2022 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

2022 Intel Cup Undergraduate Electronic Design Contest

- Embedded System Design Invitational Contest

作品设计报告

Final Report

Intel Cup Embedded System Design Contest

报告题目：Lighting--视障用户购物辅助系统

学生姓名：李达 闫采奕 骆畅然

指导教师：何春

参赛学校：电子科技大学

2022 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

参赛作品原创性声明

本人郑重声明：所呈交的参赛作品报告，是本人和队友独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果，不侵犯任何第三方的知识产权或其他权利。本人完全意识到本声明的法律结果由本人承担。

参赛队员签名：

闫东奕　　骆畅然，　　李达

日期：　2022 年 7 月 20 日

Lighting---视障用户购物辅助系统

摘要

视力障碍人群由于客观身体条件限制，其活动范围往往受到极大的制约。本项目以超市为目标场景，以 GNS-V40 边缘计算主机为中心设备，提供服务支持，以树莓派为可穿戴分布设备的处理器。利用 IMU 和 UWB 室内定位结合数字孪生建模提供导航支持、以摄像头采集图像提供文字扫描和条码识别、以耳机和麦克风采集并输出语音提供智能的语音交互操作，实现无障碍购物的全程辅助。该项目充分发挥了边缘计算算力强大，传输时延低的特点，同时作为公共辅助设施，能够对视障人群购物的全过程提供引导，有一定的使用和推广价值，并为其他场景的类似应用提供了示范和参考。

关键词：边缘计算，视力障碍辅助，语音交互，室内定位以及导航

Lighting---Shopping assistance for visually impaired users

ABSTRACT

Due to the limitation of objective physical conditions, the range of activities of visually impaired people is often greatly restricted. This project takes the supermarket as the target scene, uses the GNS-V40 edge computing host as the central device to provide service support, and uses the Raspberry Pi as the processor of the wearable distribution device. Using IMU and UWB indoor positioning combined with digital twin modeling to provide navigation support, capture images with cameras to provide text scanning and barcode recognition, and use headphones and microphones to capture and output voice to provide intelligent voice interactive operations to achieve full assistance for barrier-free shopping. This project takes full advantage of the strong computing power of edge computing and low transmission delay. At the same time, as a public auxiliary facility, it can guide the whole process of shopping for visually impaired people. The application provides demonstration and reference.

Key words: Edge computing, Visually impaired assistance, Voice interaction, Indoor positioning and navigation

目　录

第一章　项目背景.....1
1.1　立项背景和意义.....1
1.2　国内外发展现状.....1
1.3　项目特点及优势.....2
第二章　项目概述.....3
2.1　研究开发内容.....3
2.2　系统工作流程.....3
2.3　系统架构.....3
2.4　硬件设计.....4
2.4.1　GNS-v40.....4
2.4.2　树莓派.....5
2.4.3　LinkTrack 定位模块.....5
2.4.4　IMU.....6
2.4.5　摄像头.....6
2.5　软件流程.....6
第三章　关键技术和特色.....7
3.1　数字孪生.....7
3.1.1　UWB 定位.....7
3.1.2　IMU 角度识别.....9
3.1.3　路径规划与导航.....9
3.2　通信协议与交互.....**错误!未定义书签。**
3.3　终端进程协同.....10
3.3.1　节点管理进程（master）.....11
3.3.2　图像数据进程（photo）.....11
3.3.3　主进程（rasp）.....11
3.3.4　音频监听进程（listen）.....12
3.3.5　音频输出进程（speaker）.....13
3.4　语音服务模块.....13
3.4.1　语音识别 ASR.....13
3.4.2　语音合成 TTS.....14
3.4.3　基于 Intel oneAPI 的加速.....15
3.5　视觉检测模块.....16
3.5.1　文字识别.....16
3.5.2　条码识别.....17
第四章　系统测试.....18
4.1　硬件测试.....18
4.2　功能测试.....19
4.3　测试数据.....19

第五章 附录.....19
5.1 程序清单.....19
5.1.1 语音识别 ASRT.....19
5.1.2 语音合成 TTS.....20
5.1.3 文字识别 OCR.....21
5.1.4 条码扫描.....21
5.1.5 室内导航.....21
5.1.6 树莓派端.....21
5.2 程序源码.....22
5.2.1 语音识别 ASRT.....22
5.2.2 语音合成 TTS.....24
5.2.3 文字识别 OCR.....26
5.2.4 条码扫描.....27
5.2.5 室内导航.....28
5.2.6 树莓派端程序.....36

表目录

表 1 国内外市场常见的几种视听转化类设备.....2
表 2 GNS-v40 硬件信息.....5
表 3 树莓派硬件信息.....5
表 4 IMU 硬件信息.....6
表 5 硬件测试结果.....18
表 6 OneAPI 优化前后速度和准确率.....19

图目录

图 1 系统设计示意图.....4
图 2 整体硬件流程图.....4
图 3 摄像畸变示意图.....6
图 4 软件流程图.....7
图 16 UWB 三边定位原理示意.....8
图 17 本场景 UWB 定位原理示意.....9
图 18 计算流程图.....9
图 19 室内导航示意图.....10
图 5 帧结构示意图.....11
图 6 导航功能进程交互示意图.....11
图 7 报文结构示意图.....12
图 8 主进程运行示意图.....12
图 9 音频监听程运行示意图.....13
图 10 音频输出程运行示意图.....13
图 11 语音识别模型训练和推理过程.....14
图 12 TTS 模型框架图.....15
图 13 文字识别.....17
图 14 文字识别效果示意图.....17
图 15 EAN-13 码的结构与编码方式.....18

第一章　项目背景

1.1　立项背景和意义

我国是世界上视力障碍人群数量最多的国家，占世界总数的 18%-20%，据《柳叶刀》杂志数据显示：2019 年，中度、重度视力障碍和失明的整体患病率分别为 3.23%、0.33%和 0.61%，相当于分别约有 4592 万、467 万和 869 万患者，其中每年新增的盲人数量高达 45 万，并且其增长数据正在逐年攀升。

我国的残疾人支持体系不够成熟，公共配套设施不完善，相关场所对视力障碍人群的支持不到位，其活动往往被限定在熟悉的区域，限定在简单的活动场景，无法参与较为复杂的社会活动。目前市面上常见的视力障碍辅助设备如：导盲犬，导盲杖等，前者训练成本高，数量有限，很难普及；而导盲杖等助盲设备存在功能单一、结构简单、极易损坏等缺点。部分的电子视障辅助设备也只能泛化的提供如障碍物提示，人脸识别等功能，只能扮演辅助角色而无法提供特定复杂场景的解决方案。

随着科技的进步，机器学习，计算机视觉，语音处理，文字识别等技术的蓬勃发展为项目的实现提供了算法基础；边缘计算主机的部署更靠近用户侧，提供了低时延的强大算力；深度相机等先进的图像采集设备让环境的感知能力更上一层楼；智能的语音交互加强了人与机器的联系。这些都使得更加智能的视力障碍辅助系统有了实现的基础。

本项目作为公共辅助设施，以超市为特定目标场景，以 GNS-V40 边缘计算主机为中心设备，利用 IMU、UWB 基站定位模块、耳机、摄像头、麦克风、树莓派等构成可穿戴分布设备，实现无障碍购物的全程智能辅助，让视障人群也能享受到正常人群的购物体验，同时该项目也能为其他场景的应用提供示范效应，促进全社会对视障人群的关注，推动视障人群公共辅助设施的建设。

1.2　国内外发展现状

我们的项目对应于视障辅助智能产品中的视听转化类，是一种可穿戴的视觉辅助产品，能够实时地将视觉信息转换成语音，一般含图像采集设备、中央处理系统，语音输出设备。图像采集设备将其拍摄到的信息传递至中央处理系统，借助计算机视觉和机器学习技术，经芯片处理后将音频数据通过语音输出设备播放出来。是一种具有自动识别障碍物和交通标志、导航和语音提示等一种或多种功能，能够引导盲人顺利到达目的地、进行文字识别辅助阅读等多种功能的盲人辅助工具。目前市场上视听转化类智能助视器发展速度较快。

如 1.1 中所述，我国虽然视觉障碍人群数量庞大，但是一直以来类似的电子视障辅助系统未能形成完善的产品并进行大规模的市场推广，我国视障、盲用辅具企业长期以低科技、低成长性企业为主。《残疾人基本辅助器俱指导目录(2020 版》中涉及到视障人群的助视器限于放大镜、低视力眼睛、电子助视器等，呈现低技术附加值特点。视障群体能使用的智能产品更是极度稀缺，相关自主知识产权较少。

目前国内外市场常见的几种视听转化类设备如下所示：

表 1 国内外市场常见的几种视听转化类设备

国家	公司	产品	特点	缺点
瑞士	Eyra	Horus	由骨传导耳机和高清摄像头、袖珍 AI 处理器组成，外观形似运动耳机，能进行物体的识别和辅助阅读。	未形成产品并进行市场推广，仅停留在原型机。
多伦多	eSight	eSight3	包括两个高清彩色显示器，并带有根据用户的特定处方量身定制的处方镜片。使用高对比度 OLED 屏幕，1024x768 分辨率和 37.5 度视场。具有 HDMI 和 USB 输入，蓝牙，Wi-Fi 和可移动 SD 卡。	只适用于视觉模糊的人群不适用于完全失明，且售价高达 1 万美元（约合人民币 6.4 万），仅售出了 1000 副左右。
以色列	orCam	OrCam MyEye	可解析视觉输入元素，捕捉任何平面上涵盖的印刷或数码文字，通过磁铁附着在用户自己的眼镜框上；更适合在“阅读”报纸、菜单或书本的场景下使用，能够进行人脸识别，通过已存储的信息告知用户其身份。	设备续航能力差，识别能力有限，只能识别人脸和简单物体，售价为 3500 美元（约合人民币 2.2 万）。
中国	视氪	避障眼镜	首家将三维立体信息获取技术应用到视觉辅助领域，并首创了立体声交互系统，用户通过特殊编码的立体声可以在脑海中“虚拟"呈现环境信息；更适合障碍物和通路检测、场景检测和精准定位导航。	功能单一，仅能进行物体的检测和导航，售价 1.9 万元人民币。

1.3 项目特点及优势

1. 场景固定：该项目与 1.2 中介绍的国内外产品相比，后者针对简单泛化的场景，无法提供复杂场景的支持，我们将主要针对于超市这一公共区域，以边缘计算主机作为中心设备，提供数据和算力，为分布式的辅助设备提供支持。为视障人群的购物提供全流程的决策支持。

2. 用户群体大：现存的各种视障辅助设备都聚焦于完全失明者，而忽略了为视力存在缺陷的用户提供更加优质的生活。我们的产品覆盖的用户面包括不同程度的视力障碍及盲人群体，更加广泛。

3. 价格优势：我们将采用通用的算法和硬件设备，以中心+分布的形式，降低单个用户终端的设备成本，从而降低系统造价，便于进行市场推广，更好的服务视障群体。目前市场还处在“蓝海”阶段，该产品将进一步丰富国内的视障群体智能辅助设备市场，填补相关功能的空缺。

第二章　项目概述

2.1　研究开发内容

本项目聚焦于开发一套部署于商场的视力障碍用户购物辅助系统，系统整体主要有两个模块组成，一是边缘计算主机，在该主机上部署 AI 处理算法实现语音识别，视觉检测以及路径规划功能，二是用户眼镜终端，用户终端主要进行相机数据、语音数据、路径行走数据的采集，并将数据上传到边缘计算主机进行处理，并根据处理结果执行相应的操作。最后系统能够实现视障用户进入商场并佩戴用户眼镜终端后，可以利用语音与系统进行交互，并且用户眼镜终端利用后台的边缘计算主机可以帮助视障用户顺利前往相应的购物区，购买需要的物品。

2.2　系统工作流程

当用户进入目的场景后，需要将分布式设备佩戴在身上、连接好麦克风，使用特定词语（如木子木子）唤醒语音助手以便提供服务。语音助手被唤醒后，将会询问用户需要的服务，然后做出反馈。

首先是基础购物功能，用户需告知语音助手所需要购买什么商品或需要前往的货柜区域，助手将会确定其所在的具体位置，并规划路径。助手确定好路径后，需要不断获取用户所在位置，根据位置确定用户行进的方向，告知用户前进的方向、直行或者转向。如有偏航，助手会提醒用户，并重新规划路径。

当到达目的位置后，语音助手会进行提醒，然后等待用户的下一命令。用户此时可以选择拿起某样商品，对助手下达文字识别命令，助手将识别当前摄像机角度下能够被看到的文字信息。识别完成后将通过语音播报给用户。此时，用户还可下达条码识别服务，对所持商品的条码进行扫描，进而得到商品的价格信息。如果用户所持商品条码被遮挡或角度无法满足识别要求，助手将语音提醒用户调整角度，如将商品翻转等。识别条码信息后，将数据播放，从而使用户判断该商品是否满足自己的要求。如不满足，用户可更换其他商品重复命令。用户如有其他需要，可下达导航命令，告知助手下一种所需的商品或下个货柜区域，助手继续导航直至用户的所有购物需求都被满足。商品选择完成后，用户告知助手选择结束，进行结账，助手将引导用户前往收银区域进行结账，购物结束。

2.3　系统架构

本系统实现上需要使用硬件设备主要为 IMU、UWB 基站定位模块、耳机、摄像头、麦克风、树莓派、边缘计算主机（GNS-v40），本系统工作时，需要主机和用户终端协调。为体现平台强大的算力优势及边缘计算、靠近用户侧低时延的特点，所有的运算均部署在板端，降低了传输带来的延迟。整体工作流程如下：

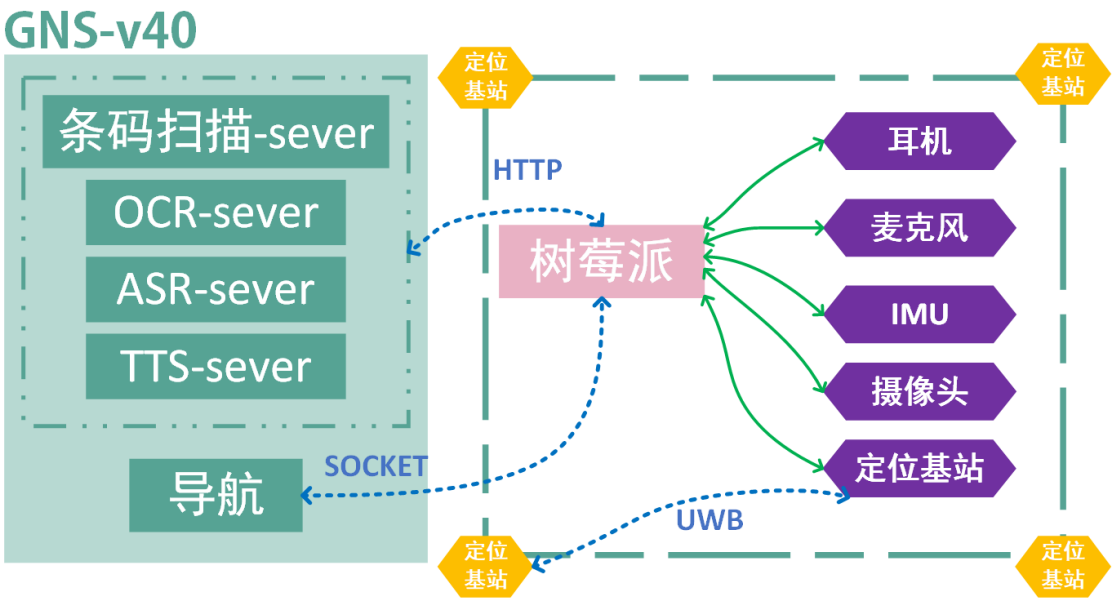

图 1 系统设计示意图

2.4 硬件设计

其中硬件框架设计示意图如下：

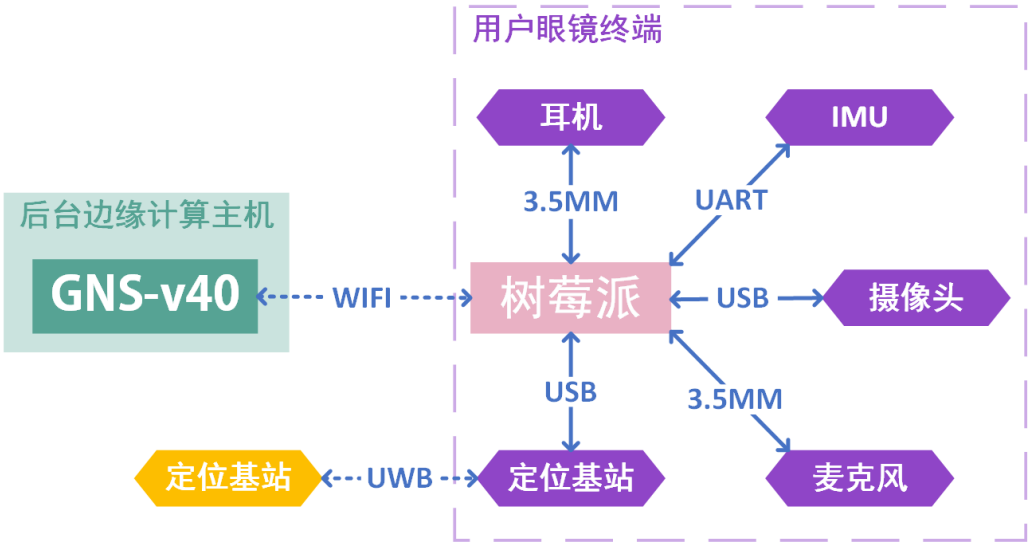

图 2 整体硬件流程图

系统在工作时，用户眼镜终端通过 socket 和 http 接入与主机通信，用户眼镜终端主要承担数据采集功能，基本不承担计算任务。终端主要通过外设，如麦克风，摄像头来采集用户语音、外界环境画面等数据，而树莓派将数据上传到主机，主机处理后将结果反馈到 AI 眼镜终端，AI 眼镜终端根据处理结果使用语音跟视障用户进行相应的交互。同时在场地四周布置有定位基站，通过 UWB 与终端上连接的移动基站通信，从而确定用户的位置，以便主机进行路径的规划。

2.4.1 GNS-v40

本设备为参赛的核心设备，提供高性能、低功耗、配置丰富的硬件选型，可满足各类边

缘 AI 应用场景对算力、I/O、扩展性的等要求。采用信步边缘 AI 计算平台，搭载 Intel OpenVINO 工具套件，以及来自算法合作伙伴、遍及广泛用例的可商用 AI 算法，帮助开发者快速落地及部署边缘 AI 解决方案。

表 2 GNS-v40 硬件信息

CPU	Intel CoreTM i5-1135G7 CPU, 4 Cores, up to 4.2 GHz
GPU	Intel Iris Xe Graphics, 1.30 GHz, 80 EU
内存	16GB, DDR4, 2666 MHz
储存	128G M.2
操作系统	Windows 10 64-bit, Linux
Wifi/ BT	Intel AC 7260, 2.4GHz/ 5GHz, Bluetooth 4.0
电源适配器	DC 12V-10A 适配器
设备尺寸	258*150*56mm
设备外设接口	6*USB 3.0, 2*USB 2.0
	2*HDMI, max resolution up to 4096*2160@30Hz
	5.1 Channel HDA Codec, 1 * Line-Out + MIC 2in1 3.5mm Jack
	8*GPIO, 1*Power Button, 1*GND Hole

2.4.2 树莓派

为解决分布式终端数据汇总、传输问题，我们采用树莓派 4B 完成终端的数据控制任务。树莓派 4B 体积小，重量轻，散热好，佩戴在身体上不会带来负担，也不会影响用户正常行动。同时树莓派开发简单便捷，外设接口丰富，如多个 USB 接口、3.5mm 音频输入接口，摄像头接口等，能够满足本系统对于多个外设数据采集汇总的需求，非常适合作为终端设备。

表 3 树莓派硬件信息

CPU	64-位 1.5GHz 四核
内存	8GB DDR4
外设接口	2*USB 3.0，2*USB 2.0
	2*HDMI
	3.5mm 音频接口
BT	Bluetooth 5.0
网卡	双频 80.2.11ac5G/2.4G

2.4.3 LinkTrack 定位模块

为了在商场这一室内场景进行精确定位，获取用户的具体位置，便与后续的路径规划与导航工作，我们选择 LinkTrack 作为定位设备。

LinkTrack 是一款基于 UWB 技术的多功能系统，支持 1、2、3 维定位，典型 1 维、2 维定位精度 10cm，定位频率高达 200Hz，性能优异；最远传输距离 80m，满足应用场景需

求；支持从 3.5GHz 到 6.5GHz 一共 6 个射频频段，发射增益可调，范围为 0~33.5dB，可根据不同场地大小更改频段和发射增益以取得较好的定位效果；其基站容量多达 120 个，标签容量多达 200 个，有利于后续分布式终端的产品化。并且 LinkTrack 大小仅为 60.3mm*29mm*9mm，重量为 50g，体积小，重量轻，可以轻松安装在头戴式设备上，不会造成负担。

2.4.4 IMU

为了提高系统角度识别精度，我们加设了正点原子 ATK-IMU901 角度传感器获取相关数据。其 IMU 采用自主研发姿态解算算法，X\Y 轴静态精度 0.05°，Z 轴(航向角)在无磁力计情况下，长时间无漂移，精度高。

表 4 IMU 硬件信息

角度精度	X、Y 轴:静态 0.05°动态 0.1°，Z 轴:0.5°
角度范围	角度:X、Z±180°，Y ±90°
工作电压	3.3V~5V(推荐 5V)
工作电流	<25mA
串口波特率	TTL 电平(3.3V) ，2400~921600 可调
尺寸	15.57mm×16.20mm ×2mm

2.4.5 摄像头

由于一维码没有纠错机制，拍摄影像的畸变对于条码扫描结果有较大影响。

为达到良好的文字识别和条码扫描效果，我们选择杰锐 500w_2.8mm 焦距无畸变广角摄像头，支持拍摄范围 90°，拍摄范围较大，使得视障人群在无法感知周围环境和摄像头具体位置时也能够成功拍摄到目标商品；支持手动变焦，能够较好的适应超市环境。同时其分辨率高达 2592*1944，能够更好的支持条码和文字的扫描。

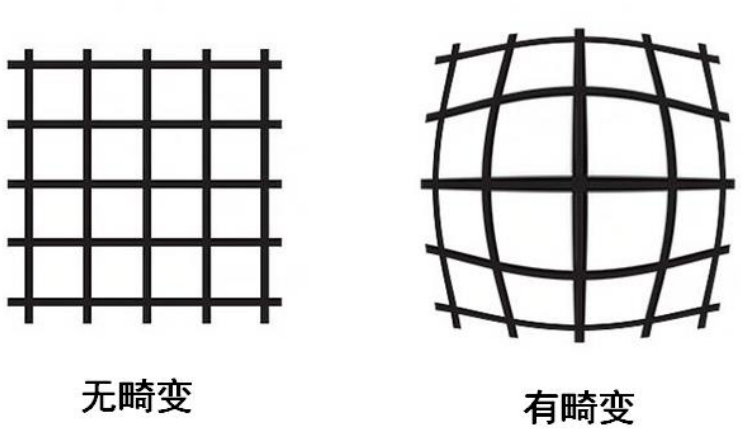

图 3 摄像畸变示意图

2.5 软件流程

本系统采用中心分布式的设计，主机和终端有各自的运行逻辑与进程，进程之间相互配合，共同完成整个购物过程的指引。具体过程如下所示

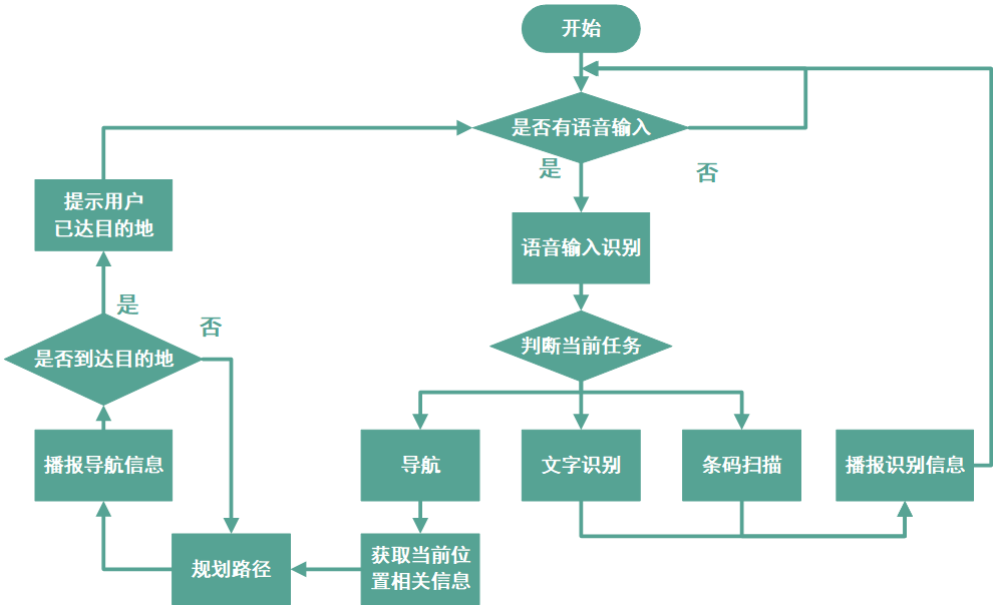

图 4 软件流程图

由上图可知，系统启动后会时刻保持语音监听，收到语音数据后，进行识别得到文本信息。根据用户语音指令判断当前任务。

如果为导航指令，树莓派与主机建立 socket 连接，发送位置和目的地信息，主逐步返回路径规划，根据语音提示引导用户前往目的地，到达目的地之后会进行提示，结束 socket 进程，然后继续保持监听。

如果为文字识别或者条码扫描指令，系统则发送对应图片和 HTTP 请求，识别或扫描完成后将获得信息，再通过 HTTP 请求获得语音，播报告知用户，并继续保持监听。

第三章　关键技术和特色

3.1　数字孪生

本部分完成对整个商场空间的建模、用户终端的定位与实时显示。通过数字孪生将虚拟与现实的连接，能够实现实时监控终端位置和具体情况。

数字孪生是充分利用物理模型、传感器更新、运行历史等数据，集成多学科、多物理量、多尺度、多概率的仿真过程，在虚拟空间中完成映射，从而反映相对应的实体装备的全生命周期过程。数字孪生是一种超越现实的概念，可以被视为一个或多个重要的、彼此依赖的装备系统的数字映射系统。我们通过建模与微型 UWB 定位基站，将整个商场的场景和用户一同转移入虚拟空间中，具有良好的实时性、灵活性。

3.1.1　UWB 定位

在当前应用场景下，终端设备由于受到移动、商场货架或者楼层本身遮挡等影响，发送信号和接收信号都大幅度衰减，室内定位导航难度极大。当前可行的室内定位解决方案有 WIFI 定位、红外定位、超声波定位等等。但是均不适用超市建筑复杂，遮挡物较多、定位目标快速移动的特性。最终我们选择了 UWB 超宽带定位技术。

超宽带技术是近年来新兴一项全新的、与传统通信技术有极大差异的通信无线新技术。，

利用纳秒至微秒级的非正弦波窄脉冲传输数据。UWB 具备时间分辨率高、穿透力强、功耗低、抗多径效果好、安全性高等优点，因此常被应用于通信与定位领域，尤其是在 GNSS（如 GPS、BDS、Glonass、Galileo）信号覆盖不到的场合。UWB 定位原理与 GPS 相似，它利用事先布置好的已知位置的锚节点和桥节点，与新加入的盲节点进行通讯，从而获取新加入节点的具体坐标信息。同时，它不需要使用传统通信体制中的载波，而是通过发送和接收具有纳秒或微秒级以下的极窄脉冲来传输数据，从而具有 3.1~10.6GHz 量级的带宽。

本部分我们使用的定位基站为 LinkTrack。LinkTrack 是一款基于 UWB 技术的多功能系统，支持 LP（局部定位）、DR（分布式测距）、DT（数传）三种模式，支持配置为标签、基站等多种角色。LP 是支持定位、导航、授时与通信（PNTC）一体化功能的实时定位模式，分为标签、基站、控制台三种角色。标签实时测量并进行坐标解算，输出自身测距、坐标等信息，基站与控制台实时输出所有标签的定位信息。典型 1 维、2 维定位精度 10cm，定位频率为 200Hz。

通过测量移动基站到固定基站的的飞行时间，乘以光速后，移动基站可以获得到固定基站的距离。通过到多个固定基站距离与参考基站距离的坐标，可以列出多组球面方程，进而由数学方法可以求解出标签的坐标。

当系统中存在多个标签时，按照一定的机制，能够使所有标签同时工作，不受影响。每一个标签独立输出自身的定位信息，以及从各个基站、控制台发送过来的数传数据；每一个基站、控制台输出信号范围内所有标签的定位信息，以及从各个标签发送过来的数传数据。

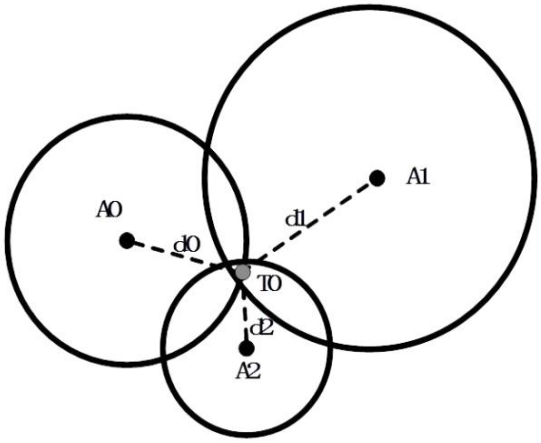

图 5 UWB 三边定位原理示意

在本场景，我们架设了 4 个固定基站（A1~A4）、一个移动基站 T0 与一个控制台 C0。定位开启后，T0 分别测量到 4 个基站的 TOF（飞行时间），乘以光速后获得 T0 到各个基站的距离 d0~d3，然后再进行数学计算求解自身坐标。为了减少用户移动带来的抖动，解算好的坐标会进行滤波，然后通过通信接口进行输出。同时，通过 UWB

无线电报，T0 将自身的坐标发送给信号范围内的基站、控制台，从而 A0~A3、C0 均能输出 T0 的坐标信息。

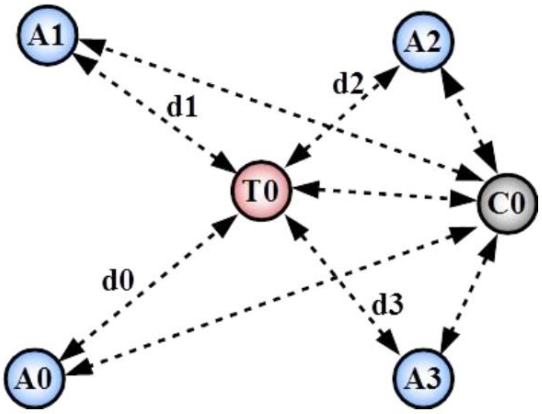

图 6 本场景 UWB 定位原理示意

计算流程如下所示：

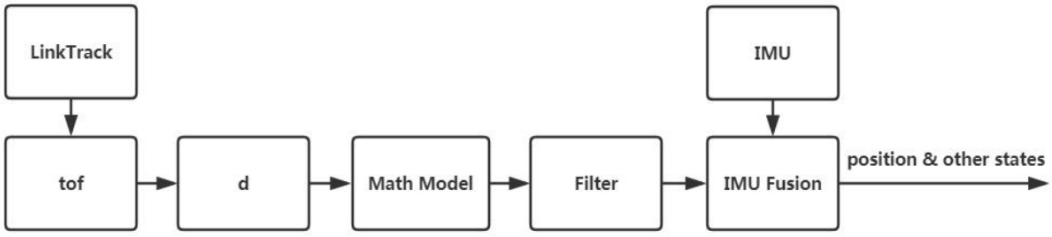

图 7 计算流程图

3.1.2 IMU 角度识别

LinkTrack 定位基站能够使用陀螺仪完成角度的获取，由于陀螺仪测量角度时使用积分，会存在积分误差，若积分时间 Dt 越小，误差就越小。不断提高采样频率，就可以使积分时间 Dt 变小，降低误差。同样地，提高陀螺仪传感器的采样频率，即可减少积分误差，能够满足精度需求的陀螺仪采样频率为 8kHz 左右，但是基站无法达到这一要求。并且，LinkTrack 没有对于角度积分的校正算法，角度误差较大。为了解决这一问题，我们选择 IMU 获取角度信息。

IMU 可获得载体的姿态、速度和位移等信息，被广泛用于汽车、机器人领域，也被用于需要用姿态进行精密位移推算的场合，如潜艇、飞机等惯性导航设备中。惯性测量装置包含三组陀螺仪和加速度传感器，分别测量三个自由度的角加速度和线加速度，通过对加速度的积分和初始速度、位置的叠加运算，得到物体在空间位置中的运动方向。并且 IMU 可以对角度识别进行滤波处理，消除用户行进过程中的抖动问题。

定位基站和 IMU 二者相互配合便能够确定用户在空间中的位置和运动方向，为实时导航提供了可能。

3.1.3 路径规划与导航

通过对模拟环境的建模，在获得定位信息和期望目的地后，可实现实时导航功能。

为了满足主动导航指引的要求，我们使用 socket 代替 http，作为导航 sever 和终端的通信协议。http 协议是一种自身不对请求和响应之间的通信状态进行保存的协议，即无状态协议。好处是：更快的处理更多的请求事务，确保协议的可伸缩性。正常在发送 http 时，都

需要建立 TCP 的连接，再发送报文，那么时间大部分都会消耗在建立和断开连接的过程中。并且 http 使用了请求 - 应答模式，客户端主动发起请求，服务器被动回复请求。由于服务器无法主动向终端发送导航信息，行进过程中的引导便不能够进行，所以 http 无法满足系统设计要求。而 socket 传输数据为字节级，传输数据可自定义，数据量小，传输数据时间短，性能高，能够降低导航 sever 行进指令的传输时延，降低用户的危险性。同时 socket 客户端可以主动发送信息，适合于客户端和服务器端之间信息实时交互，满足要求。

为了实时监控各个终端，避免意外发生、赋予系统较大的灵活性，我们引入了数字孪生的概念，将整个环境在虚拟场景里建模。建模完成后将其抽象成为二维网格，使用寻路算法进行路径规划。根据 IMU 获得的用户的朝向、定位基站确定的用户位置，可以得出用户的下一步移动方向，之后对用户进行实时语音指示，完成导航任务。

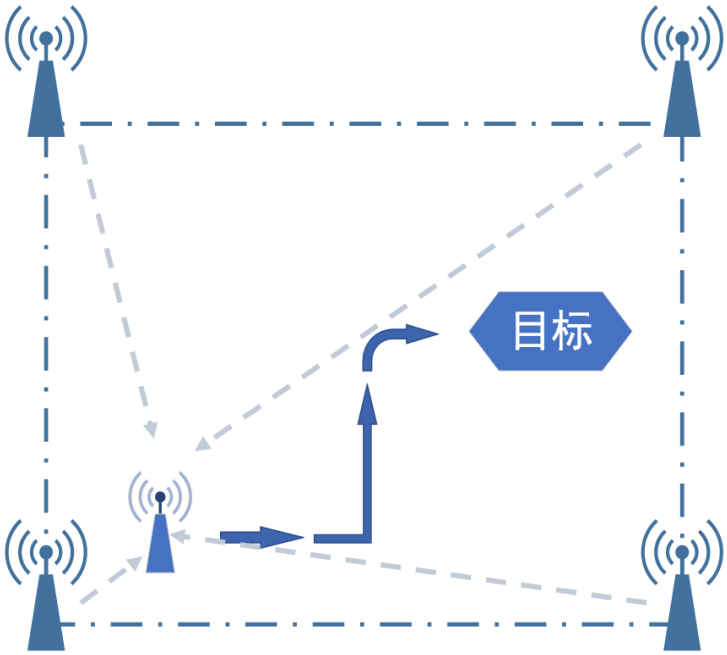

图 8 室内导航示意图

为降低用户行进过程中由于转弯角度不理想，没有朝向期望方向导致的长时间方向调整，我们对于规划出的路径进行筛查，选择转弯最少，直行最多的路径，大大提高了用户的体验，减少了花费的时间。同时，为了解决视障用户由于自身无法确定移动方向正确性导致的偏航问题，我们根据用户的偏离情况，实时重新规划路径，而不是引导用户回到原路径，保障了用户能够到达目的地。

3.2 终端进程协同

为了采集、汇总、与传输数据，并能够和服务端顺利协作，树莓派上运行了 5 个进程，分别为节点管理进程、导航主进程、音频监听进程、音频输出进程和图像数据进程。所有进程在开机时一同启动，通过下图帧结构进行信息、指令的交互。

From	To	Data/Flag

图 9 帧结构示意图

- From：源进程
- To：目的进程
- Data：相关数据/标志位

进程间根据源、目的进程，即 from、to 字段中的信息来进行交互，帧由 from 进程发送给 to 进程，该进程收到帧后根据本进程通信规则读取改帧。

以导航指令为例，进程间的交互如下所示：

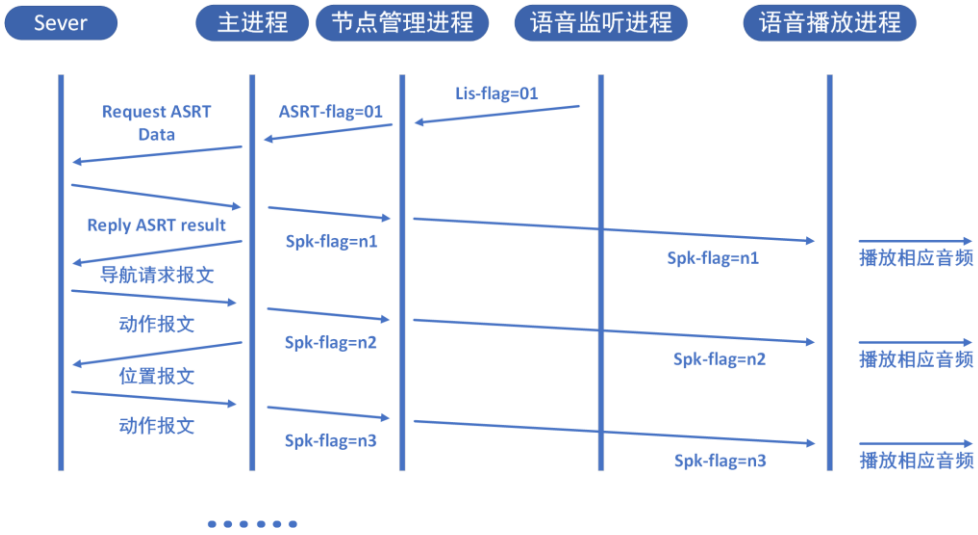

图 10 导航功能进程交互示意图

3.2.1 节点管理进程（master）

作为所有进程之间协调控制的进程，掌握着所有进程之间的调度。在此进程中记录着指令和状态的标志位，如语音监听进程中的 Lis-flag，标志着此时是否有语音需要处理。通过标志位，此进程便可通知协调其他进程完成相应的工作。

3.2.2 图像数据进程（photo）

此进程和其他进程间联系最为微弱，在开机后此进程立即运行，直接调用摄像头采集所有信息将其发送至后台边缘计算主机，当需要时后台主机才会读取图片进行处理。

3.2.3 主进程（rasp）

此进程作为和边缘服务器交互的进程，承担着请求各种服务，接收服务器处理后的信息，并与其他进程交互的任务。

首先，此进程与导航服务器的交互，采用了特定的报文结构，如下所示

图 11 报文结构示意图

1. Type A：导航请求包，包中含有类型字段、当前位置字段和目的地址字段。此包由树莓派发送至导航服务器，用于通知当前用户请求导航服务，导航服务器可根据坐标规划全程路径，并根据用户朝向的角度计算下一步的行进动作。
2. Type B：动作包，包中含有类型字段和动作字段。此包由导航服务器发送至树莓派，用于通告当前用户应当的行进动作。
3. Type P：位置包，包中含有类型字段、当前位置字段和当前的语音播报的状态，即语音是否播放完毕。此包由树莓派发送至导航服务器，用于告知服务器用户当前具体状态，便于服务器判断发送动作包的时机。

1. Pos：用户当前位置，格式为[x,y,theta]，即用户当前坐标(x,y)及朝向角度 theta。
2. Act：用户的行进动作，U：直行；D：后退/后转；L：左转；R：右转。
3. Des：用户目的地的 flag，1：收银台；2：零食区。
4. Dis：用户终端语音播放情况，True：已播放完毕；False：未完毕。

具体通信过程为：主线程获得语音识别结果后提取关键字，得到用户下达导航指令的信息，此时主线程一方面需要请求节点管理线程，需要播放相关语音回复，完成与用户的交互。具体操作为，发送目的为节点管理进程，data 为 Spk-flag=n 的帧，n 为特定音频的编号。的另一方面其将用户此时的位置和目的地信息通过 Type A 报文发送给导航服务器。之后等待导航服务器的回复 Type B 的动作报文，获取到之后再次请求节点管理线程，播放导航指示的行进动作，如：左转，直行，右转等等。

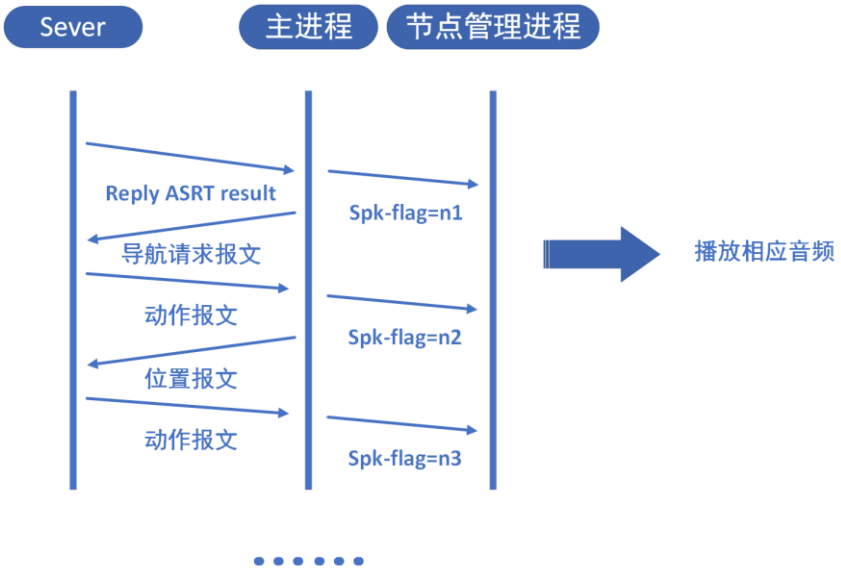

图 12 主进程运行示意图

3.2.4 音频监听进程（listen）

此进程在开机时立即启动，通过判断麦克风此时音量是否达到阈值来判断用户是否在说话。音量低于阈值时，发送给节点管理进程的帧中 data 字段为 Lis-flag=00；一旦监听到用户在说话，此进程立刻录制该语音并将帧中 Lis-flag 置为 01.节点管理进程收到该帧后，存储音频监听标志位，并发送指令给主进程，过程如下所示：

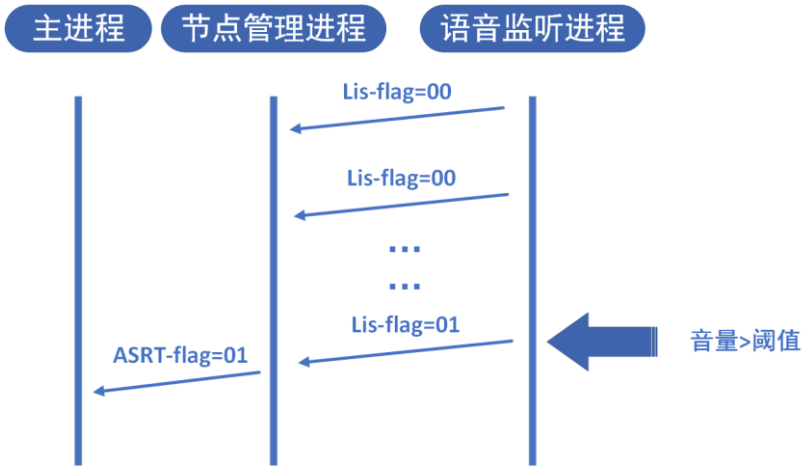

图 13 音频监听程运行示意图

3.2.5 音频输出进程（speaker）

此进程在开机时立刻启动，之后进入等待模式；当收到节点管理进程发送数据帧中 data 部份为音频播放标志位的帧后，根据编号播放对应音频。

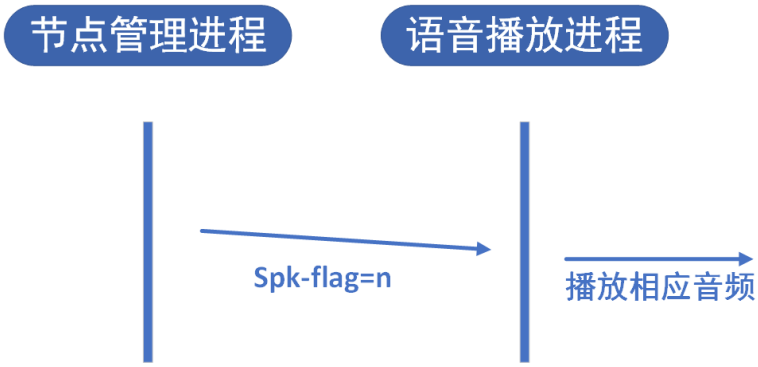

图 14 音频输出程运行示意图

3.3 语音服务模块

3.3.1 语音识别 ASR

在这部分，我们主要完成对用户语音的识别，以获取用户指令，满足与用户交互的需求。

ASRT 是一套基于深度学习实现的语音识别系统，全称为 Auto Speech Recognition Tool，其声学模型通过采用深度卷积神经网络（DCNN）和连接性时序分类（CTC）方法，使用大量中文语音数据集进行训练，将声音转录为中文拼音，并通过语言模型，将拼音序列转换为中文文本。

语音识别系统构建过程整体上包括两大部分：训练和识别。

训练通常是离线完成的，对预先收集好的海量语音、语言数据库进行信号处理和知识挖掘，获取语音识别系统所需要的“声学模型”和“语言模型”；而识别过程通常是在线完成的，对用户实时的语音进行自动识别。识别过程通常又可以分为“前端”和“后端”两大模块：“前端”模块主要的作用是进行端点检测(去除多余的静音和非说话声)、降噪、特征提取等；“后端”模块的作用是利用训练好的声学模型和语言模型对用户说话的特征向量进行统计模式识

别(又称“解码”)，得到其包含的文字信息，此外，后端模块还存在一个自适应的反馈模块，可以对用户的语音进行自学习，从而对声学模型和语音模型进行必要的“校正”，进一步提高识别的准确率。具体过程如下图所示：

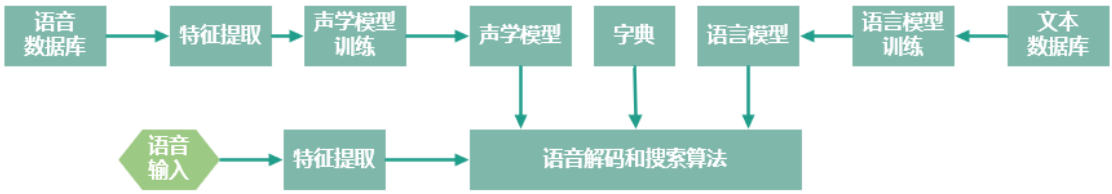

图 15 语音识别模型训练和推理过程

当 ASR-sever 接收到主机下达的识别指令后，将会对终端传输的语音数据进行识别，并返回文本结果，主机会根据结果判断用户的要求，以便做出下一步指示。

由于中文语音同音字同音词较多，对于特定词语、特定文字的识别效果不理想，不能满足精度要求。我们将对于特定词语的识别转为对拼音的识别，即识别特定发音。改正过后，对于同音字的容错率大大提高，识别准确率提升了 20%左右，达到了 80%，满足识别精度要求。

我们支持的语音识别关键词（拼音）为：

muzi：即“木子”，为唤醒词，监听到该词之后将会启动语音助手

qv：即“去”，作为导航服务的关键词，用户需要在唤醒语音助手后使用譬如“我要去买牛奶”等语句给系统下达指令。系统识别到该词后将会启动导航服务，获取位置信息与角度信息，规划路径，语音播报导航。

niunai、shouyin、shenghuo、lingshi：即“牛奶、收银、生活、零食”，作为各个区域的名称，语音识别服务器会提取到导航命令中的关键词信息，与货架名称库中的匹配，确定用户的目的地，为路径规划做好准备。

saomiao；即“扫描”，作为条码扫描的关键词，用户在到目的地后可以拿起商品，使用譬如“帮我扫描一下条码”等语句告知系统需要条码扫描服务。系统识别到该词后将会对摄像头采集的画面进行条码识别，获取商品信息，并使用语音播报的方式告知用户。

shibie：即“识别”，作为文字识别的关键词，用户为获取更多商品信息，可以使用譬如“帮我识别一下商品文字”等语句告知系统需要文字识别服务。系统识别到该词后将会对摄像头采集的画面进行文字识别，获取更多信息，并使用语音播报的方式告知用户。

3.3.2 语音合成 TTS

在这部分，我们主要使用 TTS（Text To Speech）来将商品扫描与识别后返回的信息、路径导航的指示、以及交互的回答等文本转换为语音播报给用户，以便给与用户流畅方便快捷的操作体验。

为实现功能，满足要求，我们选择 copui TTS 作为解决方案。copui TTS 是用于文本到语音转换的开源项目。它建立在最新研究的基础上，旨在实现训练，速度和质量之间的最佳平衡。copui TTS 带有预先训练的模型，以及用于测量数据集质量的工具，便于测试速度、准确度等等指标。

该项目提供了文本向频谱图的转换模型，如常见的：Tacotron，Tacotron2，Glow-TTS，Speedy-Speech。Align-TTS，FastSpeech 等模型，同时提供了端到端的模型 VITS。

我们最终选择使用 Tacotron2 模型，Tacotron2 是由 Google Brain 2017 年提出来的一个语音合成框架，模型主要由三部分组成：

1. 声谱预测网络：一个引入注意力机制（attention）的基于循环的 Seq2seq 的特征预测网络，用于从输入的字符序列预测梅尔频谱的帧序列。
2. 声码器（vocoder）：一个 WaveNet 的修订版，用预测的梅尔频谱帧序列来生成时域波形样本。
3. 中间连接层：使用低层次的声学表征-梅尔频率声谱图来衔接系统的两个部分。

其整体的框架如下所示：

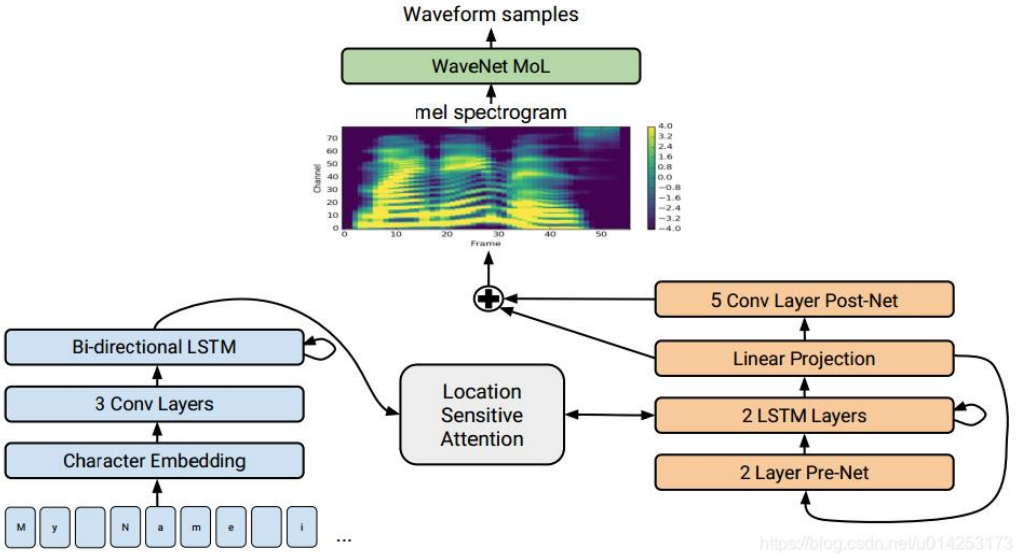

图 16 TTS 模型框架图

Tacotron 每个解码器步骤的输出帧数，称为“降低率”（r）。r 越高，模型产生相同长度输出所需的解码器步数就越少。因此，该模型实现了更快的训练收敛和更容易的注意力对齐，但是较大的 r 值也会生成更平滑的输出帧，因此会减少帧级细节。在推理时，由于域外的单词，长输入文本或目标语言的复杂性，Tacotron 这一使用注意力机制的模型容易受到注意力对齐问题的影响。

虽然使用较大的 r 可以更好地对齐，但是这种做法会降低预测帧的质量。我们在模型中结合 DDC（Double Decoder Consistency）的注意力集中机制，来对抗长文本输入或域外字符序列中存在的注意力对齐问题。DDC 由两个解码器组成，旨在解决注意力对齐和由缩小因子调整的预测帧质量之间的权衡，来得到性能和时间的平衡。

同时我们还使用了 GDT（Global Style Tokens）的无监督语音风格（如：副语言信息，重音、语调、语速、流畅度，重读，风格）控制方式，来达到更好的语音风格控制，这一方法最初也基于 Tacotron 开发的，能够获得更好的听觉效果。同时这也是一种无监督的算法，不需要事先进行数据集的标注。

模型训练采用 baker：标贝中文女生开源数据集，这是一套专门用于语音合成的标准普通话女声音库，共 10 小时，所有音频 wav 都有人工精标注，训练效果好。

3.3.3 基于 Intel oneAPI 的加速

oneAPI 是一个跨架构的编程工具，旨在简化跨 GPU、CPU、FPGA 和 AI 加速器之间的编程，可以与英特尔自身设备，或其他厂商的芯片配合使用，以优化工作负载。oneAPI 是一个跨平台的编程接口，客户可以针对哪种特定的架构来最优化、最高效地加速处理程序，并支持 AVX-512 和 DL Boost 这些英特尔的特定指令。

在 ASR 和 TTS 中，我们分别采用了 TensorFlow 和 PyTorch 的框架，而英特尔为这两种

主流的机器学习框架提供了支持，即：Intel Optimization for TensorFlow 和 Intel Optimization for PyTorch。两者需要在 Linux 系统下安装，故而我们在 GNS-v40 的 window 系统上使用了适用于 Linux 的 Windows 子系统 (WSL) ，在 Ubuntu 22.04 LTS 的环境下进行开发。

对于 TensorFlow，我们将其替换为 intel-TensorFlow，同时本主机支持 avx512 指令集，可以使用 intel-TensorFlow-avx512 来达到更好的效果，本次使用的 intel-TensorFlow-avx512 版本为 2.9.1。经过对比 intel-TensorFlow 和 intel-TensorFlow-avx512 对比原版本的 TensorFlow 分别提升了 7%和 10%的推理速度。

对于 PyTorch，我们需要额外的安装对应 torch 版本的 intel-extension-for-pytorch 来对模型进行优化，本次使用的 pytorch 和 intel-extension-for-pytorch 分别为 1.11.0 和 1.11.200。经过对比优化后的速度约比原先提升 5%左右。

3.4 视觉检测模块

3.4.1 文字识别

在这部分，我们主要完成文字的位置和文字内容的识别，即文本检测和文本识别。通过对文字位置和文字内容为用户提供商品信息。

文字识别是计算机视觉研究领域的分支之一，目前最常见的就是光学字符识别 OCR（Optical haracter Recognition），是指电子设备（例如扫描仪或数码相机）采用光学的方式将纸质文档中的文字转换成为黑白点阵的图像文件，并经过识别软件将图像中的文字转换成文本格式的技术，可以大致分成识别特定场景下的专用 OCR 以及识别多种场景下的通用 OCR。通用的 OCR 就是使用在更多、更复杂的场景下，拥有比较好的泛性。基于深度学习的 OCR 是一种相对成熟的解决方案。

当前深度学习 OCR 算法均采用的两阶段模式：文本检测+文本识别。OCR 的研究开始较早，目前已经很多的成熟的解决方案，有较多 python 库和开源项目。其中使用较多的有：CHINESEOCR，EasyOCR，PaddleOCR。由于商场环境不确定性和中文文字本身的特点，比如：图片背景极其丰富、亮度不均衡、光照不均衡、残缺遮挡、文字扭曲、字体多样等等问题，会带来极大的挑战。但是 PaddleOCR 不仅支持超轻量级中文 OCR 预测模型，总模型仅 8.6M，其中检测模型 DB（4.1M）+识别模型 CRNN（4.5M）），中英文数字组合识别、竖排文本识别、长文本识别，由于是国内的开源项目，对中文的支持较好，而且提供多种文本检测训练算法（EAST、DB）和多种文本识别训练算法（Rosetta、CRNN、STAR-Net、RARE），对环境的适应性好，同时能输出文本的位置信息，更便于我们进行语音播报。因此，在 PaddleOCR 使用的是飞桨的框架，而非主流的 tenserflow 或 PyTorch 的情况下，我们仍然选择使用 PaddleOCR 作为解决方案。

OCR 分为文字检测和文字识别两个步骤，为满足支持中英文及字符识别的要求，中文检测模型选择 ch_PP-OCRv3_det，中文识别模型选择 ch_PP-OCRv3_rec。最近一次的更新中增加了表格文字图像、图像关键信息抽取任务和不规则文字图像的标注功能，对于商品中各类文本的支持更加优秀。其工作过程如下所示：

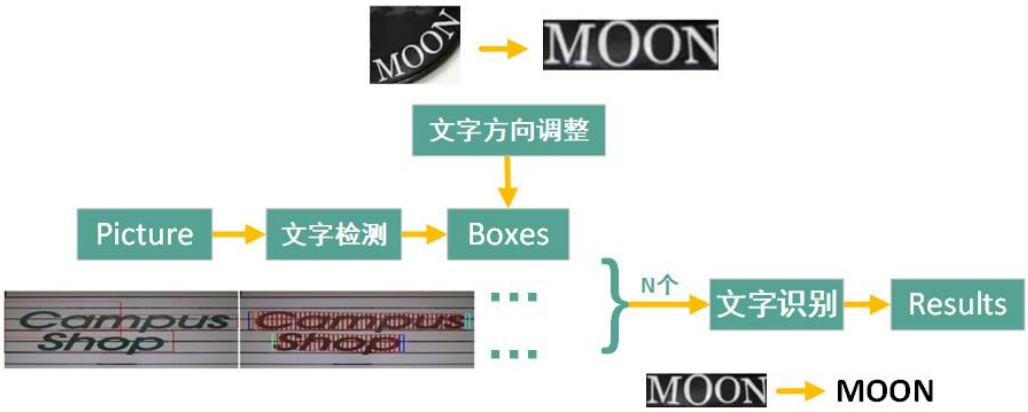

图 17 文字识别

处理具体过程：首先需要将图片中含有文本的区块识别并分割出来，然后调整其方向，将扭转、倾斜、颠倒的文字调整成为横向连续文字的图片。多个区块的文字方向调整完成之后，并行进行文字识别，最终得到输出结果。

识别效果如下图所示

图 18 文字识别效果示意图

3.4.2 条码识别

在这部分，我们将实现通过对一维条形码的识别，获取商品相关信息。

一维条形码，即 EAN 码（European Article Number），是国际物品编码协会制定的一种商品用条码，通用于全世界。EAN 码符号有标准版（EAN-13）和缩短版（EAN-8）两种。EAN-13 标准码（长码)共 13 位数，主要应用于超级市场和其它零售业。其中，国家代码占 3 位，厂商代码占 4 位，产品代码占 5 位，以及检查码占 1 位。EAN-13 码的结构与编码方式如图所示。

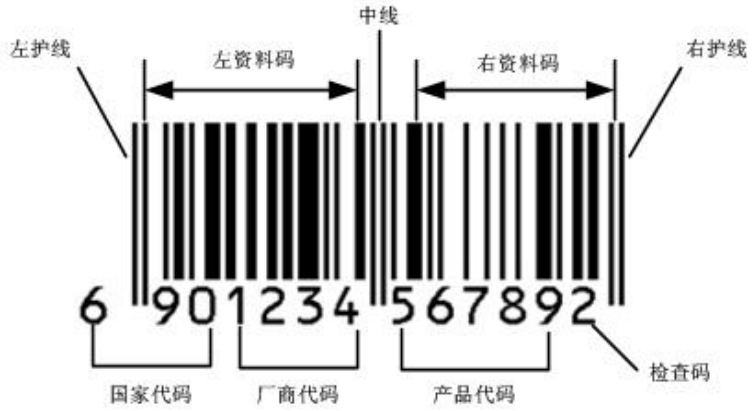

图 19 EAN-13 码的结构与编码方式

常见的条形码识别项目有 zbar，zxing 两大类别。zbar 出现的较早，并有 zblight 这一更加小巧的改进版本，更加小巧。但是二者内核一致，zbar 仅支持 python 2.x 版本，不支持 python3.x 版本。同时 zblight 也不支持 windows，均不满足实际需求。之后，zbar 出现了 pyzbar 这一改进版本，能够支持 python3.x，对条码的识别效果也较好。同时，pyzbar 能够在条码倾斜、扭转时取得较好的扫描效果。

而 zxing 为基于 java 开发的条码识别项目，之后又推出了在 python 上移植的 pyzxing，支持对一维码和二维码的识别。

由于一维码没有纠错机制，并且商品包装褶皱、反光、扭转、倾斜等现象较为严重，zbar 和 zxing 在使用低分辨率摄像头进行识别时，目标条码打印在曲面上或者目标条码密集或扫描角度非垂直的情况下，很难识别，且条码扫描会产生误读使效果不尽如人意，这也是现如今红外扫码枪仍作为工业级扫描设备普遍使用的原因。经测试，在当前设备、环境下，pyzbar 的效果优于 pyzxing，于是我们选择 pyzbar 作为一维码扫描的最终解决方案。

第四章　系统测试

4.1　硬件测试

本系统拥有较多的外设：树莓派、麦克风、耳机、摄像头、定位基站、IMU 等，外设相互之间需要协调运作。为测试硬件之间的运行情况，我们进行了 50 次购物全流程模拟测试，测试结果如下：

表 5 硬件测试结果

测试项目	测试结果
GNS-v40 运行测试	正常
GNS-v40 与树莓派通信测试	正常
树莓派运行测试	正常
麦克风运行测试	正常
耳机运行测试	正常

摄像头运行测试	正常
固定基站运行测试	正常
移动基站运行测试	正常
IMU 运行测试	正常

4.2 功能测试

由于边缘计算主机 GNS-v40 上运行了较多的服务，为测试各个服务的运行情况、精度、速度等指标，我们单独对各功能进行了独立测试。

4.3 测试数据

1、定位服务

目前场地环境下，目前精度范围，步长 0.3m，步频 0.8s 步/s，基本能达到实时跟随效果

2、语音识别 ASRT：

语音识别 ASRT 使用了基于 Intel oneAPI 的加速，对于识别速度和精度有着一定的提升；同时，为了提高关键词的提取准确率，我们使用拼音替代文字作为识别结果。

针对 ASRT 的实际应用场景，使用 400 段长短不一，平均字数为 20 字的语音文件多次对优化前后进行测试，最终测试结果如下所示：

表 6 OneAPI 优化前后速度和准确率

	识别平均速度	识别平均准确率
优化前	0.923/100 字	62%
优化后	0.839/100 字	85%
提升幅度	约 10%	约 23%

3、语音合成 TTS：

语音合成 TTS 使用了基于 Intel oneAPI 的加速，对于识别速度和精度有着一定的提升；我们使用长度约为 50 字的 250 个文本文件对优化前后进行测试，测试结果为：

优化后 1.07s/10 字，1.21s/10 字，提升约 5%~6%

4、文字识别 OCR：

针对 OCR 实际应用场景，使用收集的 300 张图像，每张图平均有 17 个文本框，图像内容包括合同，车牌，铭牌，火车票，化验单，表格，证书，街景文字，名片，数码显示屏等，测试结果为：

每张图片的识别时间约为 0.27s。

第五章 附录

5.1 程序清单

5.1.1 语音识别 ASRT

```
asrserver_http.py                    # 采用 flask 的语音识别客户端
```

```
│   asrt_config.json                    # 模型基本参数配置
│   client_http.py                      # 基于 HTTP 的服务请求
│   data_loader.py                      # 加载语音数据集
│   dict.txt                            # 用户字典
│   evaluate_speech_model.py            # 训练效果评价
│   language_model3.py                  # 基于 N-Gram 的语言模型
│   speech_model.py                     # 声学模型基础功能模板定义
│   train_speech_model.py               # 模型训练
├──model_language                       # 语言模型字典
│       language_model1.txt
│       language_model2.txt
├──save_models                          # 模型权重
│       SpeechModel251bn.model.base.h5
│       SpeechModel251bn.model.h5
├──speech_features
│   │   base.py                         # 基本声学特征提取模块
│   │   sigproc.py                      # 特征计算的信号处理
│   │   speech_features.py              # 声学特征提取模块
├──utils
│   │   config.py                       # 加载 ASRT 配置文件相关
│   │   ops.py                          # 常用操作函数的定义
│   │   thread.py                       # 多线程
```

5.1.2 语音合成 TTS

```
├──models
│   ├──tts_models-zh-CN-baker-tacotron2-DDC-GST    # 基于 tacotron2 的中文合成
│   │       config.json
│   │       model_file.pth
│   │       scale_stats.npy
├──TTS
│   │   model.py                                   # 基础训练模型配置
│   ├──config
│   │   │   shared_configs.py                      # 基础参数配置
│   │   ├──utils                                   # 工具集如训练可视化
│   │   │   │   training.py
│   │   │   │   visual.py
│   ├──server
│   │   ├──server_output                           # 存储历史语音合成输出
│   │   │   client.py                              # 语音合成客户端
│   │   │   conf.json                              # 服务端配置
│   │   │   server.py                              # 基于 flask 的服务端
│   ├──tts
│   │   │ tacotron2_config.py                      # tacotron2 模型参数配置
│   │   ├──layers                                  # tacotron2 各层定义
```

```
│   │   │   losses.py
│   │   │   attentions.py
│   │   │   gst_layers.py
│   │   │   ……
│   │   ├──models                          # tacotron2 上层模型
│   │   │   tacotron2.py
```

5.1.3 文字识别 OCR

```
│ ocr_ server.py                  # 基于 pdserving&flask 的 OCR 服务端程序
│ ocr_client.py                 # HTTP 请求客户端
├─ocr_det_model                   # 文本检测模型
│   inference.pdiparams           # paddle 模型和参数
│   inference.pdmodel
│   serving_server_conf.prototxt  # pdserving 的转换模型
│   serving_server_conf.stream.prototxt
└─ocr_rec_model                   # 文本识别模型
    inference.pdiparams
    inference.pdmodel
    serving_server_conf.prototxt
    serving_server_conf.stream.prototxt
```

5.1.4 条码扫描

```
│   product.txt                   # 商品信息
│   Scan_client.py                # 扫描客户端
│   Scan_server.py                # 基于 flask 的扫描客户端
├──pyzbar                        # pyzbar 源文件
│   │   libiconv.dll
│   │   libzbar-64.dll
│   │   locations.py
│   │   pyzbar.py
│   │   pyzbar_error.py
│   │   wrapper.py
│   │   zbar_library.py
│   ├──scripts
│   │       read_zbar.py
└──TestPictures                  # 测试图片
```

5.1.5 室内导航

```
│   client.py                     # socket 客户端
│   map.py                        # 地图建模和路径 地图可视化
│   myplot.jpg                    # 地图可视化
│   server.py                     # socket 服务端 监听和导航可视化
```

5.1.6 树莓派端

```
│   Camera.py                     # 调用摄像头
```

```
|   PyaudioPlay.py                    # 调用声卡播放
|   PyaudioRecord.py                  # 录音进程
|   RaspClient.py                     # 树莓派主程序
├──IMG_and_AUDIO                      # 播报语音和摄像头截取图片
|   |   .....
```

5.2 程序源码

由于篇幅限制，对于上图中的程序清单，我们此处仅展示部分的关键代码。

5.2.1 语音识别 ASRT

在这一部分我们仅展示服务器部分的代码，如下所示：

```python
import json
import base64
from flask import Flask, Response, request
from language_model3 import ModelLanguage
from speech_features import Spectrogram
from speech_model import ModelSpeech
from speech_model_zoo import SpeechModel251BN
from utils.ops import decode_wav_bytes

API_STATUS_CODE_OK = 200000  # OK
API_STATUS_CODE_CLIENT_ERROR = 400000
API_STATUS_CODE_CLIENT_ERROR_FORMAT = 400001  # 请求数据格式错误
API_STATUS_CODE_CLIENT_ERROR_CONFIG = 400002  # 请求数据配置不支持
API_STATUS_CODE_SERVER_ERROR = 500000
API_STATUS_CODE_SERVER_ERROR_RUNNING = 500001  # 服务器运行中出错


app = Flask("ASRT API Service")
# app.debug = True

AUDIO_LENGTH = 1600
AUDIO_FEATURE_LENGTH = 200
CHANNELS = 1
# 默认输出的拼音的表示大小是1428，即1427个拼音+1个空白块
OUTPUT_SIZE = 1428
sm251bn = SpeechModel251BN(
    input_shape=(AUDIO_LENGTH, AUDIO_FEATURE_LENGTH, CHANNELS),
    output_size=OUTPUT_SIZE
)
feat = Spectrogram()
ms = ModelSpeech(sm251bn, feat, max_label_length=64)
ms.load_model('save_models/' + sm251bn.get_model_name() + '.model.h5')

ml = ModelLanguage('model_language')
ml.load_model()


class AsrtApiResponse:
    '''''
    ASRT语音识别基于HTTP协议的API接口响应类
    '''

    def __init__(self, status_code, status_message='', result=''):
        self.status_code = status_code
        self.status_message = status_message
        self.result = result

    def to_json(self):
        '''''
```

```
50.         类转 json
51.         '''
52.         return json.dumps(self, default=lambda o: o.__dict__,
53.                           sort_keys=True)
54.
55.
56. # api 接口根 url:GET
57. @app.route('/', methods=["GET"])
58. def index_get():
59.     '''''
60.     根路径 handle GET 方法
61.     '''
62.     buffer = ''
63.     with open('assets/default.html', 'r', encoding='utf-8') as file_handl
    e:
64.         buffer = file_handle.read()
65.     return Response(buffer, mimetype='text/html; charset=utf-8')
66.
67.
68. # api 接口根 url:POST
69. @app.route('/', methods=["POST"])
70. def index_post():
71.     '''''
72.     根路径 handle POST 方法
73.     '''
74.     json_data = AsrtApiResponse(API_STATUS_CODE_OK, 'ok')
75.     buffer = json_data.to_json()
76.     return Response(buffer, mimetype='application/json')
77.
78.
79. # 获取分类列表
80. @app.route('/<level>', methods=["POST"])
81. def recognition_post(level):
82.     '''''
83.     其他路径 POST 方法
84.     '''
85.     # 读取 json 文件内容
86.     try:
87.         if level == 'speech':
88.             request_data = request.get_json()
89.             samples = request_data['samples']
90.             wavdata_bytes = base64.urlsafe_b64decode(bytes(samples, encoding
    ='utf-8'))
91.             sample_rate = request_data['sample_rate']
92.             channels = request_data['channels']
93.             byte_width = request_data['byte_width']
94.
95.             wavdata = decode_wav_bytes(samples_data=wavdata_bytes,
96.                                        channels=channels, byte_width=byte_wi
    dth)
97.             result = ms.recognize_speech(wavdata, sample_rate)
98.
99.             json_data = AsrtApiResponse(API_STATUS_CODE_OK, 'speech level
    ')
100.             json_data.result = result
101.             buffer = json_data.to_json()
102.             print('output:', buffer)
103.             return Response(buffer, mimetype='application/json')
104.         elif level == 'language':
105.             request_data = request.get_json()
106.
107.             seq_pinyin = request_data['sequence_pinyin']
108.
109.             result = ml.pinyin_to_text(seq_pinyin)
110.
111.             json_data = AsrtApiResponse(API_STATUS_CODE_OK, 'language level
    ')
112.             json_data.result = result
113.             buffer = json_data.to_json()
```

```
114.             print('output:', buffer)
115.             return Response(buffer, mimetype='application/json')
116.         elif level == 'all':
117.             request_data = request.get_json()
118.
119.             samples = request_data['samples']
120.             wavdata_bytes = base64.urlsafe_b64decode(samples)
121.             sample_rate = request_data['sample_rate']
122.             channels = request_data['channels']
123.             byte_width = request_data['byte_width']
124.             print(sample_rate, channels, byte_width)
125.             wavdata = decode_wav_bytes(samples_data=wavdata_bytes,
126.                                        channels=channels, byte_width=byte_w
    idth)
127.             print(wavdata.shape)
128.             result_speech = ms.recognize_speech(wavdata, sample_rate)
129.             result = ml.pinyin_to_text(result_speech)
130.
131.             json_data = AsrtApiResponse(API_STATUS_CODE_OK, 'all level')
132.             json_data.result = result
133.             buffer = json_data.to_json()
134.             print('ASRT Result:', result, 'output:', buffer)
135.             return Response(buffer, mimetype='application/json')
136.         else:
137.             request_data = request.get_json()
138.             print('input:', request_data)
139.             json_data = AsrtApiResponse(API_STATUS_CODE_CLIENT_ERROR, '')
140.             buffer = json_data.to_json()
141.             print('output:', buffer)
142.             return Response(buffer, mimetype='application/json')
143.     except Exception as except_general:
144.         request_data = request.get_json()
145.         json_data = AsrtApiResponse(API_STATUS_CODE_SERVER_ERROR, str(excep
    t_general))
146.         buffer = json_data.to_json()
147.         print("output:", buffer, "error:", except_general)
148.         return Response(buffer, mimetype='application/json')
149.
150.
151. if __name__ == '__main__':
152.     app.run(host='0.0.0.0', port=8090)
```

5.2.2 语音合成 TTS

在这一部分我们仅展示服务器部分的代码，如下所示：

```
1.  import json
2.  import os
3.  from pathlib import Path
4.  from typing import Union
5.
6.  from TTS.utils.manage import ModelManager
7.  from TTS.utils.synthesizer import Synthesizer
8.  from flask import Flask, request, send_file
9.  from pydub import AudioSegment
10.
11. path = Path(__file__).parent / "../.models.json"
12. manager = ModelManager(path)
13.
14. tts_checkpoint = '/home/intel/.local/share/tts/tts_models--zh-CN--baker--tac
    otron2-DDC-GST/model_file.pth'
15. tts_config_path = '/home/intel/.local/share/tts/tts_models--zh-CN--baker--ta
    cotron2-DDC-GST/config.json'
16. tts_speakers_file = None
17. vocoder_checkpoint = None
```

```
vocoder_config = None
use_cuda = False,

# load models
synthesizer = Synthesizer(
    tts_checkpoint=tts_checkpoint,
    tts_config_path=tts_config_path,
    tts_speakers_file=tts_speakers_file,
    tts_languages_file=None,
    vocoder_checkpoint=vocoder_checkpoint,
    vocoder_config=vocoder_config,
    encoder_checkpoint="",
    encoder_config="",
    use_cuda=use_cuda,
)

use_multi_speaker = hasattr(synthesizer.tts_model, "num_speakers") and (
        synthesizer.tts_model.num_speakers > 1 or synthesizer.tts_speakers_file is not None
)

speaker_manager = getattr(synthesizer.tts_model, "speaker_manager", None)
use_gst = synthesizer.tts_config.get("use_gst", False)
app = Flask(__name__)


def style_wav_uri_to_dict(style_wav: str) -> Union[str, dict]:
    """
    转换 uri 中的音频
    参数:  style_wav (str): uri 中的字符串
    返回:  Union[str, dict]: 文件路径 (str) 或 gst 样式 (dict)
    """
    if style_wav:
        if os.path.isfile(style_wav) and style_wav.endswith(".wav"):
            return style_wav

        style_wav = json.loads(style_wav)
        return style_wav
    return None


empty = AudioSegment.from_wav(f'./empty1.wav')
from ffmpeg import audio

import re


@app.route("/tts", methods=["GET"])
def tts():
    # a = time.time()
    text = str(request.args.get("text"))
    text = re.sub('[a-zA-Z/!&@#$%^()]', '', text)  # sub 是查找替换，找到英文和数字[a-zA-Z0-9],替换成空字符，替换后首位会有空格
    text = text.replace(' ', '')
    text = text.replace('。,。', '。')
    print(text)
    # b = time.time()
    # print(b - a)
    if os.path.exists(f'/home/intel/TTS/TTS/server/server_output/{text}.wav'):
        print(" > 输入{}已经存在".format(text))
        return send_file(f'/home/intel/TTS/TTS/server/server_output/{text}.wav', mimetype="audio/wav")
    else:
        speaker_idx = request.args.get("speaker_id", "")
        style_wav = request.args.get("style_wav", "")
        style_wav = style_wav_uri_to_dict(style_wav)
        print("输入: {}".format(text))
```

```
        wavs = synthesizer.tts(text, speaker_name=speaker_idx, style_wav=style_wav)
        # c = time.time()
        # print(c - b)
        # out = io.BytesIO()
        # synthesizer.save_wav(wavs, out)
        synthesizer.save_wav(wavs, f'/home/intel/TTS/TTS/server/server_output/temp.wav')
        sound = AudioSegment.from_wav(f'/home/intel/TTS/TTS/server/server_output/temp.wav')
        output = empty + sound
        output.export(f'/home/intel/TTS/TTS/server/server_output/temp.wav', format="wav")
        # d = time.time()
        # print(d - c)
        #     audio.a_speed(f'./server_output/temp.wav', "1.2", f'./server_output/output.wav')
        # return send_file(f'./server_output/output.wav', mimetype="audio/wav")
        if len(text) > 100:
            text = 'output'
        # e = time.time()
        # print(e - d)
        audio.a_speed(f'/home/intel/TTS/TTS/server/server_output/temp.wav', "1.2", f'./server_output/{text}.wav')
        return send_file(f'/home/intel/TTS/TTS/server/server_output/{text}.wav', mimetype="audio/wav")


def main():
    app.run(debug=False, host="0.0.0.0", port=5002)


if __name__ == "__main__":
    main()
```

5.2.3 文字识别 OCR

在这一部分我们仅展示服务器部分的代码，如下所示：

```
from paddle_serving_app.reader import OCRReader
import cv2
import sys
import numpy as np
from paddle_serving_app.reader import Sequential,URL2Image, ResizeByFactor
from paddle_serving_app.reader import Div, Normalize, Transpose
from paddle_serving_app.reader import DBPostProcess, FilterBoxes, GetRotateCropImage, SortedBoxes
from paddle_serving_server.web_service import WebService
from paddle_serving_app.local_predict import LocalPredictor
import base64


class OCRService(WebService):
    def init_det_debugger(self, det_model_config):
        self.det_preprocess = Sequential([
            ResizeByFactor(32, 960), Div(255),
            Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]), Transpose((2, 0, 1))])
        self.det_client = LocalPredictor()
        self.det_client.load_model_config(det_model_config)
        self.ocr_reader = OCRReader()

    def preprocess(self, feed=[], fetch=[]):
        data = base64.b64decode(feed[0]["x"].encode('utf8'))
```

```
        data = np.fromstring(data, np.uint8)
        im = cv2.imdecode(data, cv2.IMREAD_COLOR)
        ori_h, ori_w, _ = im.shape
        det_img = self.det_preprocess(im)
        _, new_h, new_w = det_img.shape
        det_img = det_img[np.newaxis, :]
        det_img = det_img.copy()
        det_out = self.det_client.predict(
            feed={"x": det_img}, fetch=["save_infer_model/scale_0.tmp_1"], b
atch=True)
        filter_func = FilterBoxes(10, 10)
        post_func = DBPostProcess({
            "thresh": 0.3,
            "box_thresh": 0.5,
            "max_candidates": 1000,
            "unclip_ratio": 1.5,
            "min_size": 3
        })
        sorted_boxes = SortedBoxes()
        ratio_list = [float(new_h) / ori_h, float(new_w) / ori_w]
        dt_boxes_list = post_func(det_out["save_infer_model/scale_0.tmp_1
"], [ratio_list])
        dt_boxes = filter_func(dt_boxes_list[0], [ori_h, ori_w])
        dt_boxes = sorted_boxes(dt_boxes)
        get_rotate_crop_image = GetRotateCropImage()
        img_list = []
        max_wh_ratio = 0
        for i, dtbox in enumerate(dt_boxes):
            boximg = get_rotate_crop_image(im, dt_boxes[i])
            img_list.append(boximg)
            h, w = boximg.shape[0:2]
            wh_ratio = w * 1.0 / h
            max_wh_ratio = max(max_wh_ratio, wh_ratio)
        if len(img_list) == 0:
            return [], []
        _, w, h = self.ocr_reader.resize_norm_img(img_list[0],
                                                  max_wh_ratio).shape
        imgs = np.zeros((len(img_list), 3, w, h)).astype('float32')
        for id, img in enumerate(img_list):
            norm_img = self.ocr_reader.resize_norm_img(img, max_wh_ratio)
            imgs[id] = norm_img
        feed = {"x": imgs.copy()}
        fetch = ["save_infer_model/scale_0.tmp_1"]
        return feed, fetch, True

    def postprocess(self, feed={}, fetch=[], fetch_map=None):
        rec_res = self.ocr_reader.postprocess_ocrv2(fetch_map, with_score=Tr
ue)
        res_lst = []
        for res in rec_res:
            res_lst.append(res[0])
        res = {"res": res_lst}
        return res


ocr_service = OCRService(name="ocr")
ocr_service.load_model_config("ocr_rec_model")
ocr_service.prepare_server(workdir="workdir", port=9292)
ocr_service.init_det_debugger(det_model_config="ocr_det_model")
ocr_service.run_debugger_service()
ocr_service.run_web_service()
```

5.2.4 条码扫描

```
from pyzbar.pyzbar import decode, ZBarSymbol
```

```
import cv2
import numpy as np
import json
import base64
from flask import Flask, request, jsonify

app = Flask(__name__)
app.config['JSON_AS_ASCII'] = False
app.debug = False
# 初始化一个 panda 吧~
import pandas as pd

df = pd.read_table('product.txt', sep='\t', index_col=0)


@app.route('/scan/', methods=['post'])
def post_http():
    def base2numpy(img_str: str):
        img_decode = img_str.encode('utf8')
        img_decode = base64.b64decode(img_decode)  # 解 base64 编码，得图片的二进制
        img_np = np.frombuffer(img_decode, np.uint8)
        img = cv2.imdecode(img_np, cv2.COLOR_RGB2BGR)  # 转为 opencv 格式
        return img

    if not request.data:  # 检测是否有数据
        return ('fail')
    params = request.data.decode('utf-8')
    params = json.loads(params)
    img_str = params['data']
    img = base2numpy(img_str)

    if params['method'] == 'QRCODE':
        res = decode(img, symbols=[ZBarSymbol.QRCODE])
    elif params['method'] == 'EAN13':
        res = decode(img, symbols=[ZBarSymbol.EAN13])
    else:
        res = decode(img)

    return_data = {'data': []}
    temp = {'data': None, 'type': None, 'quality': None, 'orientation': None}
    for item in res:
        print(item)
        id = item.data.decode('utf-8')
        if params['method'] == 'EAN13':
            id_series = df.loc[int(id), :]
            temp['data'] = {'ID': id, 'PRICE': id_series.PRICE, 'NAME': id_series.NAME,
                            'TIME': id_series.TIME}
        else:
            temp['data'] = item.data.decode('utf-8')
        temp['type'] = item.type
        temp['quality'] = item.quality
        temp['orientation'] = item.orientation
        return_data['data'].append(temp)
    return jsonify(return_data)


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

5.2.5 室内导航

该部分的代码分为地图和客户端：

首先是地图的建模和路径的计算：

```
import networkx as nx
from networkx.generators.lattice import grid_2d_graph
from networkx.algorithms.shortest_paths.unweighted import all_pairs_shortest_path
import networkx.drawing.nx_pylab as pltnx
import matplotlib.pylab as plt
from networkx.algorithms.shortest_paths.weighted import multi_source_dijkstra_path
from networkx.classes.function import neighbors
from networkx.algorithms.shortest_paths.generic import all_shortest_paths
from tqdm import tqdm

# %%
temp = None


class Mymap:
    def __init__(self, X, Y, barriers: list, target: dict, Grid_size=0.5):
        print('Map init')
        self.X = X
        self.Y = Y
        self.barrier = barriers
        self.Grid_size = Grid_size
        # 生成二维网格图
        self.map = grid_2d_graph(int(self.X / self.Grid_size) + 1, int(self.Y / self.Grid_size) + 1,
                                 create_using=nx.DiGraph)
        self._last_point = (0, 0)
        x_size = 2 / int(X / Grid_size + 1)
        y_size = 2 / int(Y / Grid_size + 1)
        # pos 用于绘制的时候的 layout
        self.pos = {}
        for node in self.map.nodes:
            self.pos.update({node: (-1 + node[0] * x_size, -1 + node[1] * y_size)})
        # 边界节点之间
        edge_nodes = []
        for i in range(0, int(X / Grid_size + 1)):
            edge_nodes.append((i, 0))
            edge_nodes.append((i, int(X / Grid_size)))
        for i in range(0, int(Y / Grid_size + 1)):
            edge_nodes.append((0, i))
            edge_nodes.append((int(Y / Grid_size), 0))
        # 生成障碍物 移除相关的边
        for barrier in barriers:
            left, up = int(barrier[0][0] / Grid_size), int(barrier[0][1] / Grid_size)
            right, down = int(barrier[1][0] / Grid_size), int(barrier[1][1] / Grid_size)
            # print(left, up, right, down)
            for a in range(left, right + 1):
                edge_nodes.append((a, up))
                edge_nodes.append((a, down))
            for b in range(down, up + 1):
                edge_nodes.append((left, b))
                edge_nodes.append((right, b))
            for i in range(left + 1, right):
                for j in range(down + 1, up):
                    # print(i, j)
                    for a, b in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
                        ii, jj = i + a, j + b
                        if self.inborder(ii, jj):
                            # print(f'{i, j}-->{ii, jj}')
                            # print(f'{ii, jj}-->{i, j}')
                            # map[(ii, jj)][(i, j)]['weight'] = float('inf')
```

```
                            # map[(i, j)][(ii, jj)]['weight'] = float('inf
')
                            try:
                                self.map.remove_edge((ii, jj), (i, j))
                                self.map.remove_edge((i, j), (ii, jj))
                            except:
                                pass
        # 所有路径赋值
        for edge in self.map.edges:
            self.map[edge[0]][edge[1]]['weight'] = 1
        # for e, datadict in self.map.edges.items():
        #     print(e, datadict)

        # 生成目标点
        self.target = {}
        for key in target:
            area = target[key]
            # 这里的 area 是一个如同 barrier 的样子
            left, up = int(area[0][0] / Grid_size), int(area[0][1] / Grid_si
ze)
            right, down = int(area[1][0] / Grid_size), int(area[1][1] / Grid
_size)
            temp = []
            for i in range(left, right + 1):
                for j in range(down, up + 1):
                    temp.append((i, j))
            self.target.update({key: temp})
        # 将目标节点之间的路径设置为无穷
        for key in self.target:
            nodes = self.target[key]
            for node1 in nodes:
                for node2 in nodes:
                    try:
                        self.map.remove_edge(node1, node2)
                        # self.map[node1][node2]['weight'] = float('inf')
                    except:
                        pass
        # 将边界点和边界点之间的权重放大
        for node1 in edge_nodes:
            for node2 in edge_nodes:
                try:
                    # self.map[node1][node2]['weight'] = 9999
                    self.map.remove_edge(node1, node2)
                 except:
                     pass
         for e, datadict in self.map.edges.items():
             print(e, datadict)
         print('Map init done')

    def compute_path(self, load_from_disk=False):
        print('Computing path')
        if load_from_disk:
            fp = open(f'pathv31{self.X}{self.Y}{self.Grid_size}.txt', 'r
')
            self.path = eval(fp.read())
            fp.close()
            return
        self.path = {}
        target = []
        for key in self.target:
            target.extend(self.target[key])
        for s in tqdm(self.map.nodes):
            if s[0] == 0 or s[1] == 0 or s[0] == 16 or s[1] == 20 or s =
= (1, 19) or s == (15, 14):
                continue
            # for s in self.map.nodes:
            spath = {}
            for t in target:
                # for t in target:
```

```
                try:
                    temp = all_shortest_paths(self.map, s, t, weight='weigh
t')
                    best_score = 4
                    best_path = None
                    count = 0
                    for p in temp:
                        score = 0
                        count += 1
                        if count >= 30000:
                            break
                        # print(count)
                        abandon = False
                        for i in range(len(p) - 2):
                            fir, sec = p[i], p[i + 2]
                            if fir[0] == sec[0] or fir[1] == sec[1]:
                                score += 1
                                if score >= best_score:
                                    abandon = True
                                    break
                        if abandon:
                            pass
                        else:
                            best_score = score
                            best_path = p
                    print(f'update:{s},{t},{best_score},{best_path}')
                    spath.update({t: best_path})
                except nx.NetworkXNoPath:
                    pass
            self.path.update({s: spath})
        fp = open(f'pathv31{self.X}{self.Y}{self.Grid_size}.txt', 'w')
        fp.write(str(self.path))
        fp.close()
        print('compute done!')

    def inborder(self, i, j):
        if 0 <= i < int(self.X / self.Grid_size) and 0 <= j < int(self.
Y / self.Grid_size):
            return True
        else:
            return False

    def getcell(self, i, j):
        sx, sy = int(i / self.Grid_size), int(j / self.Grid_size)
        return sx, sy

    def getpath_weight(self, source, target):
        sx, sy = source[0], source[1]
        target = self.target[str(target)]
        if not isinstance(source[0], int):
            sx, sy = self.getcell(sx, sy)
        # print("getpath_weight", [(sx, sy)], target)
        best = float('inf')
        best_path = None
        for t in target:
            try:
                path = self.path[(sx, sy)][(t[0], t[1])]
                if len(path) < best:
                    best = len(path)
                    best_path = path
            except:
                pass
        return best_path, None

    def draw_map(self, load_from_disk=False):
        if load_from_disk:
            return
        self.fig, self.ax = plt.subplots(figsize=(18, 18))
```

```
        pltnx.draw_networkx(self.map, pos=self.pos, node_color='#FFEFD5', node_size=700, ax=self.ax)
        self.fig.tight_layout()
        color = {'1': '#FFA07A', '2': '#98FB98', '3': '#6495ED', '4': '#FFD700'}
        for key in self.target:
            area = self.target[key]
            for node in area:
                self.draw_point(node, key, False, color[key])
        self.ax.legend()
        self.fig.savefig('myplot.jpg')
        import cv2 as cv
        IMG_SIZE = 1080
        img = cv.imread('myplot.jpg')
        img_resize = cv.resize(img, (IMG_SIZE, IMG_SIZE))
        cv.imwrite('test-resize.jpg', img_resize)

        # ax.plot(self.pos[(0, 0)][0], self.pos[(0, 0)][1], 'o', markersize=30, color='#FFA500')
        # return self.ax, self.fig

    def draw_point(self, cnode, label, remove_last=True, color='orange'):
        # color:orange red
        if color == 'orange':
            color = '#FFA500'
        elif color == 'red':
            color = '#FF4500'
        else:
            pass
        cx, cy = cnode[0], cnode[1]
        if remove_last:
            self.ax.plot(self.pos[self._last_point][0], self.pos[self._last_point][1], 'o', markersize=30,
                         color='#FFEFD5')
            self._last_point = (cx, cy)
        self.ax.plot(self.pos[(cx, cy)][0], self.pos[(cx, cy)][1], 'o', markersize=30, color=color, label=label)


if __name__ == '__main__':
    GRID_SIZE = 0.2
    X = 5
    Y = 6
    # [(4.0, 4.0), (4.0, 4.0)],
    BARRIERS = [
        [(1.93, 2.7), (3.8, 2.1)],
        [(0.0, 6.0), (0.6, 0.0)],
        [(0.0, 0.6), (5.0, 0)]
    ]
    TARGET = {
        '1': [(1.3, 6.0), (2.6, 5.4)],  # 32 42 零食
        '4': [(3.8, 4.4), (4.8, 4.0)],  # 38 48 牛奶
    }
    mymap = Mymap(X, Y, BARRIERS, TARGET, GRID_SIZE)
    # mymap.draw_map(load_from_disk=False)
    mymap.compute_path(load_from_disk=False)
    print('all done!')
```

随后是服务端监听和导航的代码：

```
from socket import *
import os, time
from threading import Thread
import cv2 as cv
import re
import queue
from mapv32 import Mymap
```

```
8.
9.  theta = 'R'
10. action = 'R'
11. QUEUE = queue.Queue(10)
12. # 可能会被其他程序引用的全局变量
13. GRID_SIZE = 0.3
14. X = 5
15. Y = 6
16. # [(4.0, 4.0), (4.0, 4.0)],
17. BARRIERS = [
18.     [(1.93, 2.7), (3.8, 2.1)],
19.     [(0.0, 6.0), (0.6, 0.0)],
20.     [(0.0, 0.6), (5.0, 0)]
21. ]
22. TARGET = {
23.     '1': [(1.3, 6.0), (2.6, 5.4)],  # 32 42 零食
24.     '4': [(3.8, 4.4), (4.8, 4.0)],  # 38 48 牛奶
25. }
26. PORT = 8848
27. load_from_disk = True
28. # 本程序用到的全局变量
29. cpos = [0.6, 0.6, 0]
30. cdes = None
31. display_done = True
32. cdesnode = None
33.
34.
35. class Myserver:
36.     def __init__(self, ip='0.0.0.0', port=8848, bufsize=4096, maxcon=5):
37.         self.ip = ip
38.         self.port = port
39.         self.bufsize = bufsize
40.         self.maxcon = maxcon
41.         print("Server init")
42.         ADDR = (ip, port)
43.         self.serverSocket = socket(AF_INET, SOCK_STREAM)  # 创建欢迎套接字
44.         self.serverSocket.bind(ADDR)  # 将该服务器端口号与套接字关联
45.         # self.serverSocket.listen(self.maxcon)  # 等待并聆听客户敲门,设置 5 为
    请求连接的最大数
46.         self.serverSocket.listen()  # 等待并聆听客户敲门,设置 5 为请求连接的最大
    数
47.         print('The server is ready to receive')
48.
49.     def listen(self):
50.         # 等待处理客户端连接请求
51.         global cpos, cdes, QUEUE, display_done
52.         while True:
53.             print('return outer while!')
54.             cdes = None
55.             # 用户敲门时，调用 accept()创建 connectionSocket 的新套接字
56.             self.connfd, addr = self.serverSocket.accept()
57.             # 与客户端连接成功
58.             print('connect from ', addr)
59.             while True:
60.                 messages = self.connfd.recv(self.bufsize).decode()  # 接收消
    息字节数
61.                 # print('messages', messages, type(messages))
62.                 # 如果 data 为空意味着客户端断开
63.                 if not messages:
64.                     print('not message')
65.                     break
66.                 for item in re.findall("{(.*?)}", messages):
67.                     mes = eval('{' + item + '}')
68.                     # print("Receive:", mes, time.strftime('%M-%S'))
69.                     ctype = mes['TYPE']
70.                     if ctype == 'A':
71.                         temp = mes['DES']
72.                         if int(temp) == 1 or int(temp) == 4:
73.                             cdes = temp
```

```
                    if ctype == 'P':
                        display_done = bool(int(mes['STATE']))
                        # print(f"{mes['STATE']},display{display_done}")
                    temp = [float(i) for i in mes['POS']]
                    if 0 < temp[0] < 5 and 0 < temp[1] < 6:
                        if QUEUE.full():
                            QUEUE.get()
                        QUEUE.put(temp)
                        temp = [0, 0, 0]
                        for i in range(QUEUE.qsize()):
                            item = QUEUE.get()
                            temp = [i + j for i, j in zip(temp, item)]
                            QUEUE.put(item)
                        cpos = [i / QUEUE.qsize() for i in temp]
                    # print(f'cpos:{cpos},cdes:{cdes}')
            self.connfd.close()

    def send(self, taction: str):
        try:
            if self.connfd:
                typeB = {
                    'TYPE': 'B',  # B 发送给客户端
                    'ACT': taction,  # 操作 Left Right Up Down End False
                }
                self.connfd.send(str(typeB).encode())
        except BaseException as e:
            pass

    def closeconection(self):
        self.serverSocket.close()


def leadway(server: Myserver, map: Mymap):
    def whichaction(source, target) -> str:
        global cdesnode
        sx, sy = source[0], source[1]
        sx, sy = map.getcell(sx, sy)
        path, _ = map.getpath_weight((sx, sy), target)
        if path is None:
            print(f'\n{sx, sy}向{target}不可达')
            return 'F'
        if len(path) <= 2:
            print(f'\n{sx, sy}向{target}到达目的地')
            return 'E'
        print(f'best path{path[0]},{path[1]}...{path[-1]}', end='\t')
        cdesnode = path[-1]
        nx, ny = path[1][0], path[1][1]
        if sx == nx and sy < ny:
            tempaction = 'U'
        elif sx == nx and sy > ny:
            tempaction = 'D'
        elif sx < nx and sy == ny:
            tempaction = 'R'
        else:
            tempaction = 'L'

        return tempaction

    def whichtheta(cpos: list):
        # 首先将 cpos 中的最后一个角度转换为方向
        # 右转为正 左转为负 [-180,180]之间
        temptheta = float(cpos[-1])
        if 0 <= temptheta < 45 or -45 <= temptheta < 0:
            theta = 'U'
        elif -135 <= temptheta < -45:
            theta = 'R'
        elif -180 <= temptheta <= -135 or 135 <= temptheta < 180:
            theta = 'D'
        else:
```

```
143.             theta = 'L'
144.         return theta
145.
146.     def consider_theta(theta: str, action: str):
147.         # 首先将 cpos 中的最后一个角度转换为方向
148.         #   右转为正 左转为负 [-180,180]之间
149.         if theta == 'U':
150.             return action
151.         elif theta == 'R':
152.             # 原先：映射后的
153.             map = {'R': 'U', 'L': 'D', 'U': 'L', 'D': 'R'}
154.             return map[action]
155.         elif theta == 'D':
156.             map = {'R': 'L', 'L': 'R', 'U': 'D', 'D': 'U'}
157.             return map[action]
158.         elif theta == 'L':
159.             map = {'R': 'D', 'L': 'U', 'U': 'R', 'D': 'L'}
160.             return map[action]
161.
162.     global cpos, cdes, theta, action, display_done, cdesnode
163.     last_action = None
164.     while True:
165.         if cdes is None or display_done is False:
166.             # 如果目的地空 或者没有播放完毕 仅计算角度提供绘图支持 然后推出
167.             theta = whichtheta(cpos)
168.             continue
169.         else:
170.             # 计算当前朝向
171.             theta = whichtheta(cpos)
172.             # 理想情况下计算下一步动作
173.             tempaction = whichaction(cpos, cdes)
174.             # 验证是否达到终点
175.             sx, sy = cpos[0], cpos[1]
176.             sx, sy = map.getcell(sx, sy)
177.             target = mymap.target[str(cdes)]
178.             for tar in target:
179.                 if sx == tar[0] and sy == tar[1]:
180.                     tempaction = 'E'
181.             # 到达则置空
182.             if tempaction == 'E':
183.                 cdes = None
184.                 last_action = None
185.                 cdesnode = None
186.                 server.send(tempaction)
187.             elif tempaction == 'F':
188.                 pass
189.             else:
190.                 # 否则就计算在当前朝向下的动作 更新 action
191.                 action = consider_theta(theta, tempaction)
192.                 print(f'当前{theta}下一步{action}')
193.                 if action == last_action:
194.                     pass
195.                 else:
196.                     print(f'send-->{action}\t' * 10)
197.                     server.send(action)
198.                 last_action = action
199.             time.sleep(1)
200.
201.
202. def showpath(mymap: Mymap):
203.     # mymap.draw_map()
204.     global cpos, cdes, theta, action, display_done, cdesnode
205.     RADIUS = 5
206.     img = cv.imread('resize.jpg', 1)
207.     CELL = (55, 100)
208.     IMG_SIZE = 1080
209.     STEPX, STEPY = (IMG_SIZE - CELL[1] - CELL[0]) / mymap.X, (IMG_SIZE - CE
    LL[0] - CELL[1]) / mymap.Y
210.     while True:
```

```
211.         i, j = cpos[0], cpos[1]
212.         # 注意在坐标转换的时候 原图中的00是从左下角标注的 但是在cv2中像素的00是从左上角开始的 也就是0x
213.         i, j = int(CELL[0] + i * STEPX), int(CELL[1] + (mymap.Y - j) * STEPY)
214.         img_copy = img.copy()
215.         cv.circle(img_copy, center=(i, j), radius=RADIUS, color=(0, 165, 255), thickness=2 * RADIUS)
216.         if cdes is not None:
217.             heredes = mymap.target[str(cdes)]
218.         else:
219.             heredes = None
220.         cv.putText(img_copy, org=(50, 50), fontFace=cv.FONT_HERSHEY_TRIPLEX, fontScale=0.5,
221.                    text='Current pos:[{:.3f},{:.3f},{:.3f}],Current des:{},display:{}'.format(
222.                        cpos[0], cpos[1], cpos[2], cdesnode, display_done), color=(0, 0, 0), thickness=1)
223.         cv.putText(img_copy, org=(50, 70), fontFace=cv.FONT_HERSHEY_TRIPLEX, fontScale=0.5,
224.                    text='Green:current Blue:next step', color=(0, 0, 0), thickness=1)
225.         if action == 'U':
226.             ii, jj = i, j - int(STEPY / 6)
227.         elif action == 'D':
228.             ii, jj = i, j + int(STEPY / 6)
229.         elif action == 'R':
230.             ii, jj = i + int(STEPX / 6), j
231.         else:
232.             ii, jj = i - int(STEPX / 6), j
233.         cv.arrowedLine(img_copy, (i, j), (ii, jj), (128, 0, 0), RADIUS, 8, 0, 0.2)
234.         if theta == 'U':
235.             ii, jj = i, j - int(STEPY / 6)
236.         elif theta == 'D':
237.             ii, jj = i, j + int(STEPY / 6)
238.         elif theta == 'R':
239.             ii, jj = i + int(STEPX / 6), j
240.         else:
241.             ii, jj = i - int(STEPX / 6), j
242.         cv.arrowedLine(img_copy, (i, j), (ii, jj), (0, 128, 0), RADIUS, 8, 0, 0.2)
243.
244.         cv.imshow('img', img_copy)
245.         cv.waitKey(30)
246.
247.
248. if __name__ == '__main__':
249.     mymap = Mymap(X, Y, BARRIERS, TARGET, Grid_size=GRID_SIZE)
250.     mymap.draw_map(load_from_disk=load_from_disk)
251.     mymap.compute_path(load_from_disk=load_from_disk)
252.     myserver = Myserver()
253.     listen_thread = Thread(target=myserver.listen, args=())
254.     listen_thread.start()
255.     lead_thread = Thread(target=leadway, args=(myserver, mymap))
256.     lead_thread.start()
257.     draw_thread = Thread(target=showpath, args=(mymap,))
258.     draw_thread.start()
```

5.2.6 树莓派端程序

树莓派端的程序分为调用摄像头，调用声卡播放，录制，以及主程序。

首先是摄像头调用：

```
1. import cv2
```

```
def StreamToPC():
    try:
        global s
        global img
        global img_numpy
        cap = cv2.VideoCapture(0)
        PicNum=5
        print("摄像头是否已经打开 ？ {}".format(cap.isOpened()))
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 2594)  # 设置图像宽度
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1944)  # 设置图像高度
        while True:
            for i in range(PicNum):
                _, img = cap.read()
                #print("cap",img.shape)
                _, send_data = cv2.imencode('.jpg', img, [cv2.IMWRITE_JPEG_QUALITY, 70])

                #s.sendto(send_data, addr)
                cv2.imwrite('test'+str(i)+'.jpg', img)
                #img_numpy = cv2.imread('test.jpg')
                #cv2.waitKey(1)  # 延时等待，防止出现窗口无响应流数据发送给视频的接收端
    except KeyboardInterrupt:
        print("摄像头已关闭")
        cap.release()

StreamToPC()
```

而后是录音程序：

```
import socket
import wave

import numpy as np
import pyaudio

# 连接线程管理服务器
ProcessServer = socket.socket()  # 创建 socket 对象    导航服务器
ProcessServerhost = socket.gethostname()
ProcessServerport = 12345  # 设置端口号
print("正在连接到线程管理服务器...")
ProcessServer.connect((ProcessServerhost, ProcessServerport))
print("线程管理服务器已连接...")


def GetTalkingFlag():
    try:
        # 查看是否有音频需要进行识别
        # print('读取录音信息')
        Talkingfile = open('TalkingFlag.txt')
        TalkingFlag = Talkingfile.read()
        # print(recordFlag)
        Talkingfile.close()
        return TalkingFlag
    except KeyboardInterrupt:
        ProcessServer.close()
        print("发生错误,关闭 Socket")


def listen():
    global ProcessServer
    try:
        frames = []
        minValue = 800  # 音量阈值
        stopValue = 30  # 静止时间 截断
        filenum = 1
        temp = 20  # temp 为检测声音值
```

```
        CHUNK = 1024
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 16000
        RECORD_SECONDS = 2
        WAVE_OUTPUT_FILENAME = "PyTest.wav"
        p = pyaudio.PyAudio()

        stream = p.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        frames_per_buffer=CHUNK)
        # snowboydecoder.play_audio_file()

        stat = True  # 流输入状态
        stat2 = False  # 表示可以录音 30-1000 之间
        stat3 = False  # 需要截断录音
        tempnum = 0  # tempnum、tempnum2、tempnum3 为时间
        tempnum2 = 0
        tempnum3 = 0
        while stat:
            data = stream.read(CHUNK, exception_on_overflow=False)
            audio_data = np.frombuffer(data, dtype=np.short)
            temp = np.max(audio_data)
            # print("音量------",temp)
            if temp > minValue and stat2 == False:  # 音量超过阈值
                # TalkingGlag=GetTalkingFlag()
                TalkingGlag = '0'
                if TalkingGlag == '0':
                    stat2 = True  # 设置标志位 ， 清空数据列表
                    frames = []
                    print("录音开始")
                else:
                    print("请稍等")
            elif stat2 and tempnum3 > stopValue:  # 录音过程音量小于 30 超过 sto
    pValue 次
                stat3 = True
            elif stat2 and temp < minValue:
                tempnum3 = tempnum3 + 1
            if stat2:  # 可录音状态
                tempnum2 = tempnum2 + 1
                print("录音·····", tempnum2, "音量------", temp)
                frames.append(data)
            if stat3:
                print("录音结束")
                TempPacket = {"F": "L", "T": "M", "D": '1'}
                ProcessServer.send(str(TempPacket).encode())
                # oiceFlag=open('VoiceFlag.txt','w')
                # VoiceFlag.write('1')
                # time.sleep(0.1)#写入过程等待
                # VoiceFlag.close()
                # 录音结束需要进行标志位标记 作为线程之间的通信
                wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
                wf.setnchannels(CHANNELS)
                wf.setsampwidth(p.get_sample_size(FORMAT))
                wf.setframerate(RATE)
                wf.writeframes(b''.join(frames))
                wf.close()
                filenum = filenum + 1
                stat = True  # 流输入状态
                stat2 = False  # 表示可以录音 30-1000 之间
                stat3 = False  # 需要截断录音
                 tempnum = 0  # tempnum、tempnum2、tempnum3 为时间
                 tempnum2 = 0
                 tempnum3 = 0
             # print(str(temp)  +  "      " +  str(tempnum))
             tempnum = tempnum + 1
             if tempnum > 150000:  # 超时直接退出
```

```
                    stat = False
    except KeyboardInterrupt:
        print("结束音频输入流")
        stream.stop_stream()
        stream.close()
        p.terminate()
        ProcessServer.close()
        print("发生错误,关闭 Socket")


if __name__ == '__main__':
    try:
        listen()
    except KeyboardInterrupt:
        ProcessServer.close()
        print("发生错误,关闭 Socket")
```

而后是调用声卡播放录音队列：

```
import json
import queue
import socket
import threading
import wave

import pyaudio

Zero = '00'  # 表示没有音频需要播放
Hello = '01'  # 初始打招呼
Help = '02'  # 提示服务
WordHelp = '03'  # 提示识别的语句
WordDect = '04'  # 识别到的文字语音
Navigation = '05'  # 识别到需要导航提示
SnackNavigation = '06'  # 识别到需要提示零食区域开始导航
DailyUseNavigation = '07'  # 识别到需要生活用品区域导航
MilkNavigation = '08'  # 识别到需要牛奶用品区域导航
CheckNavigation = '09'
SorryDestination = '10'
TurnRight = '11'  # 提示右转
TurnLeft = '12'  # 提示左转
GoStraight = '13'  # 提示直行
GoBack = '14'
NavigaEnd = '15'
PlayQueue = queue.Queue(20)
# 连接线程管理服务器
ProcessServer = socket.socket()  # 创建 socket 对象    导航服务器
ProcessServerhost = socket.gethostname()
ProcessServerport = 12345  # 设置端口号
print("正在连接到线程管理服务器...")
ProcessServer.connect((ProcessServerhost, ProcessServerport))
print("线程管理服务器已连接...")

MSPacketLen = 31


def RequestPlaySocket():
    try:
        global ProcessServer, PlayQueue
        while 1:
            if PlayQueue.qsize() <= 1:  # 队列少可以进行请求
                Temppacket = {"F": 'S', "T": 'M', "D": '00'}
                # print("Speaker--请求添加列表")
                ProcessServer.send(str(Temppacket).encode())  # Master 向 Spea
ker 发送
    except KeyboardInterrupt:
        ProcessServer.close()
```

```
47.         print("发生错误,关闭 Socket")
48.     except ConnectionResetError:
49.         ProcessServer.close()
50.         print("发生 ConnectionResetError 错误,关闭 Socket")
51.
52.
53. def GetPlaySocket():
54.     try:
55.         global ProcessServer, PlayQueue
56.         print("开启线程管理树莓派主线程客户端")
57.         while 1:
58.             rec = ProcessServer.recv(MSPacketLen).decode("utf-8")
59.             rec = rec.replace("\'", "\"")
60.             front_index = rec.find('{')
61.             rear_index = rec.find('}')
62.             # print("rec=",rec)
63.             if rec != '' and front_index != -1 and rear_index != -1 and fron
    t_index < rear_index:  # 如果获取到非空报文
64.                 rec = rec[front_index:rear_index + 1]  # 保证取出了{}之间的内
    容
65.                 Speakerpacket = json.loads(rec)
66.                 # print("Speaker--Recv",rec)
67.                 # 首先确保收到的 Listener 的包
68.                 From = Speakerpacket['F']  # 这里面只有两种类型 P/G     POST 类型
    和 GET 类型
69.                 To = Speakerpacket['T']
70.                 DATA = Speakerpacket['D']  # 该数据决定了后面的数据处理情况
71.                 if From == "M" and To == "S":  # Master 向播放线程发送的
72.                     # print("当前的录音情况--",DATA)
73.                     if DATA == '00':  # 没有音频播放
74.                         continue
75.                         # print("没有音频播放")
76.                     else:
77.                         PlayQueue.put(DATA)  # 加入到播放队列中
78.     except KeyboardInterrupt:
79.         ProcessServer.close()
80.         print("发生错误,关闭 Socket")
81.     except ConnectionResetError:
82.         ProcessServer.close()
83.         print("发生 ConnectionResetError 错误,关闭 Socket")
84.
85.
86. def GetPlayFlag():
87.     try:
88.         global ProcessServer, PlayQueue
89.         # print("PlayQueue.qsize()=",PlayQueue.qsize())
90.         if PlayQueue.empty():  # 如果播放队列小于三个，向主程序请求播放音频
91.             return None
92.         else:  # 如果有音频需要播放，直接在队列中返回一个即可
93.             return PlayQueue.get()
94.     except KeyboardInterrupt:
95.         ProcessServer.close()
96.         print("发生错误,关闭 Socket")
97.     except ConnectionResetError:
98.         ProcessServer.close()
99.         print("发生 ConnectionResetError 错误,关闭 Socket")
100.
101.
102. # if PlayQueue.qsize()<3:#如果播放队列小于三个，向主程序请求播放音频
103. #             Temppacket = {"F":'S',"T":'M',"D":'00'}
104. #             print("Speaker--请求添加列表")
105. #             ProcessServer.send(str(Temppacket).encode())#Master 向 Speaker
    发送
106. #             if PlayQueue.empty():
107. #                 return None
108. #             else: return PlayQueue.get()
109. # 下面是导航音频播放结果的收发过程
110. # 设置全局变量，NavPlay 确定每次导航的音频是否处理成功
111.
```

```
112.
113. def Speak():
114.     Speaking = False
115.     filenum = 1
116.     temp = 20  # temp为检测声音值
117.     LastPlay = None
118.     CHUNK = 1024
119.     FORMAT = pyaudio.paInt16
120.     CHANNELS = 2
121.     RATE = 22050
122.     RECORD_SECONDS = 2
123.     WAVE_OUTPUT_FILENAME = 'test'
124.     state = 1
125.     LastPlay = None
126.     IfNav = False
127.     PlayNum = 10  # 允许先播放一些
128.     global PlayQueue
129.     try:
130.         # instantiate PyAudio
131.         p = pyaudio.PyAudio()
132.         # open stream
133.         stream = p.open(format=FORMAT,
134.                         channels=CHANNELS,
135.                         rate=RATE,
136.                         output=True)
137.         while state:
138.             data = b''
139.             # 查看各个标志位
140.             Flag = GetPlayFlag()
141.             PlayOne = 0
142.             # print("Speak Flag==",Flag)
143.             if LastPlay == Help and Flag == Help:
144.                 Flag = Zero
145.             if Flag == Hello:
146.                 print("播放" + "hello.wav")
147.                 f = wave.open("hello.wav", "rb")
148.                 # read data
149.                 data = f.readframes(CHUNK)
150.                 # paly stream
151.                 while data != b'':
152.                     stream.write(data)
153.                     data = f.readframes(CHUNK)
154.                 # 清空标志位
155.                 f.close()
156.             elif Flag == Help:
157.                 if IfNav != True:  # 如果在导航过程中，忽略杂音
158.                     print("播放" + "help.wav")
159.                     f = wave.open("help.wav", "rb")
160.                     # read data
161.                     data = f.readframes(CHUNK)
162.                     # paly stream
163.                     while data != b'':
164.                         stream.write(data)
165.                         data = f.readframes(CHUNK)
166.                         PlayOne += 1
167.                         print("播放了: ", PlayOne)
168.                         if PlayOne >= PlayNum:  # 如果播放了一部分，然后发现队
    列非空
169.                             if PlayQueue.empty() != True:  # 如果非空就退出
    循环
170.                                 if PlayQueue.qsize() == 2:
171.                                     print("队列数量达到2，退出")
172.                                     # f.close()
173.                                     break
174.                                 else:
175.                                     print("继续播放")
176.                     # 清空标志位
177.                     f.close()
178.             elif Flag == WordHelp:
```

```
                print("播放" + "detect.wav")
                f = wave.open("detect.wav", "rb")
                # read data
                data = f.readframes(CHUNK)
                # paly stream
                while data != b'':
                    stream.write(data)
                    data = f.readframes(CHUNK)
                # 清空标志位
                f.close()
            elif Flag == WordDect:
                print("播放" + "torch-output.wav")
                f = wave.open("torch-output.wav", "rb")
                # read data
                data = f.readframes(CHUNK)
                # paly stream
                while data != b'':
                    stream.write(data)
                    data = f.readframes(CHUNK)
                # 清空标志位
                f.close()
            elif Flag == Navigation:
                print("播放" + "NavigationRemind.wav")
                f = wave.open("NavigationRemind.wav", "rb")
                # read data
                data = f.readframes(CHUNK)
                # paly stream
                while data != b'':
                    stream.write(data)
                    data = f.readframes(CHUNK)
                # 清空标志位
                f.close()

            elif Flag == SnackNavigation:
                IfNav = True
                print("播放" + "PathSnack.wav")
                f = wave.open("PathSnack.wav", "rb")
                # read data
                data = f.readframes(CHUNK)
                # paly stream
                while data != b'':
                    stream.write(data)
                    data = f.readframes(CHUNK)

                # 清空标志位
                f.close()

            elif Flag == DailyUseNavigation:
                IfNav = True
                print("播放" + "PathDailyUse.wav")
                f = wave.open("PathDailyUse.wav", "rb")
                # read data
                data = f.readframes(CHUNK)
                # paly stream
                while data != b'':
                    stream.write(data)
                    data = f.readframes(CHUNK)
                # 清空标志位
                f.close()
            elif Flag == MilkNavigation:
                IfNav = True
                print("播放" + "PathSMilk.wav")
                f = wave.open("PathMilk.wav", "rb")
                # read data
                data = f.readframes(CHUNK)
                # paly stream
                while data != b'':
                    stream.write(data)
                    data = f.readframes(CHUNK)
```

```
248.                 # 清空标志位
249.                 f.close()
250.             elif Flag == CheckNavigation:
251.                 IfNav = True
252.                 print("播放" + "PathCheck.wav")
253.                 f = wave.open("PathCheck.wav", "rb")
254.                 # read data
255.                 data = f.readframes(CHUNK)
256.                 # paly stream
257.                 while data != b'':
258.                     stream.write(data)
259.                     data = f.readframes(CHUNK)
260.                 # 清空标志位
261.                 f.close()
262.             elif Flag == SorryDestination:
263.                 print("播放" + "SorryDestination.wav")
264.                 f = wave.open("SorryDestination.wav", "rb")
265.                 # read data
266.                 data = f.readframes(CHUNK)
267.                 # paly stream
268.                 while data != b'':
269.                     stream.write(data)
270.                     data = f.readframes(CHUNK)
271.                     PlayOne += 1
272.                     print("播放了：", PlayOne)
273.                     if PlayOne >= PlayNum:  # 如果播放了一部分，然后发现队列非
    空
274.                         if PlayQueue.empty() != True:  # 如果非空就退出循环
275.                             if PlayQueue.qsize() == 2:
276.                                 print("队列数量达到 2，退出")
277.                                 # f.close()
278.                                 break
279.                             else:
280.                                 print("继续播放")
281.                 # 清空标志位
282.                 f.close()
283.             elif Flag == TurnLeft:
284.                 print("播放" + "left.wav")
285.                 f = wave.open("left.wav", "rb")
286.                 # read data
287.                 data = f.readframes(CHUNK)
288.                 # paly stream
289.                 while data != b'':
290.                     stream.write(data)
291.                     data = f.readframes(CHUNK)
292.                 # 清空标志位
293.                 f.close()
294.
295.             elif Flag == TurnRight:
296.                 print("播放" + "right.wav")
297.                 f = wave.open("right.wav", "rb")
298.                 # read data
299.                 data = f.readframes(CHUNK)
300.                 # paly stream
301.                 while data != b'':
302.                     stream.write(data)
303.                     data = f.readframes(CHUNK)
304.                 # 清空标志位
305.                 f.close()
306.             elif Flag == GoStraight:
307.                 print("播放" + "straight.wav")
308.                 f = wave.open("straight.wav", "rb")
309.                 # read data
310.                 data = f.readframes(CHUNK)
311.                 # paly stream
312.                 while data != b'':
313.                     stream.write(data)
314.                     data = f.readframes(CHUNK)
315.                 # 清空标志位
```

```
316.                 f.close()
317.             elif Flag == GoBack:
318.                 print("播放" + "Back.wav")
319.                 f = wave.open("Back.wav", "rb")
320.                 # read data
321.                 data = f.readframes(CHUNK)
322.                 # paly stream
323.                 while data != b'':
324.                     stream.write(data)
325.                     data = f.readframes(CHUNK)
326.                 # 清空标志位
327.                 f.close()
328.             elif Flag == NavigaEnd:
329.                 IfNav = False
330.                 print("播放" + "NavigaEnd.wav")
331.                 f = wave.open("NavigaEnd.wav", "rb")
332.                 # read data
333.                 data = f.readframes(CHUNK)
334.                 # paly stream
335.                 while data != b'':
336.                     stream.write(data)
337.                     data = f.readframes(CHUNK)
338.                 # 清空标志位
339.                 f.close()
340.             if Flag != None or Flag != Zero:
341.                 LastPlay = Flag
342.
343.     except KeyboardInterrupt:
344.         # stop stream
345.         print("音频输出流结束")
346.         stream.stop_stream()
347.         stream.close()
348.         # close PyAudio
349.         p.terminate()
350.         ProcessServer.close()
351.         print("发生错误,关闭 Socket")
352.     except ConnectionResetError:
353.         print("音频输出流结束")
354.         stream.stop_stream()
355.         stream.close()
356.         # close PyAudio
357.         p.terminate()
358.         ProcessServer.close()
359.         print("发生 ConnectionResetError 错误,关闭 Socket")
360.
361.
362. if __name__ == '__main__':
363.     try:
364.         SpeakerSocketHandle = threading.Thread(target=GetPlaySocket)
365.         SpeakerSocketHandle.start()
366.         RequestPlaySocket
367.         SpeakerRequestSocketHandle = threading.Thread(target=RequestPlaySoc
    ket)
368.         SpeakerRequestSocketHandle.start()
369.         Speak()
370.     except KeyboardInterrupt:
371.         ProcessServer.close()
372.         print("发生错误,关闭 Socket")
373.     except ConnectionResetError:
374.         ProcessServer.close()
375.         print("发生 ConnectionResetError 错误,关闭 Socket")
```

最终是树莓派主程序：

```
1.  import json
2.  import socket
3.  import threading
```

```
import time

import ASR
import CODE_SCAN
import OCR
import TorchTTS
import requests
import serial

# import PyaudioPlay
# import PyaudioRecord


WAVE_OUTPUT_FILENAME = "PyTest.wav"

Zero = '0'  # 表示没有音频需要播放
Hello = '1'  # 初始打招呼
Help = '2'  # 提示服务
WordHelp = '3'  # 提示识别的语句
WordDect = '4'  # 识别到的文字语音

# 连接导航服务器
NavServer = socket.socket()  # 创建 socket 对象    导航服务器
Navigationhost = '192.168.30.109'
port = 8848  # 设置端口号
print("正在连接到导航服务器...")
NavServer.connect((Navigationhost, port))
print("导航服务器已连接...")

# 连接线程管理服务器
ProcessServer = socket.socket()  # 创建 socket 对象    导航服务器
ProcessServerhost = socket.gethostname()
ProcessServerport = 12345  # 设置端口号
print("正在连接到线程管理服务器...")
ProcessServer.connect((ProcessServerhost, ProcessServerport))
print("线程管理服务器已连接...")

ret = None
frame = None
footage_socket = None
img_numpy = None

s = None
img = None

start = 1
listen = 2
finish = 3
playmusic = 4
getweather = 5
detectword = 6

name1 = "木子"
name2 = '睦子'
name3 = '目自'
name4 = 'muzi'
music = "yinyue"
detection = "shibie"
weather = "天气"

ServicerIP = '192.168.30.109'
cameraPort = 8081
TTSPort = '8090'
HELLO = 'here.mp3'
HELP = 'help.mp3'
PLAYFILE = 'torch-output.wav'

VoiceFlag = '0'  # 初始化全局变量，控制音频处理的过程

```

```
TurnRight = '11'  # 提示右转
TurnLeft = '12'  # 提示左转
GoStraight = '13'  # 提示直行
GoBack = '14'
NavigaEnd = '15'
NavPlayFlag = '0'  # 全局默认不允许进行导航

# LP 的串口测试
COM = 0
while 1:
    try:
        LPser = serial.Serial("/dev/ttyUSB" + str(COM), 921600)
        if LPser.is_open:
            print("/dev/ttyUSB" + str(COM) + "已打开")
            print("开始处理串口数据...")

        break
    except Exception as e:
        COM = COM + 1
        print("ERROR:", e)
        print("尝试下一个 USB 端口：  " + "/dev/ttyUSB" + str(COM))
        if COM > 10:
            print("串口启动失败，请连接定位模块")
            break
IMUser = serial.Serial("/dev/ttyS0", 115200)
if not IMUser.isOpen():
    print("IMU open failed")
else:
    print("open IMU success: ")
    print(IMUser)


def NavSocketRecv():
    global NavServer, ProcessServer, IMUser, LPser
    try:
        print("开启导航服务接收线程")
        while 1:
            Action = "None"
            rec = NavServer.recv(1024).decode('utf-8')
            rec = rec.replace("\'", "\"")
            print("收到导航包---", rec)
            if rec != '':  # 如果获取到非空报文
                Navpacket = json.loads(rec)
                print("Navpacket=", Navpacket)
                if Navpacket['TYPE'] == 'B':  # 导航控制类报文
                    Action = Navpacket['ACT']
                if Action == 'L':  # 左转标志位
                    Temppacket = {"F": 'P', "T": 'M', "D": TurnLeft}
                    ProcessServer.send(str(Temppacket).encode())  # 主线程向
Master 发送
                    print("设置 Left 标志位")
                if Action == 'R':
                    Temppacket = {"F": 'P', "T": 'M', "D": TurnRight}
                    ProcessServer.send(str(Temppacket).encode())  # 主线程向
Master 发送
                    print("设置 Right 标志位")
                if Action == 'U':
                    Temppacket = {"F": 'P', "T": 'M', "D": GoStraight}
                    ProcessServer.send(str(Temppacket).encode())  # 主线程向
Master 发送
                    print("设置 GoStraight 标志位")
                if Action == 'D':
                    Temppacket = {"F": 'P', "T": 'M', "D": GoBack}
                    ProcessServer.send(str(Temppacket).encode())  # 主线程向
Master 发送
                    print("设置 GoBack 标志位")
                if Action == 'E':
                    Temppacket = {"F": 'P', "T": 'M', "D": NavigaEnd}
```

```
                    ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
                    print("设置NavigaEndFlag标志位")
                # 首先确保收到的是树莓派的包
    except KeyboardInterrupt:
        IMUser.close()
        LPser.close()
        NavServer.close()
        ProcessServer.close()
        print("错误，关闭串口和socket")

    # f=open("postion.txt","w+")


# os.close(sys.stderr.fileno())
def GetWeaterInfo(city):
    BaseWeatherUrl = "https://restapi.amap.com/v3/weather/weatherInfo?city=%s&key=%s&output=%s"
    MyKey = "d855e256c47fd33d55500ee34acaab5f"

    citycode = 510117  # 这里自定义了郫都区
    MyWeatherUrl = BaseWeatherUrl % (citycode, MyKey, "JSON")
    r = requests.get(MyWeatherUrl)
    Result = r.json()
    WeatherInfo = Result["lives"][0]["province"] + Result["lives"][0]["city"] + ", 温度: " + Result["lives"][0][
        "temperature"] + "度"
    return WeatherInfo


# Master 与主线程之间的报文长度最多31
Raspacket = {"F": 'P', "T": 'M', "D": '1', 'Nav': '0'}  # P--Process M--Master  0--请求状态    1/2/3/4/5/……-上报信息
RaspacketLen = len(str(Raspacket))  # 测试获得
MPPacketLen = RaspacketLen  # Master-Process


def VoiceFlagSocket():
    global ProcessServer, NavServer, VoiceFlag, NavPlayFlag, IMUser, LPser
    try:
        print("开启线程管理树莓派主线程客户端")
        while 1:
            # print("等待主线程录音情况数据...")
            rec = ProcessServer.recv(MPPacketLen).decode("utf-8")
            rec = rec.replace("\'", "\"")
            front_index = rec.find('{')
            rear_index = rec.find('}')
            # print("主线程接收到rec=",rec)
            if rec != '' and front_index != -1 and rear_index != -1 and front_index < rear_index:  # 如果获取到非空报文
                # print("Ras-recv:",rec)
                rec = rec[front_index:rear_index + 1]  # 保证取出了{}之间的内容
                Lispacket = json.loads(rec)
                # 首先确保收到的Listener的包
                From = Lispacket['F']  # 这里面只有两种类型P/G    POST类型和GET类型
                To = Lispacket['T']
                DATA = Lispacket['D']  # 该数据决定了后面的数据处理情况
                NavPlayFlag = Lispacket['Nav']
                if From == "M" and To == "P":  # Master向主线程发送的
                    if DATA == '1':
                        print("当前的录音情况--", DATA)
                        VoiceFlag = DATA
    except KeyboardInterrupt:
        IMUser.close()
        LPser.close()
        NavServer.close()
```

```
198.         ProcessServer.close()
199.         print("错误，关闭串口和 socket")
200.
201.
202. Zero = '0'  # 表示没有音频需要播放
203. Hello = '01'  # 初始打招呼
204. Help = '02'  # 提示服务
205. WordHelp = '03'  # 提示识别的语句
206. WordDect = '04'  # 识别到的文字语音
207. Navigation = '05'  # 识别到需要导航提示
208. SnackNavigation = '06'  # 识别到需要提示零食区域开始导航
209. DailyUseNavigation = '07'  # 识别到需要生活用品区域导航
210. MilkNavigation = '08'  # 识别到需要牛奶用品区域导航
211. CheckNavigation = '09'
212. SorryDestination = '10'
213.
214.
215. def PlayFlag(Mode):
216.     # 当需要播放音频的时候对相应的文档进行设置标志位
217.     global ProcessServer
218.     if Mode == 'Hello':  # 需要进行 hello 的音频
219.         Temppacket = {"F": 'P', "T": 'M', "D": Hello}
220.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
221.         print("设置 Hello 标志位")
222.     if Mode == 'DeteHelp':  # Dect 前的提示语
223.         Temppacket = {"F": 'P', "T": 'M', "D": WordHelp}
224.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
225.         print("设置 DeteHelp 标志位")
226.     if Mode == 'DeteWord':  # 播放文字音频
227.         Temppacket = {"F": 'P', "T": 'M', "D": WordDect}
228.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
229.         print("设置 DeteWord 标志位")
230.     if Mode == 'Help':  # 帮助提示
231.         Temppacket = {"F": 'P', "T": 'M', "D": Help}
232.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
233.         print("设置 Help 标志位")
234.     if Mode == 'Navigation':  # 帮助提示
235.         Temppacket = {"F": 'P', "T": 'M', "D": Navigation}
236.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
237.         print("设置 Navigation 标志位")
238.     if Mode == 'Snack':  # 帮助提示
239.         Temppacket = {"F": 'P', "T": 'M', "D": SnackNavigation}
240.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
241.         print("设置 Snack 标志位")
242.     if Mode == 'DailyUse':  # 帮助提示
243.         Temppacket = {"F": 'P', "T": 'M', "D": DailyUseNavigation}
244.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
245.         print("设置 DailyUse 标志位")
246.     if Mode == 'Milk':  # 帮助提示
247.         Temppacket = {"F": 'P', "T": 'M', "D": MilkNavigation}
248.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
249.         print("设置 Milk 标志位")
250.     if Mode == 'Check':  # 帮助提示
251.         Temppacket = {"F": 'P', "T": 'M', "D": CheckNavigation}
252.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
253.         print("设置 Check 标志位")
254.     if Mode == 'SorryDestination':  # 帮助提示
255.         Temppacket = {"F": 'P', "T": 'M', "D": SorryDestination}
256.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
257.         print("设置 SorryDestination 标志位")
258.
259.
260. def GetVoiceFlag():
261.     global ProcessServer, VoiceFlag, IMUser, LPser
262.     try:
263.         # 需要进行请求的发送查看是否有音频需要进行识别
264.         Temppacket = {"F": 'P', "T": 'M', "D": '00'}
265.         ProcessServer.send(str(Temppacket).encode())  # 主线程向Master发送
266.         # print("向线程服务器请求录音情况")
```

```
267.         return VoiceFlag
268.     except KeyboardInterrupt:
269.         IMUser.close()
270.         LPser.close()
271.         NavServer.close()
272.         ProcessServer.close()
273.         print("错误，关闭串口和 socket")
274.
275.
276. # myrecord()
277. def process(case):
278.     TotalLength = 20  # 总长度应该是 172
279.     DATALen = 100  # 去除帧头以及控制位
280.     ReadingFlag = 0  # 表示在获取一个包的数据
281.     MarkingFlag = 0  # 表示正在获取 Mark 标记 因为 TAG 包的长度固定
282.     NumOne = 1
283.     NumTwo = 2
284.     NumAcc = 14  # 不需要校验和，但是在分片的时候，含左不含右，所以需要比总数据字节数长度大一 1+12+1
285.     NumAng = 8  # 1+6+1
286.     NumID = 1
287.     NumFirstData = 1
288.     Flag = 0
289.     start = end = 0.  # 计时
290.     AngZ0 = -0.516357421875
291.     AccXFloat = AccYFloat = AccZFloat = 0.
292.     AngXFloat = AngYFloat = 0.
293.     X = Y = angleZ = 0.
294.     global NavServer, VoiceFlag, ProcessServer, NavPlayFlag, IMUser, LPser
295.     Test = 0
296.     Num = 0
297.     startdata = 0
298.     DataRead = 0
299.
300.     while 1:
301.         try:
302.             Num = (Num + 1) % 150
303.             IMUcount = IMUser.inWaiting()
304.             if IMUcount > 0 and Flag == 0:
305.                 startdata = IMUser.read(NumOne)
306.                 IMUcount -= 1
307.             if startdata == b'\x55':
308.                 # print("startdata=",startdata)
309.                 Flag = 1
310.             if IMUcount > NumAcc and Flag == 1:
311.                 FrontDATA = IMUser.read(NumTwo)
312.                 IDdata = FrontDATA[NumID:NumID + 1]
313.                 if IDdata == b'\x01':
314.                     start = time.time()  # 开始位移积分计时
315.                     DATA = IMUser.read(NumAng)
316.                     # print("DATA=",DATA)
317.                     AngleData = DATA[NumFirstData:NumAng]
318.                     AngX = AngleData[0:2]
319.                     AngY = AngleData[2:4]
320.                     AngZ = AngleData[4:6]
321.                     AngX = int.from_bytes(AngX, 'little', signed=True)
322.                     AngY = int.from_bytes(AngY, 'little', signed=True)
323.                     AngZ = int.from_bytes(AngZ, 'little', signed=True)
324.                     # print("AngX=",AngX,"AngY=",AngY,"AngZ=",AngZ)
325.                     AngXFloat = AngX / 32768.0 * 180
326.                     AngYFloat = AngY / 32768.0 * 180
327.                     angleZ = -(AngZ / 32768.0 * 180)
328.
329.                     NavPacket = {"TYPE": 'P', "POS": [X, Y, angleZ], 'STATE': NavPlayFlag}
330.                     NavServer.send(str(NavPacket).encode())
331.                     print("angleZ=", angleZ)
332.                     print("向服务器发送---", str(NavPacket))
```

```
                        TotalLength -= 1
                        if TotalLength == 0:
                            TotalLength = 200
                            # print("X=",X," Y=",Y," Z=",Z)
                            # print("AngX=",angleX,"AngY=",angleY,"AngZ=",angle
    Z)
                            print("angleZ=", angleZ)
                            print("向服务器发送---", str(NavPacket))
                    Flag = 0
                    DataRead += 1

                count = LPser.inWaiting()
                Header = None
                if count > 1 and ReadingFlag == 0:
                    Header = LPser.read(1)  # 逐位读取帧头
                if Header == b'\x55':
                    # print(Header)
                    ReadingFlag = 1  # 表示读取到帧头，开始获取一帧数据
                if ReadingFlag == 1:
                    if MarkingFlag == 0:
                        Mark = LPser.read(1)  # Read the Mark
                        if Mark == b'\x04':  # 这是 NodeFrame2 协议
                            MarkingFlag = 1
                            # print("TAG ",Mark)

                        else:
                            # 标志位复位，开始下一次读包过程,这里是因为\55 下一个字
    节如果不是 01 控制位则改包可以直接丢弃

                            ReadingFlag = 0  # 表示在获取一个包的数据
                            MarkingFlag = 0  # 表示正在获取 Mark 标记 因为 TAG 包的
    长度固定
                    elif MarkingFlag == 1:  # 以及获取 MarkFlag 等待缓存区解包
                        count = LPser.inWaiting()
                        if count > DATALen:  # 判断是否足够一个 Tag 包
                            DATA = LPser.read(DATALen)
                            # print("DATA:",DATA)
                            Length = int.from_bytes(DATA[0:2], 'little', signed
    =True)
                            # print("Length=",Length)
                            ROLE = DATA[2:3]  # 第三位是类型 这里应该是 TAG
                            ID = DATA[3:4]  # 第四位是 ID 位

                            POSITION = DATA[11:20]
                            # 其他的数据就不解析出来了
                            PX = POSITION[0:3]
                            PY = POSITION[3:6]
                            PZ = POSITION[6:9]
                            # print("AngX=",AngX,"AngY=",AngY,"AngZ=",AngZ)
                            X = int.from_bytes(PX, 'little', signed=True)
                            Y = int.from_bytes(PY, 'little', signed=True)
                            Z = int.from_bytes(PZ, 'little', signed=True)
                            # print("Before T--->  X=",X," Y=",Y," Z=",Z)
                            X /= 1000.0
                            Y /= 1000.0
                            Z /= 1000.0
                            # s.send("X{}Y{}Z{}E".format(X,Y,Z).encode())
                            # 标志位复位，开始下一次读包过程
                            ReadingFlag = 0  # 表示在获取一个包的数据
                            MarkingFlag = 0  # 表示正在获取 Mark 标记 因为 TAG 包的
    长度固定
                            DataRead += 1
                            count = LPser.inWaiting()
                            rest = LPser.read(count)
                            IMUcount = IMUser.inWaiting()
                            rest = IMUser.read(IMUcount)
                #  print("数据读取结束")
                if case == start:
                    recordFlag = GetVoiceFlag()
```

```
                print(recordFlag)
                if recordFlag == '1':
                    print("请求语音服务")
                    # 这里添加语音识别服务请求函数
                    # 并且需要根据请求的结果进行标志位的设置
                    myresult = ASR.asr_client(WAVE_OUTPUT_FILENAME, ServicerIP, method='speech', port=TTSPort)
                    print("识别拼音结果=", myresult)
                    if name1 in myresult or name2 in myresult or name3 in myresult or name4 in myresult:
                        PlayFlag('Hello')  # 需要播放 hello 的音频
                        case = listen

            elif case == listen:
                # time.sleep(0.1)
                # print("获取 flag")
                recordFlag = GetVoiceFlag()
                if recordFlag == '1':
                    print("请求语音服务")
                    # 这里添加语音识别服务请求函数
                    # 并且需要根据请求的结果进行标志位的设置
                    myresult = ASR.asr_client(WAVE_OUTPUT_FILENAME, ServicerIP, method='speech', port=TTSPort)
                    """
                                        Test=(Test+1)%5
                    if Test%5==0:
                        myresult="muzi"
                    elif Test%5==1:
                        myresult="qulingshiquyu"
                    elif Test%5==2:
                        myresult="qumainiunai"
                    elif Test%5==3:
                        myresult="qushenghuoquyu"
                    elif Test%5==4:
                        myresult="hasjhjb"

                    """
                    print("识别拼音结果=", myresult)
                    if 'muzi' in myresult or 'zi' in myresult or 'mu' in myresult:
                        PlayFlag('Hello')  # 需要播放 hello 的音频
                    elif weather in myresult:
                        WeaterInfo = GetWeaterInfo("郫都区")
                        case = listen
                    elif detection in myresult:
                        PlayFlag('DeteHelp')
                        wordresult = OCR.ocr_client('test0.jpg', ServicerIP)
                        print(wordresult)
                        TorchTTS.torch_tts_client(wordresult['result']['res'], ip=ServicerIP)
                        time.sleep(0.1)
                        PlayFlag('DeteWord')
                        case = listen
                    elif 'saotiaoma' in myresult or 'sao' in myresult or 'tiaoma' in myresult or 'tiao' in myresult or 'miao' in myresult:
                        PicNum = 5
                        PlayFlag('DeteHelp')
                        GetNewResult = ' '
                        GetFlag = 0
                        for i in range(PicNum):
                            Scanresult = CODE_SCAN.getcodescan('test' + str(i) + '.jpg', ServicerIP)
                            # print(Scanresult)
                            if '没有' not in Scanresult[0]:  # 扫描成功了
                                GetNewResult = Scanresult[0]
                                GetFlag = 1
                                break
                        if GetFlag:
```

```
458.                             Scanresult[0] = GetNewResult
459.                             print("扫描成功：", Scanresult)
460.                         print("合成：", Scanresult)
461.                         TorchTTS.torch_tts_client(Scanresult, ip=ServicerI
    P)
462.                         time.sleep(0.2)
463.                         PlayFlag('DeteWord')
464.                         case = listen
465.                     elif "zaijian" in myresult:
466.                         case = finish
467.                     elif name1 in myresult or name2 in myresult or name3 i
    n myresult or name4 in myresult:
468.                         PlayFlag('Hello')  # 需要播放 hello 的音频
469.                     # elif 'qu'in myresult:#这里的逻辑应该是如果第一次说的目的
    地在可选的范围内就直接告诉它即将导航前往，否则就提示我能为您导航到 A，B，C，D 四个地
    方
470.                     #     index = myresult.find('qu')
471.                     #     des=myresult[index+2:]
472.                     elif 'lingshi' in myresult or "ling" in myresult:  # 零
    食区域
473.                         # PlayFlag('Navigation')#需要播放零食路径已经规划
474.
475.                         NavPacket = {"TYPE": 'A', "POS": [X, Y, angleZ], 'D
    ES': 1}
476.                         NavServer.send(str(NavPacket).encode())
477.                         print("已发送--", str(NavPacket))
478.                         print("前往零食区域")
479.                         PlayFlag('Snack')  # 需要播放零食路径已经规划
480.                         # 发送导航目的地+位置信息
481.                         # 导航语音提示标志
482.                     elif 'shouyintai' in myresult or 'shou' in myresult o
    r 'souyin' in myresult or 'yintai' in myresult or 'tai' in myresult:
483.                         # PlayFlag('Navigation')#需要播放零食路径已经规划
484.
485.                         NavPacket = {"TYPE": 'A', "POS": [X, Y, angleZ], 'D
    ES': 4}
486.                         NavServer.send(str(NavPacket).encode())
487.                         print('前往收银台区域')
488.                         PlayFlag('Check')  # 需要播放生活用品区域路径已经规
    划
489.                         # 发送导航目的地+位置信息
490.                         # 导航语音提示标志
491.                         # else:#没听清目的地，请重新说一遍
492.                         #     print("对不起，没听清您的目的地，请重说一遍。")#S
    orryDestination
493.                         #     PlayFlag('SorryDestination')#需要播放生活用品区
    域路径已经规划
494.                     # elif 'shenghuo'in myresult or 'yongpin' in myresult o
    r "sheng" in myresult or "huo" in myresult:  #生活用品区域
495.                     #     #PlayFlag('Navigation')#需要播放零食路径已经规划
496.
497.                     #     NavPacket={"TYPE":'A',"POS":[X,Y,angleZ],'DES':
    2}
498.                     #     NavServer.send(str(NavPacket).encode())
499.                     #     print("前往生活用品区域")
500.                     #     PlayFlag('DailyUse')#需要播放生活用品区域路径已经规
    划
501.                     #     #发送导航目的地+位置信息
502.                     #     #导航语音提示标志
503.                     # elif 'niunai' in myresult or 'nai' in myresult or 'ni
    u' in myresult:
504.                     #     #PlayFlag('Navigation')#需要播放零食路径已经规划
505.
506.                     #     NavPacket={"TYPE":'A',"POS":[X,Y,angleZ],'DES':
    3}
507.                     #     NavServer.send(str(NavPacket).encode())
508.                     #     print("前往牛奶区域")
509.                     #     PlayFlag('Milk')#需要播放生活用品区域路径已经规划
510.                     #     #发送导航目的地+位置信息
```

```
511.                         #      #导航语音提示标志
512.
513.                         else:
514.                             PlayFlag('Help')  # 这里面播放的应该是对不起我没听清的
    语音
515.                         if recordFlag == '1':
516.                             VoiceFlag = '0'  # 本地置 0 否则可能会出现连续两次需要
    录音处理的情况
517.
518.                         count = LPser.inWaiting()
519.                         rest = LPser.read(count)
520.                         IMUcount = IMUser.inWaiting()
521.                         rest = IMUser.read(IMUcount)
522.                 elif case == finish:
523.                     print("Exit System...")
524.                     break
525.                 # time.sleep(0.2)
526.                 print("开始下一个循环")
527.             except KeyboardInterrupt:
528.                 IMUser.close()
529.                 LPser.close()
530.                 NavServer.close()
531.                 ProcessServer.close()
532.                 print("错误，关闭串口和 socket")
533.
534.
535. def refresh():
536.     WordDectfile = open('DeteWordFlag.txt', 'w')
537.     WordDectfile.write('0')
538.     WordDectfile.close()
539.     WordHelpfile = open('WordHelpFlag.txt', 'w')
540.     WordHelpfile.write('0')
541.     WordHelpfile.close()
542.     Helpfile = open('HelpFlag.txt', 'w')
543.     Helpfile.write('0')
544.     Helpfile.close()
545.     Hellofile = open('HelloFlag.txt', 'w')
546.     Hellofile.write('0')
547.     Hellofile.close()
548.     Voicefile = open('VoiceFlag.txt', 'w')
549.     Voicefile.write('0')  # 读取之后清除标记位
550.     Voicefile.close()
551.     print("已重启")
552.
553.
554. if __name__ == '__main__':
555.     # StreamUpToPC = threading.Thread(target=StreamToPC)
556.     # StreamUpToPC.start()
557.
558.     NavSocketRecvThreading = threading.Thread(target=NavSocketRecv)
559.     NavSocketRecvThreading.start()
560.
561.     MPHandler = threading.Thread(target=VoiceFlagSocket)
562.     MPHandler.start()
563.     process(listen)
```