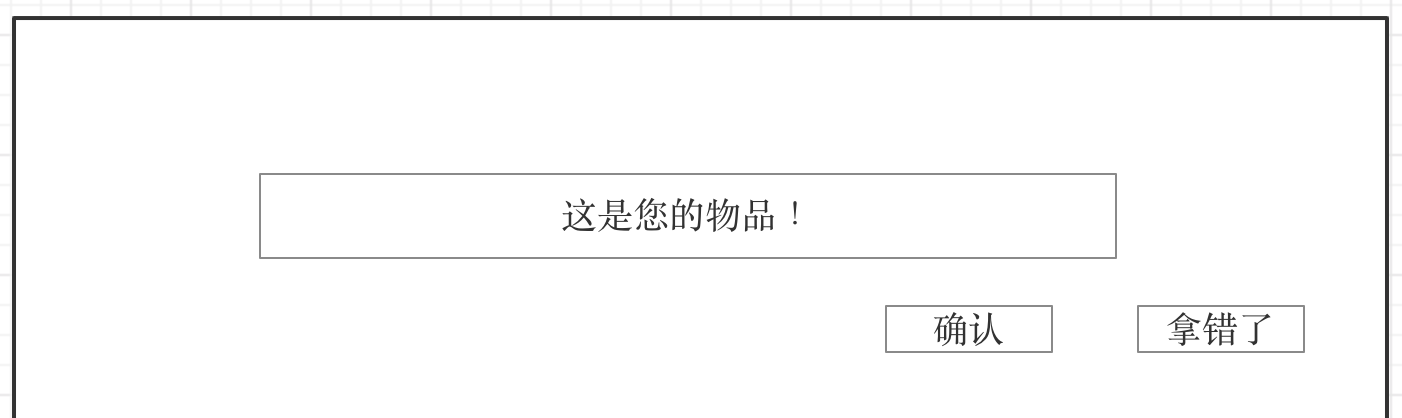
机器人开机后，用户选择目标抓取后即可进入本界面。



**图 5-1-4.1 物体抓取界面 1**

用户可手动输入目的物体并确认输入，也可通过语音输入并由机器人通过语音识别功能，解析出指令。之后机器人即可向指定目标地点运动。

机器人在抓取完成后，返回相应的用户界面。



**图 5-1-4.2 物体抓取界面 2**

用户可选择确认，并拿走物品，之后机器人返回到开机初始界面。或选择拿错了，机器人将返回抓取功能初始界面并提示用户重新输入。

**5.2 硬件内部接口设计**

我们的机器人硬件平台为启智ROS机器人，其硬件设备包括运动底盘，机械臂以及多种传感器。在我们的项目中我们主要使用了运动底盘，机械臂，激光雷达和Kinect2相机这四种硬件设备。

### 5.2.1 运动底盘

ROS机器人采用的是三轮全向式移动底盘，拥有较高的自由度。全向底盘可以在不改变朝向的情况下往水平面上的任何方向移动，这在进行目标跟踪和运动避障时，可以减少机体位置调整的步骤，减少调节时间，提高执行效率。

**5.2.2 机械臂**

启智 ROS 可以扩展安装一个用于抓取桌面上物品的机械臂，该机械臂提供两个控制量: 上升高度和手爪的闭合宽度。 通过修改这两个参数可以实现简易的物体抓取功能。

**5.2.3 激光雷达与SLAM**

激光雷达是地面移动机器人常用的一种传感器，其工作原理是用一个高速旋转的激光测距探头将周围 360°的障碍物分布状况测量出来，形成障碍物轮廓的俯视二维点阵输入到 ROS 系统里。在激光雷达的基础上，启智ROS还支持了激光雷达的数据应用：SLAM。SLAM的英文全称是“Simultaneous Localization And Mapping”，翻译过来就是“即时定 位与地图构建”。通过SLAM算法，可以利用激光雷达的定位功能对地图进行构建。

**5.2.3 Kinec2视觉传感器**

启智 ROS 版的头部装备一台 Kinect2 视觉传感器。该传感器可以输出 RGB 彩色视频流和 Depth 深度数据三维点云，借助 OpenCV 和 PCL 等开源图像库，可以对目标物进行准确识别和定位，从而用于识别和抓取目标物体。机器人检测到目标物体后，根据其他传感器获取的信息（例如深度、高度、角度等），对机械臂的行为进行决策，以便抓取目标物体。

**5.3 软件内部接口设计**

在设计机器人软件时我们的选取了面向对象的设计。根据机器人所需的各种不同的功能，设计出了以下类。类之间可以通过信息传递的方式进行交互。

**5.3.1 机器人主控制类 MainControl**

MainControl类负责机器人各模块间的总体调度控制。通过和和地图控制类(MapControl)，路径规划类(PathControl)，移动控制类(MoveControl)以及抓取控制类(GraspControl)的交互，并将机器人的相关信息返还给用户界面接口(UserInterface)。

**5.3.2 地图控制类 MapControl**

地图建立类的基础为激光雷达与地图类。实现了读取环境信息，地图建立，地图保存三个功能。MapControl类接受MainControl的地图建立请求后，通过Radar类获得环境信息从而完成地图的创建并通知Map类对地图进行保存保存。对外接口和功能介绍如下:

BuildMap()：根据Radar的环境信息创建地图。

该类可由MainControl类调用，建图成功后返回成功信号。

**5.3.3 地图类 Map**

作为实体类，Map类负责存储和读取构建好的地图信息。该类实现了读取地图接口GetMap()。该类由MapControl类通过建图来构造，并可以被PathControl类读取地图信息。对外接口和功能介绍如下:

GetMap(): 向PathControl类提供地图信息。

StoreMap(map): 保存MapControl类传来的地图信息。

该类可被MapControl类在创建地图时调用。

**5.3.4 雷达类 Radar**

作为实体类，Radar类负责调用机器人的激光雷达，利用激光雷达扫描环境以及保存雷达检测参数信息。对外接口和功能介绍如下:

getEnvironment(): 利用激光雷达扫描环境并将环境信息返回给调用方。

该类可被MapControl类在构建地图时调用，也可被ObstacleDetect类在机器人运动中进行实时障碍检测时调用。

**5.3.5 路径规划类 PathControl**

作为控制类，PathControl类主要负责为机器人计算到达指定目的地的最短路径。通过和地图类的交互获取地图信息，并从自主巡航类中接受目的地信息之后进行路径规划并返回最短路径。对外接口和功能介绍如下:

getShortestPath(location): 接受一个目的地，返回到该地点的最短路径。

该类可由MainControl类调用。MainControl发送路径规划请求，由该类完成路径规划并将结果返回给MainControl类。

**5.3.6 路径类 Path**

作为实体类，Path类负责存储规划好的路径信息。对外接口和功能介绍如下:

storePath():由PathControl类调用，保存其规划好的路径，并返回路径构建结果。

getPath():由MoveControl类调用。MoveControl发出路径信息请求，Path类返回相关的路径信息。

**5.3.7 移动控制类 MoveControl**

作为控制类，MoveControl类主要负责机器人的移动控制功能。该类从MainControl类接受机器人移动请求后，通过对Path类的调用实现按照指定路径运动的功能，以及通过对ObstacleDetect类的调用，实现运动时的实时障碍躲避功能。对外接口和功能介绍如下：

move(direction,distance)：机器人向指定方向运动指定距离

move(location)：机器人运动到指定地点

stop()：指示机器人停止运动。

MoveControl类可以由MainControl类调用，以实现机器人在自主巡航过程中在路径规划实现后的具体运动行为。

**5.3.8 障碍检测类 ObstacleDetect**

作为实体类，ObstacleDetect类通过机器人的激光雷达来检测机器人周围的障碍物。在机器人实时运动过程中，通过读取Radar类的环境信息，计算出障碍物的位置信息并返回给调用方。对外接口和功能介绍如下：

getObstacles():根据环境信息计算出机器深周边的障碍物的坐标，并以数组形式返回障碍物信息。

ObstacleDetect类可由MoveControl类调用，从而实现在机器人实时运动中的障碍物检测功能。

**5.3.9 抓取控制类 GraspControl**

作为控制类，GraspContro类负责实现机器人目标抓取控制功能。其基础为Kinect类与RobotArm类。通过向Kinect类发送目标物体识别请求以获取目标物的识别结果和物理坐标，之后可以通过设置RobotArm类的机械臂姿态参数以实现目标物体的抓取功能。

对外接口和功能介绍如下：

getObject(obj):接受一个目标物体描述，根据识别结果进行抓取并返还抓取成功信息给调用方。

GraspControl类可由MainControl类调用。

**5.3.10 物体识别类 Kinect**

作为实体类，Kinect类通过对硬件设备Kinect2视觉传感器的调用，来实现在图像中判断指定物体位置的功能。对外接口和功能介绍如下:

getObjectLoc( ):接受指定物体作为输入，在当前视觉传感器的图像中判断该物体是否存在，若找到则计算并返回该物体相对于机器人的物理坐标，否则返回Null。

物体识别类可以被GraspControl类调用。

**5.3.11 机器臂控制类 RobotArm**

作为机械臂类接受抓取控制类的机械臂参数设置，通过修改机械臂的高度以及手爪的闭合宽度实现物体抓取。并将抓取结果返回给抓取控制类。

对外接口和功能介绍如下:

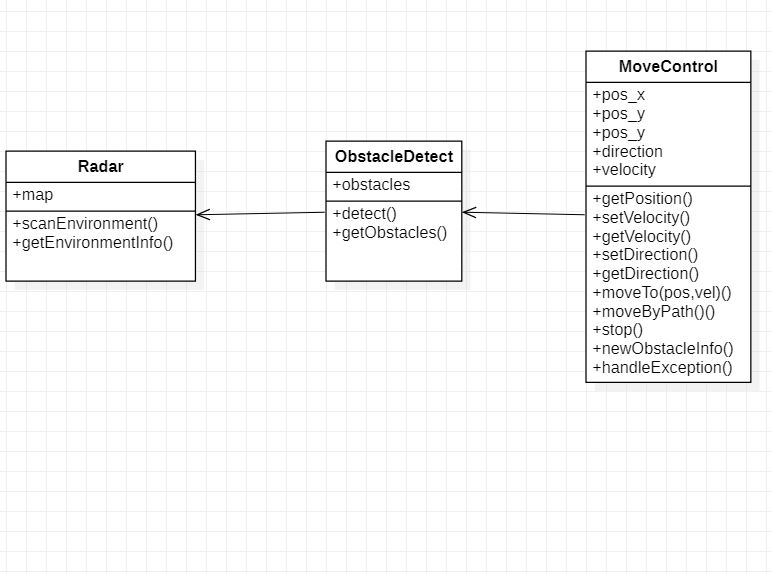
setArmHeight(h):设定机械臂高度。

setGraspWidth():设定机械爪宽度。

RobotArm类可由GraspControl类调用。

# 详细设计

## 移动控制模块



**图 6-1-1 运动控制模块类图**

运动控制模块由以下类构成：MoveControl类负责获取运动指令，设定机器人运动参数，实时检测障碍并将障碍物信息返回给路径规划模块，返回目的地到达结果。

### MoveControl类

MoveControl类的主要属性为机器人在地图中的绝对三维坐标pos；两个运动参数:二维向量运动方向direction和实数运动速度velocity。

|  |  |  |
| --- | --- | --- |
| 成员属性 | 类型 | 描述 |
| pos | (double,double, double) | 机器人在地图中的x,y,z轴的绝对坐标，由三维向量表示 |
| direction | (double,double) | 机器人的面向的方向，用二维向量表示 |
| lin\_velocity | (double,double,double) | 机器人的线速度，由三维向量表示 |
| ang\_velocity | double | ros机器人只存在绕z轴的旋转因此角速度为1维变量 |

* 函数getPosition()：获取机器人的位置信息
* 函数setVelocity(lin\_vel,ang\_vel)：设定机器人运动速度。

主要过程为：

1. 声明ros::NodeHandle对象n,并用 n 生成一个广播对象 vel\_pub，调用的 参数里指明了 vel\_pub 将会在主题“/cmd\_vel”里广播 geometry\_msgs::Twist 类型的数据。
2. 声明一个 geometry\_msgs::Twist 类型的对象 vel\_cmd，并将速度值 赋值到这个对象里。vel\_cmd中的各个速度值分别与相关的类中的属性对应。
3. vel\_cmd 赋值完毕后，使用广播对象 vel\_pub 将其发布到主题“/cmd\_vel”上去。机器人的核心节点会从这个主题接收我们发过去的速度值，并转发到硬件机体去执行。
4. ros::spinOnce()函数给其他回调函数得以执行

* 函数getVelocity()：读取机器人运动速度
* 函数setDirection(dir)：设定运动方向。

通过不断调用getVelocity和setVelocity函数，使当前运动方向与设定运动方向一致。

* 函数getDirection()；读取运动方向
* 函数moveTo(pos，vel )：向指定位置运动。

运动过程中按照参数中的速度值运动。

* 函数moveByPath(Path )：按照path中的路径进行运动。

通过不断调用setVelocity()和setDirection()两个函数，使得机器人的运动符合path中给定的轨迹。

* 函数stop()：停止运动
* 函数newObstacleInfo()：返回新的障碍物信息
* 函数handleException()：异常处理

moveTo()函数主要用于软件开发阶段和测试阶段使用。moveByPath()函数可在路径规划模块编写完成后，按照指定路径运动。同时在运动过程中和障碍检测模块不断交互，若在运动方向上检测到障碍则返回障碍物信息，并利用函数停止运动。

### ObstacleDetect类

在ObstacleDetect类中，障碍物体的坐标信息以数组形式存储。

|  |  |  |
| --- | --- | --- |
| 成员属性 | 类型 | 描述 |
| obstacles | vector<obs\_info> | 保存地图中的障碍物信息 |

* 函数detect()

检测环境中的障碍信息。在机器人运动时，由MoveControl类的两个move函数不断调用。

* 函数getObstacles()

在检测到新的障碍物时将返回障碍信息，由moveControl类根据障碍物体信息对运动姿态进行修改。

### Radar类

Radar类的主要实现了扫描环境与返回环境信息两个函数。在本模块中的应用为，由ObstacleDetect模块调用scanEnvironment()函数对物理环境进行扫描，同时利用getEnvironment()函数获取环境信息。

* 函数scanEnvironment()

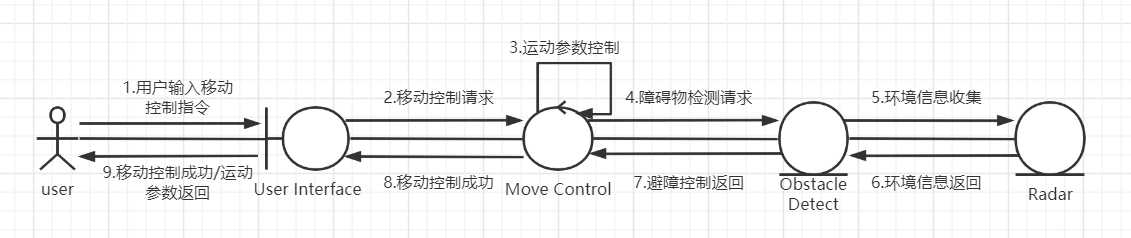
由ObstacleDetect模块调用scanEnvironment()函数对物理环境进行扫描。详细过程为

1. 声明 ros::NodeHandled对象nh
2. subcribe “/scan”主题，调用lidarCallback()函数。
3. 将相关信息保存在ros::Subscriber对象 lidar\_sub中

* 函数getObstacles()

在检测到新的障碍物时将返回障碍信息，由moveControl类根据障碍物体信息对运动姿态进行修改。

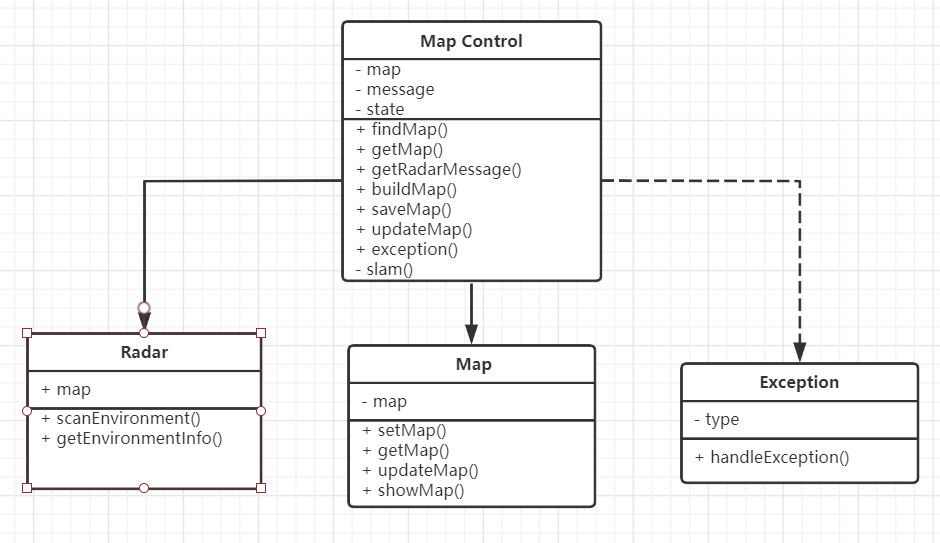
### 用例图



**图 6-1-2 运动控制模块用例图**

**用例描述：**首先，用户在GUI界面选择移动功能，并选择对应的移动控制方式。User Interface将移动控制指令发送给移动控制模块(Move Control)，移动控制模块发布Topic对ROS机器人运动参数进行调节，随后向障碍物检测模块(Obstacle Detect)请求开启避障功能。障碍物检测模块从雷达检测(Radar)中获得环境信息，判断周围是否有障碍物，如果遇到障碍物则及时向移动控制模块发送避障信息，停止机器人运动。最后，移动控制模块将移动控制成功信息和ROS机器人运动参数返回显示在GUI界面中。

## 地图构建模块



**图6-2-1 地图构建模块类图**

这个模块包含Map Control、Map、Radar和Exception等四个类，用于进行地图构建。

### Map Control类

Map Control类用于进行地图构建，包含属性map、message、state分别表示构建的地图、环境信息和控制状态。其中，findMap()和getMap()函数用于从Map类中寻找并获取地图信息；GetRadarMessage()函数用于从Radar类中获取环境信息；buildMap()、saveMap()和updateMap()函数分别用于进行地图的构建、保存和更新；buildMap()需要调用slam()算法进行具体的地图建立；最后添加异常处理函数exception()用于处理环境信息不全等异常情况。

### Map类

Map类用于进行地图存储，Map Control类会调用setMap()、getMap()和updateMap()函数进行地图保存、加载和更新。而测试过程中可以通过showMap()函数进行地图建立功能的调试。

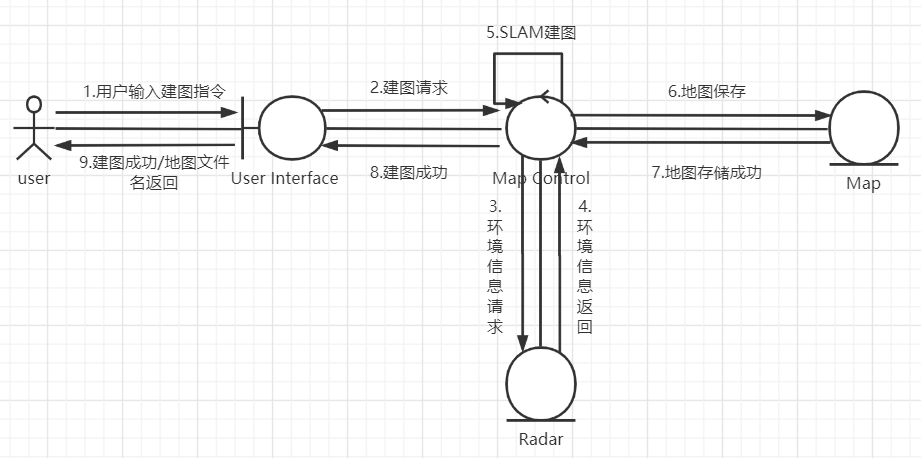
### Radar类

Radar类用于得到雷达收集的环境信息，将信息传递给Map Control类实现地图构建。

### Exception类

Exception类用于异常情况处理。

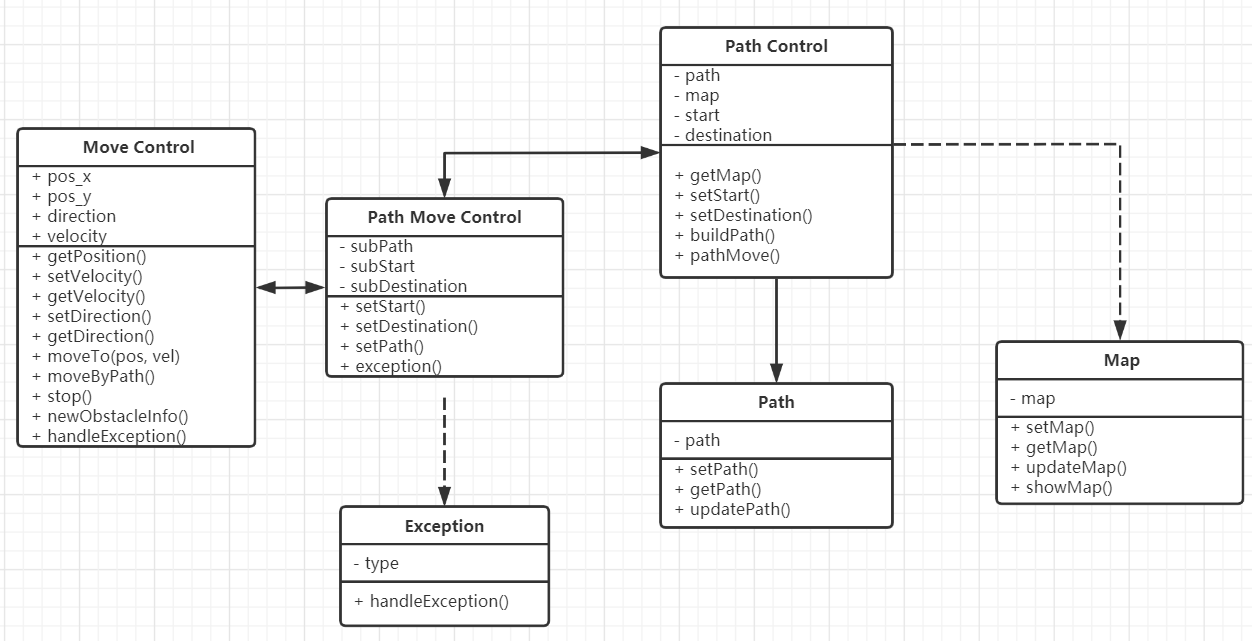
### 用例图



**图6-2-2 地图构建模块用例图**

**用例描述：**首先，用户在GUI界面选择地图建立功能。User Interface将地图构建指令发送给地图模块(Map Control)，地图模块从雷达检测(Radar)中获得环境信息，利用环境信息进行SLAM建图，并将构建好的地图保存在地图实体类(Map)中。最后，地图模块将地图构建成功信息和地图保存路径返回显示在GUI界面中。

## 自主巡航模块



**图6-3-1 自主巡航模块类图**

这个模块包含Path Control、Path Move Control、Path、Map、Move Control和Exception等六个类，其中Map和Move Control在以上模块均有介绍，用于进行自主巡航到达目的地。

### Path Control类

Path Control类用于进行路径规划，包含属性path、map、start、destination分别表示路径信息、地图信息、初始位置和目的位置。其中，getMap()函数用于从Map类中获取地图信息；setStart()和setDestination()函数用于主控制台设置初始位置和目的位置；buildPath()函数用于实现路径规划算法；pathMove()函数用于对规划好的路径进行分段，调用Path Move Control类实现机器人的分段运行；最后添加异常处理函数exception()用于处理路径不通等异常情况。

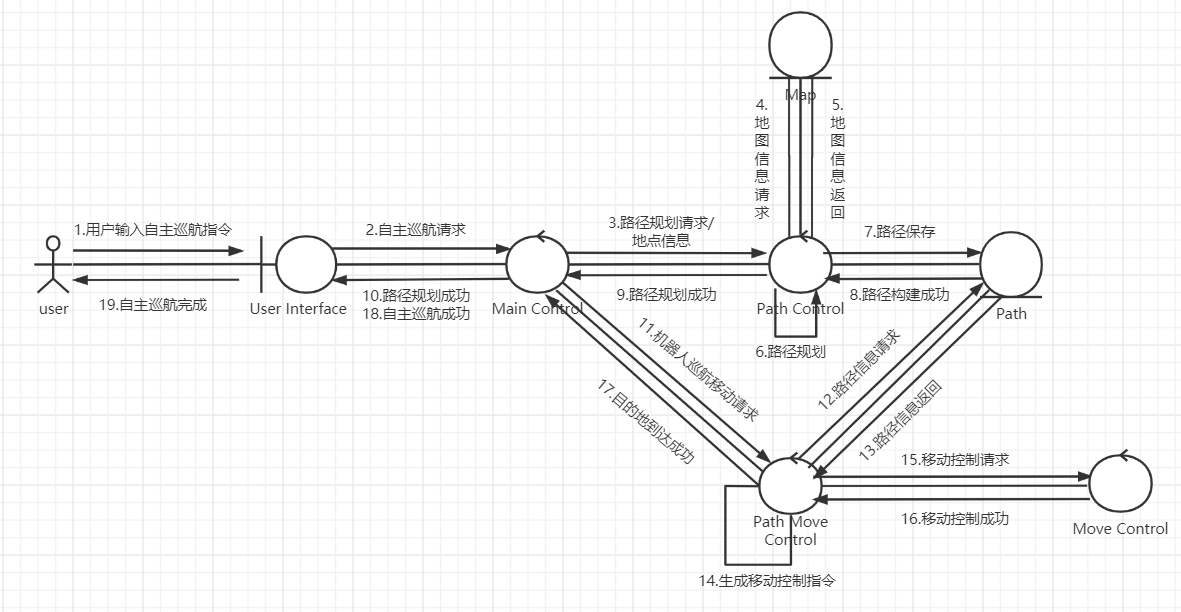
### Path Move Control类

Path Move Control类用于进行机器人的子运行，设置好子出发位置和子到达位置后，调用Move Control模块中的相关函数进行运动，运动结束后返回到达成功，等待下一段路径信息。

### Path类

Path类用于对规划路径的存储。

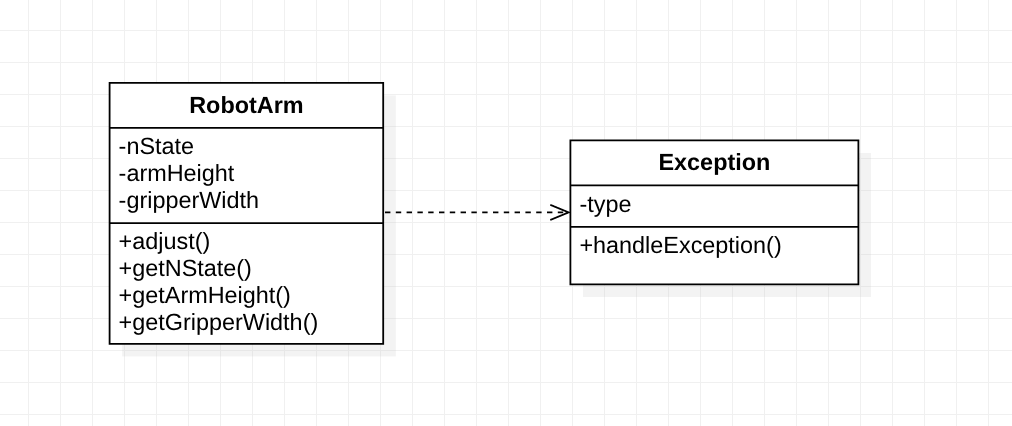
### 用例图



**图6-3-2 自主巡航模块用例图**

**用例描述：**首先，用户在GUI界面选择自主巡航功能。User Interface将自主巡航指令通过主控制模块(Main Control)发送给路径控制模块(Path Control)，路径控制模块从地图实体类(Map)中获得地图信息，利用地图信息、起始点、终点信息进行路径规划，并将规划好的路径信息保存在路径实体类(Path)中。路径控制模块将路径规划成功信息经主控制模块返回显示在GUI界面中。接着，主控制模块向巡航移动模块(Path Move Control)发送移动请求，巡航移动模块从路径实体类中得到路径信息，并利用路径信息生成移动控制指令，调用移动控制模块对机器人进行移动控制。最后，巡航移动模块返回目的地到达成功信息，GUI界面显示自主巡航成功信息。

## 机械臂控制模块

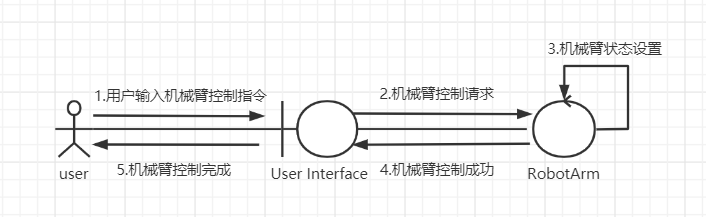


**图6-4-1 机械臂控制模块类图**

此模块只包含两个类，RobotArm类和Exception类。

RobotArm类用于机械臂的控制。内含3个主要属性。nState用于描述当前机械臂所处状态，值为WPBH\_TEST\_MANI\_ZERO等四个值。armHeight与gripperWidth分别为机械臂的两个参数，分别为机械臂高度与机械臂手爪的两指间距。

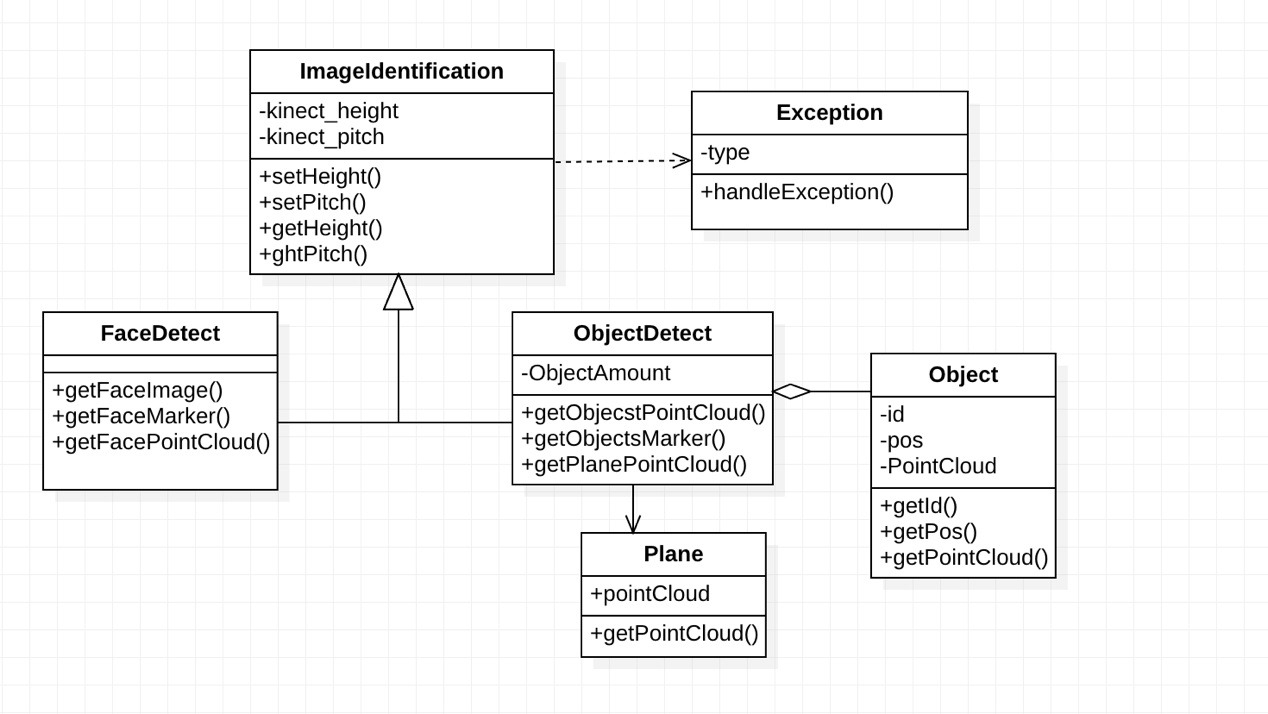
### 用例图



**图6-4-2 机械臂控制模块用例图**

**用例描述：**首先，用户在GUI界面选择机械臂功能。User Interface向机械臂控制模块(RobotArm)发送机械臂控制指令，机械臂控制模块对机械臂参数进行设置，完成机械臂控制，将信息返回到GUI界面。

## 图像识别模块

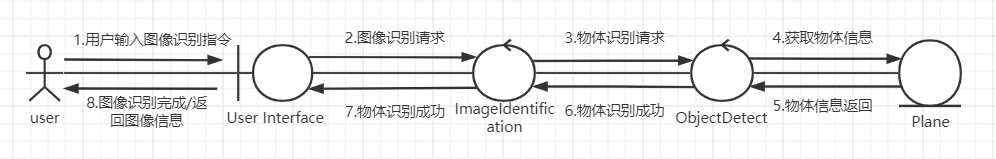


**图6-5 图像识别模块类图**

此模块包括图像识别类ImageIdentification，以及继承了此类的两个图像识别类，人脸面部识别FaceDetect和物体识别ObjectDetect。其中ObjectDetect类由Object物体类与Plane平面类聚合而成。

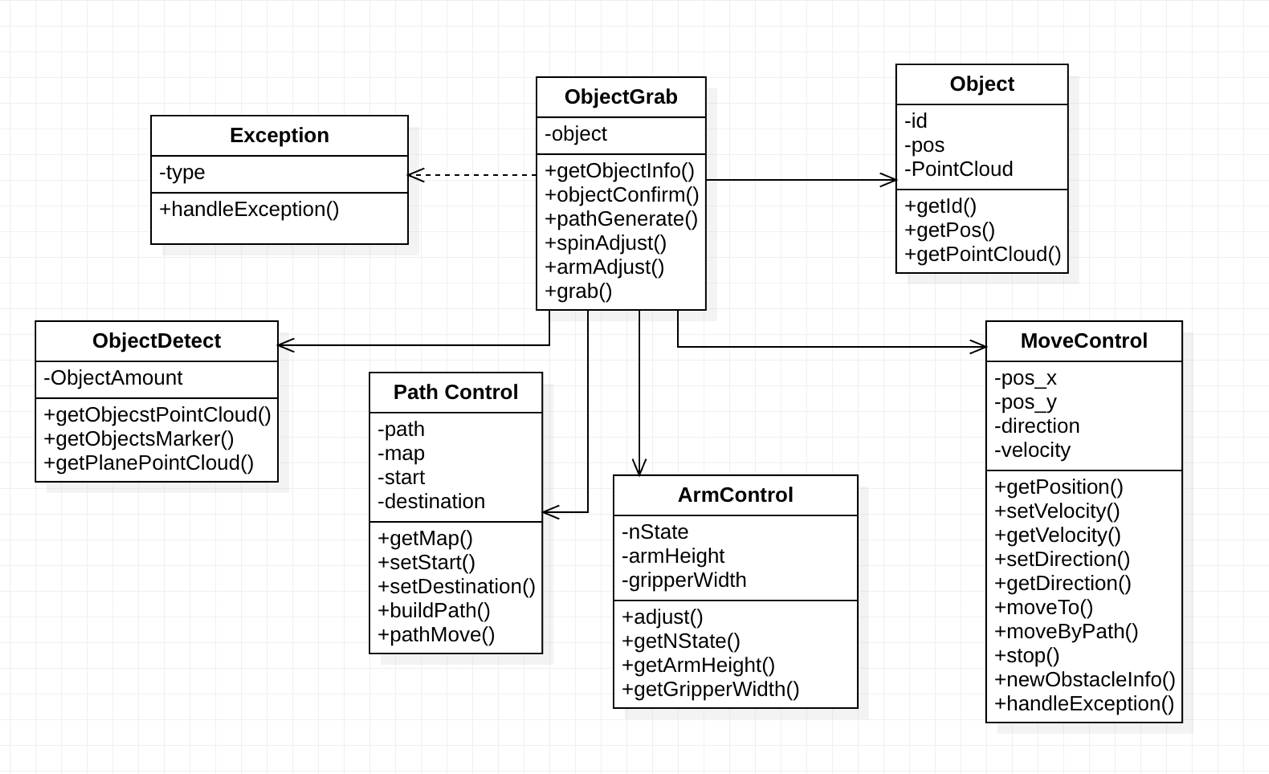
ImageIdentification类是图像识别类，kinect\_height与kinect\_pitch为kinect2摄像头的俯仰参数。FaceDetect面部识别类继承自ImageIdentification，其添加的三个方法分别代表获取彩色人脸识别图像，查看人脸三维点云和查看人脸三维坐标的方法。ObjectDetect物体识别类继承自ImageIdentification，下方三个方法分别为获取物体集三维点云、获取物体集坐标和获取平面的三维点云。物体集由Object类所代表的物体聚合而成，每个物体有自身的id，pos位置信息以及点云信息。Plane平面类只有点云信息。

### 用例图

**图6-5-2 图像识别模块用例图**

**用例描述：**首先，用户在GUI界面选择物体图像识别功能。User Interface向图像识别模块(ImageIdentification)发送图像识别指令。图像识别模块向物体检测模块(ObjectDetect)发送物体识别请求，图像识别模块通过Plane实体类中获取物体三维点云信息进行识别，如果识别成功则返回物体识别成功信息，最后显示在GUI界面中。

## 目标物抓取模块

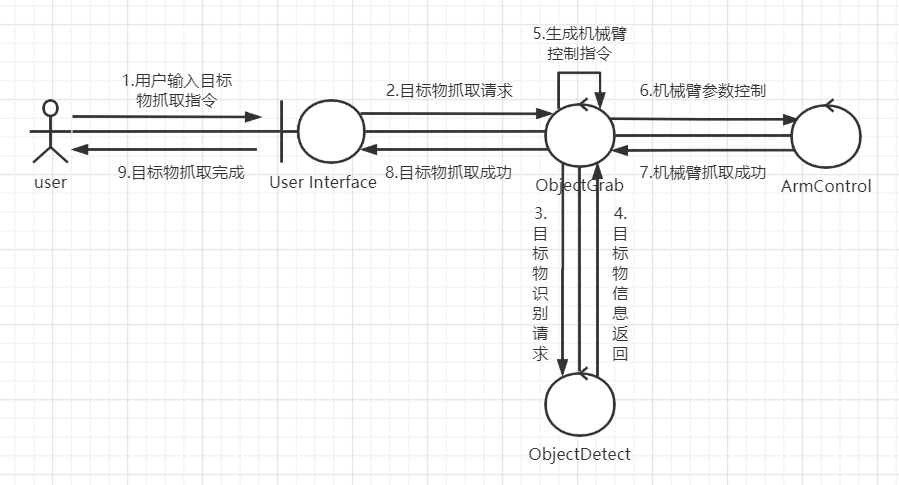


**图6-6-1 目标物抓取模块类图**

此模块由GrabControl, ObjectDetect, Path Control, RobotArm, MoveControl, Object类所组成。其中出主类ObjectGrab类外都已经在前文介绍过。

在ObjectGrab类中，其属性object为所需要抓取的物体信息。首先调用pathGenerate()方法通过Path Control类规划抵达目标地点的路线，通过MoveControl抵达目的地点。之后调用getObjectInfo()和objectConfirm两个类通过ObjectDetect类确认所需抓取的物体。之后根据获取到的物体坐标信息， spinAdjust()方法为抵达目标地点之后的姿态调整方法，包含调整角度，高度等。最后armAdjust()调整机械臂的参数，grab()进行抓取。

### 用例图



**图6-6-2 目标物抓取模块用例图**

**用例描述：**首先，用户在GUI界面选择物体抓取功能。User Interface向物体抓取模块(ObjectGrab)发送物体抓取指令。物体抓取模块从物体检测模块(ObjectDetect)获取目标物三维点云信息，并计算生成机械臂控制指令。物体抓取模块向机械臂控制模块发送机械臂控制指令，完成目标物的抓取。最后，向GUI界面返回目标物抓取成功信息。

# 运行与开发环境

## 运行环境

**硬件环境：**

* 启智ROS机器人
* 1080P高清摄像机
* TOF立体相机
* 面阵麦克风
* 机载电脑锁定器
* 可伸缩支架
* 立体扬阵阵列
* 360°激光雷达
* 三轮全向低盘
* 主控电源开关
* 紧急制动开关
* USB接口
* 电池坞
* 键盘底托
* 记载平板电脑
* 俯仰调节装置

**软件环境：**

* Ubuntu 16.04 / Ubuntu14.04
* ROS Kinetic(Ubuntu 16.04) / Ros Indigo (Ubuntu 14.04)

## 开发环境

**硬件环境：**

* 个人PC
* 最低配置:
* 操作系统: Ubuntu 14.04 or 16.04
* 处理器: Dual core from Intel or better
* 内存: 4 GB RAM
* 显卡: nVidia 320M or higher, or Radeon HD 2400 or higher, or Intel HD 3000 or higher
* 网络: 宽带互联网连接
* 存储空间: 需要 15 GB 可用空间

**软件环境：**

* Ubuntu 16.04 / Ubuntu14.04
* ROS Kinetic(Ubuntu 16.04) / Ros Indigo (Ubuntu 14.04)
* RoboWare Studio

# 需求可追踪性说明

## 功能需求

### 语音识别

**主要参与者：**用户

**目标：** 识别用户的语音输入指令，从中解析出目标物体和目标地点（可无）

**前置条件：**机器人正常

**启动：**用户操作正确

**场景：**

1. 用户发出指令
2. 机器人收到并解析指令
3. 指令解析正确则执行，否则提示用户重新输入指令

**优先级：**中

**何时可用：**用户发出语音指令时

**使用频率：**高

**次要参与者：**无

**相关模块：**Voice Detect类

### 地图建立

**主要参与者：**管理员

**目标：** 对指定区域进行扫描并建立地图

**前置条件：**机器人开机且可以正常运行

**启动：** 机器人在指定区域运动并扫描周边环境，建立地图

**场景：**

1. 管理员下达地图建立指令，并指定区域。
2. 机器人对指定区域进行遍历，并不断用激光雷达扫描周边环境
3. 在扫描过程中建立三维地图，对检测到的障碍物位置进行标记
4. 将地图存储在数据库中

**优先级：**高

**何时可用：**机器人转移到未知场所时

**使用频率：**低

**次要参与者：**激光雷达

**相关模块：**Map Control、Map、Radar类

### 路径规划

**主要参与者：**用户

**目标：** 打开/关闭机器人

**前置条件：**地图建立完毕

**启动：**用户操作正确

**场景：**

1. 从用户指令中解析出目标地与目标物
2. 对目标地进行路径规划
3. 机器人开始运动
4. 运动过程中检测障碍，若出现新的障碍则重新进行路径规划并修改地图
5. 到达指定目标点

**优先级：**高

**何时可用：**机器人关闭/开启时

**使用频率：**中

**次要参与者：**机器人运动模块，激光雷达

**相关模块：**Path Control、Path Move Control、Path、Map、Move Control类等。

### 目标物体识别抓取

**主要参与者：**用户

**目标：** 机器人在路径规划结束后成功识别出用户指定物体的位置并抓取目标物

**前置条件：**路径规划成功

**启动：**路径规划完毕后启动

**场景：**

1. 从用户指令中解析出目标地与目标物
2. 对目标地进行路径规划
3. 到达目标地
4. 检测目标物体
5. 完成抓取

**优先级：**高

**何时可用：**机器人开机时

**使用频率：**中

**次要参与者：**机械臂

**相关模块：**Grasp Control、Kinect、Robot Arm类

## 非功能需求

### 性能指标

* 机器人的主动控制响应时间应在0.5-1s内；
* 当机器人移动过程中在其激光雷达范围内突然检测到有障碍物接近，则应在1-1.5s内减速至停止；

**对应设计：**避障模块中，当雷达检测到障碍物时，机器人停止运动，重新规划路径。

* 在机器人行驶过程中，应当保持在0.15m/s左右的速度匀速行驶，在没有障碍物的情形下减少变速运动；

**对应设计：**移动控制模块进行控制。

* 对于已知起点和终点位置的简单环境，机器人路径规划应在100s之内完成起点至终点的行进；对于未知的复杂环境，机器人应在环境感知过程中缓慢接近终点，预计在400s之内完成行进；

**对应设计：**实现最短路径算法，加快任务完成速度。

* 当目标物出现在机器人感知识别的范围内时，机器人应在1-3s内选择合适的路径动态接近目标物；
* 当机器人抓取目标物时，应该确保抓取目标物行进的过程中不掉落，能够适应最少0.2m/s的行进速度；

### 系统可用性

* 用户能够正常操作控制系统，确保起点、终点的合理规划，如果出现起点和终点不可到达的情况，系统应及时提醒用户，并且不进行任何动作；
* 系统操作应当简洁，确保每个用户学习后能够成功上手；

**对应设计：**界面简洁，操作通俗易懂，详见第5节

### 系统可移植性

* 机器人每项功能代码应分为独立模块，代码耦合度不应过大；
* 在确保代码的独立性后，新增功能时保证原有代码的可用性；
* 提供适当的接口，保证BUG的修复不影响其他功能；

**对应设计：**代码运行环境为Ubuntu16.04等信息。

### 系统安全性

* 启智ROS机器人是室内机器人，在此环境之外运行可能会损坏机器人；
* 启智ROS机器人的工作平面需要能够承载不小于40kg的重量。如果表面太软，则机器人可能卡住，运动受阻。启智ROS机器人原则上在水平平面上工作，坡道坡度不大于15度， 坡道倾斜度过大可能导致倾覆；
* 启智ROS机器人不具备防水功能，在任何情况下，启智ROS机器人都不应该与雨水，雾，地面积水以及任何其他液体接触，否则可能导致电路和机构损坏；
* 启智ROS机器人设计工作温度为15°C到35°C之间，使用中务必远离明火和其他热源；

**对应设计：**在机器人运行前，需要对物理环境进行检查，保证环境不会对系统造成危害。在运行过程中，实现一个紧急制动功能，用于应对突发情况。

### 系统效率

* 在用户设定好起点和终点时，系统应当在5s内规划出一条最优路径，并能在1s内从静止加速到0.1m/s，即将抵达终点时，应保证机器人在0.2m左右开始减速，同时准备伸展机械臂，在抵达终点的同时，能在5s以内快速抓取目标物，返回起点；

**对应设计：**算法优化，设置合适的运行参数，抓取目标物时对位姿参数进行调整，方便机械臂有效抓取目标物。