摘要

随着城市化进程的不断推进，大型超市、停车场、医院、车站等公共室内空间变得越来越大，人们对室内人员感知技术的使用要求也在日益提高；同时，随着信息化水平的日益提高，室内人员感知技术也将日益深入地运用到人们的日常生活中，为人们提供更方便、更多样的智能服务。在目前的室内人员感知技术中，由于 BLE（Bluetooth Low Energy, 低功耗蓝牙）技术拥有低功耗、覆盖范围宽、低成本、高速连接、稳定性好等优点，再加上在生活中应用广泛，以故基于 BLE 的室内人员感知关键技术的研究得到了人们普遍的重视。

传统的基于视频或者红外的人员感知技术有着建设成本高、感知方向固定、隐蔽性差等较大的局限性；传统蓝牙由于通信距离和功耗的限制，通常只能采用较低精度的测距法。而 BLE 信号的新特性使其可以使用感知精度较高且部署简单的指纹法。因此，本文以架设于室内的 BLE 人员感知实验平台为基础，采用综合 RSSI（接受信号强度指示）信号抖动与指纹对比的方法实现一种室内环境下的无终端人员感知系统。本文的主要内容如下：

首先，阐述课题的研究背景和意义以及分析国内外室内人员感知技术的发展状况、常用技术等。同时，对现有的基于 BLE 信号的无终端人员感知方法以及位置指纹法进行分析和比较。其次，分析系统的设计需求，给出该系统的设计架构和工作流程说明并对其中的关键技术原理做详细的阐述，随后论证该方案的可行性及分析系统的成本。最后，给出该系统的详细实现过程，包括 BLE 人体感知实验平台的搭建、指纹数据库的设计过程及 Android 客户端的开发三个部分。随后基于实验结果分析系统的性能指标，分析结果表明该系统的人员位置感知精度达到 80%以上，满足系统设计需求，为室内人员感知系统的设计与实现提供了一个可用的范例。

关键词：室内人员感知，无终端，低功耗蓝牙，位置指纹法，加权 K 近邻算法

2.4 室内人员感知系统的评价指标

作为一个包含硬件设备和软件系统的室内人员位置感知复杂系统，其主要的评价指标有：定位精度、系统功耗、覆盖范围、部署复杂度以及部署成本等。

1）定位精度

定位精度是位置感知系统最重要最关键的指标。不同的精度适用于不同的应用场景。一般认为：5-10 米的精度称为模糊定位；3-5 米的精度称为区域定位；1-3 米的的精度称为高精度定位；0-1 米的精度称为超高精度。

2）系统功耗

系统功耗是位置感知系统的重要指标之一。因为其直接决定了该系统的使用时间和使用便捷性。一般认为：终端设备可使用两年以上称为功耗极低；终端设备可使用一年以上称为功耗低；终端设备可使用半年以上称为功耗中；终端设备可使用三个月称为功耗高；终端设备可使用一个月称为功耗极高。

3）覆盖范围

覆盖范围是位置感知系统的重要指标之一。覆盖范围直接决定了项目的实施成本和工作周期。一般认为：1 千米以上称为覆盖范围极大；500 米左右称为覆盖范围大；200 米左右称为覆盖范围中；100 米左右称为覆盖范围低；50 米左右称为覆盖范围极低。

4）部署复杂度

部署复杂度是位置感知系统的重要指标之一。其直接决定了项目的实施难度和维护难度。由于信息化项目中的网络部署是最关键的施工部分，因此一般以单基站的网络部署长度为部署复杂度的标准。一般认为：网络部署长度小于 200 米称为复杂度高；网络部署长度小于 100 米称为复杂度中；网络部署长度小于 50 米称为复杂度低；网络部署长度等于 0 米称为复杂度极低。

5）部署成本

部署成本也是位置感知系统的重要指标之一。其直接决定了系统的实际可使用性。部署成本一般以单平方米的造价作为指标。一般认为：1000 元以上称为价格极高；500 元左右称为价格高；200 元左右称为价格中；100 元左右称为价格低；50 元左右称为价格极低。

第 3 章 BLE 室内人员感知系统总体设计

本章主要针对基于 BLE 的室内人员感知系统的设计需求、总体架构及工作流程等做详细的阐述与分析。

3.1 系统需求分析

系统的需求分析是指在经过深入的调研分析后，对于所设计的系统在功能、性能和实用性等方面的具体要求，将非具体化的功能描述转换为形式化的需求定义，明确系统要做什么的过程。室内人员位置感知系统的需求主要包括以下方面：

1）良好的位置感知精度

位置感知服务是该系统的核心功能。只有提供了较好的位置感知精度才能称得上是一个具有实用价值的室内人员位置感知系统，所有基于位置的服务都与感知精度密切相关，只有在保证感知精度的基础上才能给用户提供更好的服务。本文设计的系统基于位置指纹法实现室内人员位置感知，其感知精度可达 80%以上。

2）良好的用户体验

用户体验是否良好决定了一个产品能不能获得广泛的使用。随着智能移动设备的快速发展，用户对于实时的、便捷的、直观的、交互性强的使用体验的要求越来越高，因此为保证用户使用的方便快捷，需要基于常见智能移动设备开发应用程序。本文设计的系统基于使用普遍的搭载安卓操作系统的智能手机开发有一款提供实时位置服务的 APP，界面简洁优雅，使用方便快捷。

3）低廉的成本

成本是否低廉决定了一个产品是否能获得市场的青睐。在能够提供相同的服务的前提下，越低的成本就意味着越少的消耗，也就意味着更大的市场。本文的系统采用价格较低的 BLE 设备开发，在保证感知精度的同时也有效地控制了成本。

4）良好的可维护性和可扩展性

系统时常需要维护或是更新换代，比如需要增加新的功能或是改变设备的部署位置等，这些都需要系统具有良好的可扩展性和可维护性。本文设计的系统采用自顶向下、模块化的设计模式，很好地解决了系统的扩展性和维护性的难题。

3.2 系统方案设计概述

本文所设计的基于 BLE 的无终端室内人员位置感知系统采用 C/S（客户端-服务器）架构，基于无源位置指纹法构建整个系统。主要包括人员感知实验平台、Android APP 和 MQTT 服务器三个部分。系统整体架构如图 3.1 所示。

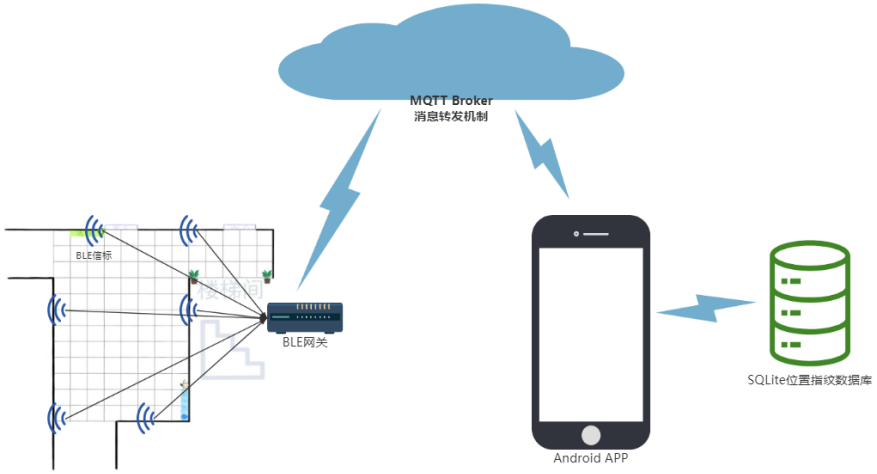


图 3.1 系统架构图

其中人员感知实验平台由 BLE 信标和 BLE 网关搭建而成，在离线收集阶段用于收集位置指纹信息构建数据库，在在线定位阶段用于为位置感知服务提供收集到的实时 RSSI 数据；Android 客户端 APP 用于展示室内地图并在接收到实时 RSSI 数据后刷新感知到的人员位置；MQTT 服务器用于接收 BLE 网关上传的数据并推送到 Android APP 端，是沟通人员位置感知实验平台和用户端的桥梁。系统的工作流程如图 3.2 所示。

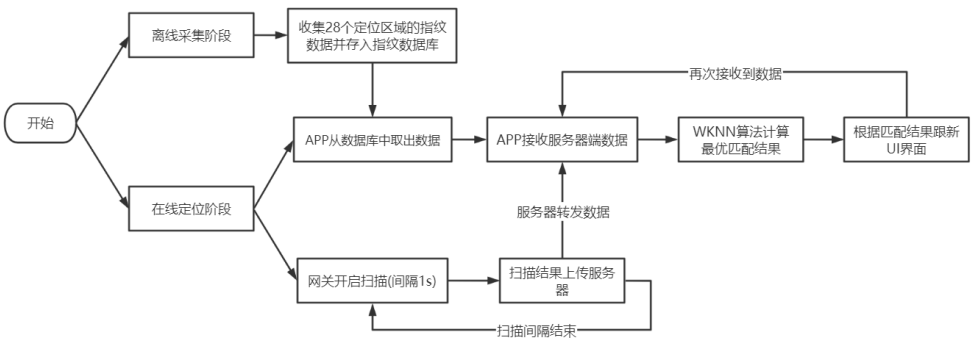


图 3.2 系统工作流程图

整个系统分为两个阶段：离线采集阶段和在线定位阶段。离线采集阶段根据绘制的室内地图和在地图上划分的 28 个定位区域（每个区域大小为 2m * 2m）分别收集基于 RSSI 的位置指纹数据并存入本地数据库；在线采集阶段由 BLE 网关每间隔 1s 扫描收集一次周围的 BLE 信标节点的 RSSI 数值并组装为 Json 格式上传 MQTT 服务器，服务器收集到数据后通过 MQTT 的消息转发机制转发数据到 Android 客户端，Android 端处理接收到的 Json 字符串获取 RSSI 数值并与数据库中的每个定位位置的 RSSI 指纹做对比，由 WKNN 匹配算法求出最终定位结果后，在 APP 的地图展示界面实时刷新当前的定位位置。其中 Android 客户端 APP 的数据处理逻辑如图 3.3 所示。

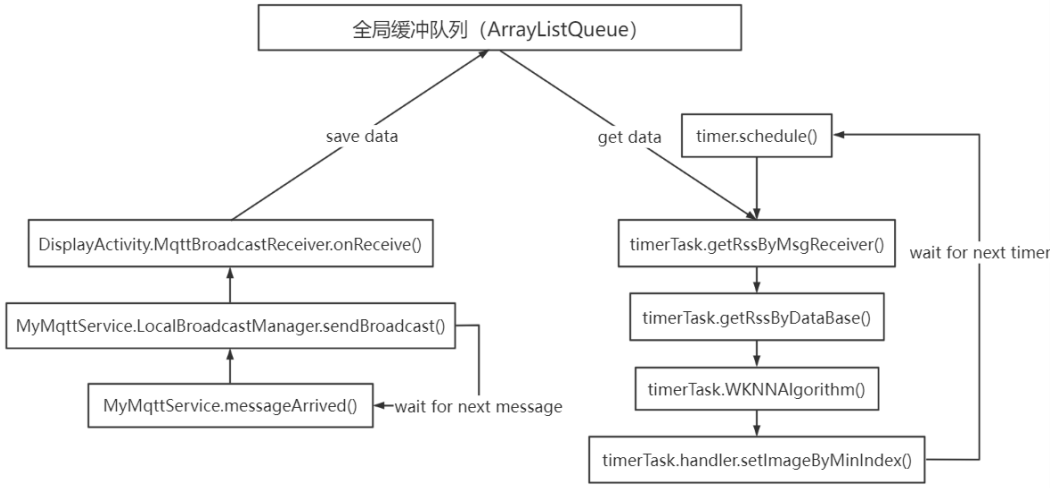


图 3.3 Android APP 工作流程图

由上图可以看出，Android 端在通过 MQTT Service 的接收消息回调函数中接收到服务器推送的 Json 字符串后通过安卓本地广播机制发送到展示界面中，在地图展示界面的广播接收者函数中将收到的 Json 数据写入全局缓冲队列；同时，启动了一个安卓定时器任务，每间隔 1s 从全局缓冲队列中获取一条数据，并从 Json 格式的数据中解析出 RSSI 数值，同时从本地 SQLite 数据库中获取所有定位位置的指纹数据，将两组数据传入 WKNN 定位算法中，最终在求出最优的匹配位置结果后，通过安卓 Handler 机制在定时器子线程中更新 UI 线程中的地图展示效果，将人员位置展示在室内地图上。

3.3 系统关键技术分析

本文所设计的室内人员位置感知系统主要用到的硬件设备有 BLE 信标和 BLE 网关以及安卓开发平台；所用到的技术原理有 WKNN 位置感知算法和 MQTT 数据传输协议等。下面分别予以介绍。

3.3.1 BLE 硬件设备

目前市场上的蓝牙技术主要有以下两种：传统蓝牙技术和低功耗蓝牙技术。而低功耗蓝牙技术也是基于传统的蓝牙技术的基础上发展而来的，与传统蓝牙不同的是，低功耗蓝牙设备能以极低的功耗实现快速连接和数据传输，仅靠纽扣电池就能有一年以上的工作时间。且低功耗蓝牙基于免费的频段，和 WiFi、电磁炉等一样，都是使用的 ISM2.4GHz 频段，这使得它不需要缴纳授权费用。低功耗蓝牙是一种专门针对物联网应用领域而开发的无线通信技术，它在移动设备、汽车电子、医疗保健和室内定位等领域都有广泛的应用前景。

本文设计的室内人员位置感知系统使用的是国内公司四月兄弟所推出的 aBeacon 信标节点和 BLE 网关。aBeacon 信标节点是一种用于定位和营销的低功耗蓝牙产品，该产品使用 1 块 CR2450 型号电池和板载大天线，传输距离可达 60 米，超长的续航能力使得各类场景的布置非常轻松。可应用于线下顾客广告、优惠券推送、精准营销、商场定位、微信摇一摇推送和展会签到等领域。该产品具有如下特点：

1）空旷传输距离可达 60 米；

2）胶贴安装方式，等级 IP65；

3）采用 1 节 CR2450 电池；

4）支持微信摇一摇接入；

5）防尘。

BLE 网关是一种用于扫描收集周边 BLE 信标节点广播数据的蓝牙设备，该设备可实现对于蓝牙信标节点的监测和管理。BLE 网关收集信标节点的广播数据后，可以通过数据传输协议发送到本地 TCP 服务器或网络 HTTP/MQTT 服务器。但是蓝牙网关本身是不具备数据解析能力的，只能获取周边信标节点的 MAC 地址、

UUID、major、minor 和 RSSI 等数据并组装为 Json 格式或 MessagePack 格式上传服务器端。该产品具有如下特点：

1）采用 NRF52832 蓝牙芯片，超高蓝牙接收灵敏度，实现大范围监测；

2）Cortex-M4 微控制器内核，超高处理速度；

3）支持 MQTT、HTTP 和 WebSocket 三种数据上传方式。可扫描多种 BLE 数据类型（定向广播、非定向广播、可连接广播和不可连接广播等）；

4）可实现 RSSI 数据根据 MAC 地址过滤；

5）内置看门狗，确保产品稳定；

6）BLE PCB 天线。

aBeacon 节点和 BLE 网关的产品图及实物图如图 3.4 所示。

(a)aBeacon 节点产品图

(b)BLE 网关产品图

(c)aBeacon 节点和 BLE 网关实物图

图 3.4 aBeacon 节点产品图(a)、BLE 网关产品图(b)及实物图(c)

3.3.2 安卓开发平台

智能手机是移动互联网时代最常见的电子设备之一。其像 PC 一样具有独立的操作系统支持，内置有无线互联网接入、多媒体应用、网页浏览等诸多功能，并可通过第三方服务商安装更多的移动端软件，使得其功能可以得到扩展。相比于传统手机，智能手机的综合处理能力更出色，因此成为信息化时代人手必备的电子产品之一。智能手机在其内部安装了移动操作系统，最常见的是 iOS 系统和 Android 系统。移动操作系统是 PC 端操作系统在移动端的迁移和发展，它继承了很多 PC 操作系统的优点，又有其适应于移动端特点的新特性，移动操作系统的进步促进了智能手机行业的快速发展。当下，搭载 Android 操作系统的手机已经占据了 80%以上的市场份额，Android 系统以其开放性和易用性获得了广大用户的青睐。

由于 Android 操作系统的开源性且其硬件设备比较便宜，方便开发与研究，所以本文将基于 Android 智能手机开发 APP（所用的型号是 Realme Q2），为用户提供室内位置服务。

Android APP 的开发基于 Google 推出的集成开发环境 Android Studio，使用 Java1.8 版本编写程序。Android Studio 集成了开发与调试功能且拥有强大的 UI 编辑器，其基于 Gradle 构建程序，拥有专属的重构和快速修复功能，是一款优秀的集成开发环境。其开发界面如图 3.5 所示。

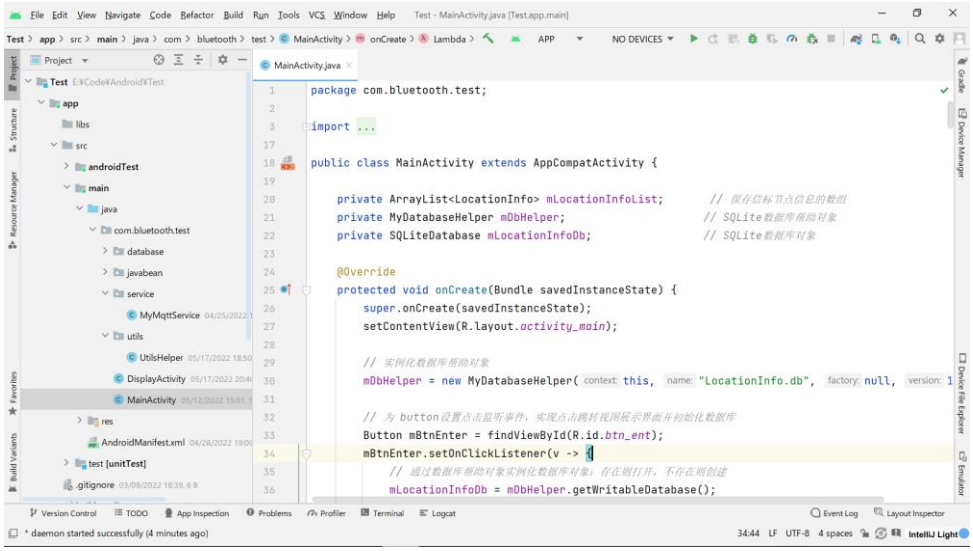


图 3.5 Android Studio 开发环境示意图

3.3.3 人员位置感知算法

基于位置指纹法的人员位置感知算法主要有以下几种：NN（最近邻算法），KNN（K-近邻算法）以及 WKNN（加权 K-近邻算法）等，下面将分析这几种常见算法，并选取其中最合适的作为本文系统所采用的方法。

1）NN 算法

NN 算法直接计算实时采集的 RSSI 数据和数据库中的每一组 RSSI 数据的欧氏距离，并从中选出最小值作为最优匹配点。假设离线采集阶段收集了 m 个位置的 RSSI 数据，这样就获得了 m 组指纹数据，分别表示为 $\{L_1, L_2, \cdots, L_m\}$，$L_i = (\overline{RSSI_{i1}}, \overline{RSSI_{i2}}, \cdots, \overline{RSSI_{ij}}, \cdots, \overline{RSSI_{in}})$，其中 L_i 表示第 i 个位置的指纹数据，$\overline{RSSI_{ij}}$ 表示在第 i 个位置处测得的第 j 个的 Beacon 节点的 RSSI 数据。待测点收集到的 RSSI 数据与数据库指纹的欧氏距离计算公式如下：

$$d_i = \sqrt{(\sum_{j=1}^{n} (RSSI_j - \overline{RSSI_{ij}})^2)} \tag{3.1}$$

根据上式计算所有组的欧氏距离并从中选取值最小的组作为最终的定位点。由于 RSSI 数据受到环境的影响比较大，因此 NN 算法的位置感知算法精度偏低。

2）KNN 算法

KNN 算法是基于 NN 算法的改进而来的。KNN 算法使用多个最近欧氏距离的点来估计最终的定位点坐标，而不是像 NN 算法一样只使用唯一的最小点，这样有利于减小由环境因素带来的误差。KNN 算法的计算流程如下所示：首先仍然基于 NN 算法计算所有组指纹与实时收集的数据的欧氏距离，然后对所有组的指纹根据欧氏距离的大小排序，选出前 K 个位置的指纹，得到其对应的坐标，最后根据多边形的形心计算公式即可得到最终的定位点坐标。KNN 计算公式如下：

$$(x, y) = \frac{1}{k} \sum_{i=1}^{k} (x_i, y_i) \tag{3.2}$$

其中，(x_i, y_i) 代表第 i 个采样点的坐标，(x, y) 代表最终定位点的坐标。

KNN 算法基于 NN 算法做出了一定的改进，使用多个最近点作为最终定位点的估测的方法有效地避免了 RSSI 波动带来的影响，但引入多个最优定位点又带来了如何规划多个定位点的权重的问题，因此接着引入 WKNN 算法。

3）WKNN 算法

WKNN 算法和 KNN 算法一样，选取最小欧氏距离的 K 个点作为最终定位点的估测，但改进之处是对这多个点乘上了权重系数，最常见的权重系数是欧氏距离的倒数。以欧式距离倒数为权重的 WKNN 算法的计算公式如下：

$$(x, y) = \sum_{i=1}^{k} w_i (x_i, y_i) \tag{3.3}$$

其中 (x, y) 为最终定位点的估测坐标，(x_i, y_i) 为第 i 个采样点的位置坐标，w_i 为加权系数，w_i 又可表示为：

$$w_i = \frac{\frac{1}{d_i^2}}{\sum_{j=1}^{k} \frac{1}{d_j^2}} \tag{3.4}$$

其中 d_i 为实时收集的 RSSI 数据与第 i 个位置指纹的欧氏距离。

WKNN 算法进一步弥补了 KNN 算法的不足之处，有效地减小了位置感知误差，提高了位置感知精度。使用欧氏距离倒数为权重的 WKNN 算法实现简单且位置感知精度高，在实际的室内位置感知系统中得到了广泛的应用，因此，本文也将使用 WKNN 算法作为人员位置感知算法开发系统。

3.3.4 数据传输协议

收集到的 RSSI 数据需要在人员位置感知实验平台和 Android APP 之间传输，因此需要一种稳定高速的数据传输方式；又由于 BLE 设备的低功耗特性，因此数据传输方式也要尽可能地减少带宽和开销。

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输协议）是一个基于 TCP/IP 协议栈构建的通信协议，它基于 C/S（客户端-服务器）架构并以发布/订阅（publish/subscribe）模式运作，发布客户端和订阅客户端通过一个共同的 Topic（主题）实现通信，发布端将要传输的数据发送到该主题上，MQTT 服务器将接收到的数据推送到所有订阅了该主题的客户端，实现了一种低耦合的数据传输方式。MQTT 最大的优点是可以以极少的代码的有限的带宽提供完整可靠的、实时的消息服务。作为一种低消耗的通信协议，MQTT 在物联网领域得到了充分的应用。其工作模式如图 3.6 所示。

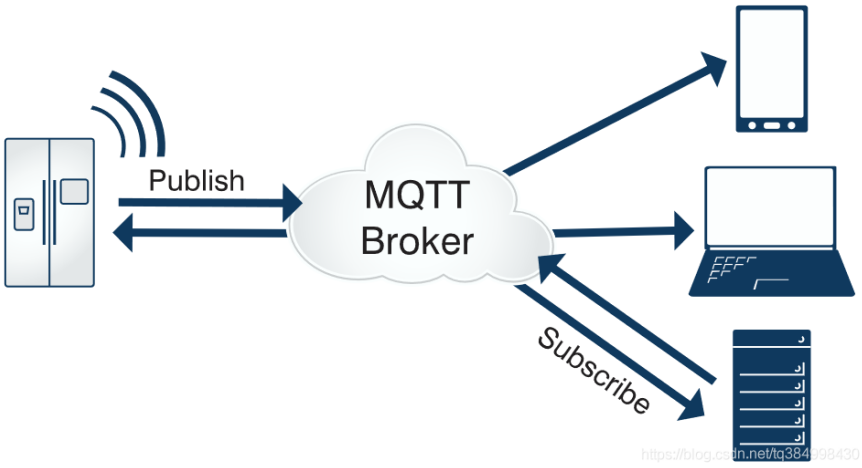


图 3.6 MQTT 工作模式图

由于 MQTT 协议优秀的特性，因此，本文的室内位置感知系统将使用 MQTT 协议作为数据传输协议。

3.4 方案论证及成本分析

本文所设计的系统是一个基于 BLE 硬件设备和安卓开发平台构建的室内人员位置感知应用。该系统基于典型的室内环境绘制地图，划分定位方格，并将 BLE 设备部署在实际环境中，BLE 网关收集每个定位区域的 RSSI 数据中作为位置指纹存储于数据库中；在定位阶段时，网关每间隔 1s 扫描收集周围部署的 BLE 信标节点的广播数据并上传 MQTT 服务器，MQTT 服务器转发数据并推送到 Android 端，Android 端处理接收到数据并和本地位置指纹数据库做对比，在通过 WKNN 匹配算法计算定位结果后在地图上实时更新人员位置。

由上述可得，该系统具有如下优点：

1）良好的人员位置感知精度：基于位置指纹法和 WKNN 定位算法开发，可提供较好的定位精度。

2）良好的用户体验：开发的 Android APP 界面简洁优雅，室内地图清晰明了，可以自动刷新人员位置，提供了良好的用户体验。

3）低廉的成本：系统的主要成本在于 BLE 设备，由于 BLE 设备的可使用时间长，价格较低，所以该系统的成本低廉，有利于大范围的铺开使用。系统成本分析如表 3.1 所示。

4）良好的可扩展性和可维护性：系统基于典型的室内环境开发，经过一定程度的扩展后也可应用于其他室内环境；系统的主要维护项目在于 BLE 硬件设备，由于其较低的成本和较长的使用时间，更换或是维修都比较容易实现。因此系统具有良好的可维护性。

表 3.1 系统成本分析

项目名称	数量	成本说明
BLE 信标节点	6	￥45
BLE 网关	1	￥256
数据库	1	本地数据库，忽略
MQTT 服务器	1	公用服务器，忽略

第 4 章 BLE 室内人员感知系统详细设计

本章主要阐述基于 BLE 的室内人员感知系统的具体实现和测试结果。包括人员位置感知实验平台的搭建、指纹数据库的设计、Android APP 的开发以及人员位置感知的测试结果与分析。

4.1 人员位置感知实验平台的搭建

本系统所搭建的室内人员位置感知实验平台位于学校的第八教学楼 B 栋的一楼茶水间和楼梯交界区域。搭建过程包括 BLE 设备的部署以及蓝牙网关和 MQTT Broker 的配置。

4.1.1 BLE 设备的部署

BLE 设备包括 BLE 信标节点和 BLE 网关。在基于指纹法的位置感知应用中，BLE 信标节点一般部署在离地高度 1.5m~2.0m 之间的墙壁上，水平间距保持 2m~4m 左右；本系统将 BLE 信标节点通过双面胶粘贴在走廊的墙壁上并且保持水平间距在 2m~4m 之间。BLE 网关放置在空旷区域保证信号接受的流畅；本系统将 BLE 网关置于廊道的中间空旷处。

具体的实验场所及 BLE 设备部署示意如图 4.1 所示。

(a) (b)

图 4.1 人员位置感知实验平台实景图(a)和 BLE 设备部署示意图(b)

4.1.2 蓝牙网关的配置

为了实现蓝牙网关能够收集和上传 BLE 信标节点的广播数据，需要配置网关开启主动扫描模式并设置 RSSI 的 MAC 地址过滤，只保留指定 MAC 地址的数据；配置网关通过无线热点连接手机并配置其以 MQTT 协议上传数据到 MQTT Broker 的指定 Topic RssUpload，发送间隔为 1s，QoS（消息服务质量）为 0 且发送的数据格式为 Json。MQTT 服务器采用开源的公用服务器 EMQX，设置网关的连接方式为 TCP 连接。蓝牙网关的配置过程如图 4.2 所示。

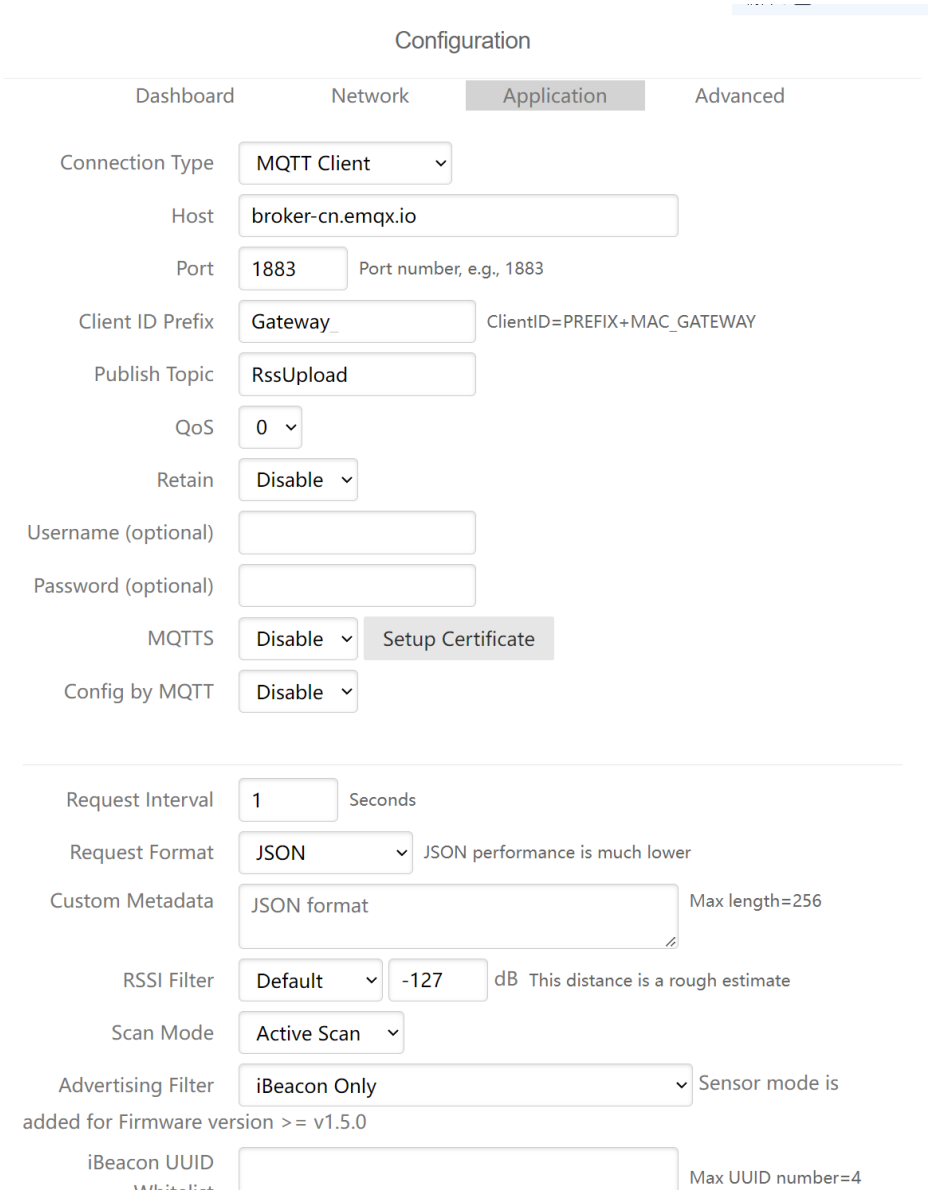


图 4.2 蓝牙网关的配置

4.2 指纹数据库的设计

位置指纹法需要存储每个定位区域的 RSSI 数据组，因此在数据库中设计有指纹表 location_info 用于记录这些数据。指纹表 location_info 主要由定位区域编号及 BLE 信标节点的信息组成。具体的设计如表 4.1 所示。

表 4.1 位置指纹数据库 location_info 表

字段	字段说明	数据类型	备注
location	定位区域编号	Integer	主键、自增
beacon_id_01	1 号节点编号	Integer	
beacon_mac_01	1 号节点 MAC 地址	Text	
beacon_rss_01	1 号节点 RSSI 数值	Integer	
beacon_id_02	2 号节点编号	Integer	
beacon_mac_02	2 号节点 MAC 地址	Text	
beacon_rss_02	2 号节点 RSSI 数值	Integer	
beacon_id_03	3 号节点编号	Integer	
beacon_mac_03	3 号节点 MAC 地址	Text	
beacon_rss_03	3 号节点 RSSI 数值	Integer	
beacon_id_04	4 号节点编号	Integer	
beacon_mac_04	4 号节点 MAC 地址	Text	
beacon_rss_04	4 号节点 RSSI 数值	Integer	
beacon_id_05	5 号节点编号	Integer	
beacon_mac_05	5 号节点 MAC 地址	Text	
beacon_rss_05	5 号节点 RSSI 数值	Integer	
beacon_id_06	6 号节点编号	Integer	
beacon_mac_06	6 号节点 MAC 地址	Text	
beacon_rss_06	6 号节点 RSSI 数值	Integer	

数据库存储采用 Android 本地 SQLite 数据库，location_info 表在 Android 设备中的存储示例如图 4.3 所示。

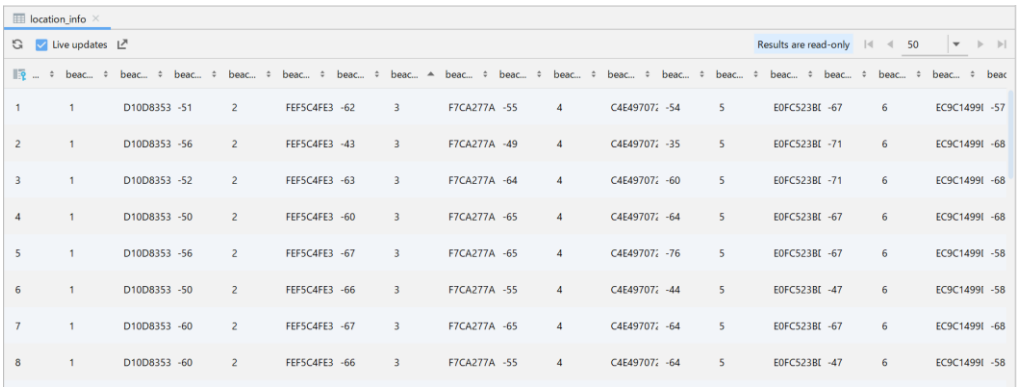


图 4.3 SQLite 数据库示例

4.3 Android APP 的开发

Android APP 主要为用户展示当前自己室内地图上的实时位置。主要设计包含以下四个部分：界面 UI 设计、MQTT 客户端的实现、数据解析模块以及 WKNN 定位算法的实现。

4.3.1 界面设计

简洁明了的 UI 设计有利于提高用户的使用体验。该系统所开发的 APP 包含两个界面：欢迎界面和地图展示界面。欢迎界面欢迎用户使用，通过点击按钮可跳转到地图展示界面；地图展示界面显示室内地图，在整体系统就绪并接收到 MQTT 服务器推送的数据时，该界面可实时刷新显示用户位置。UI 界面的具体设计如图 4.4 所示。

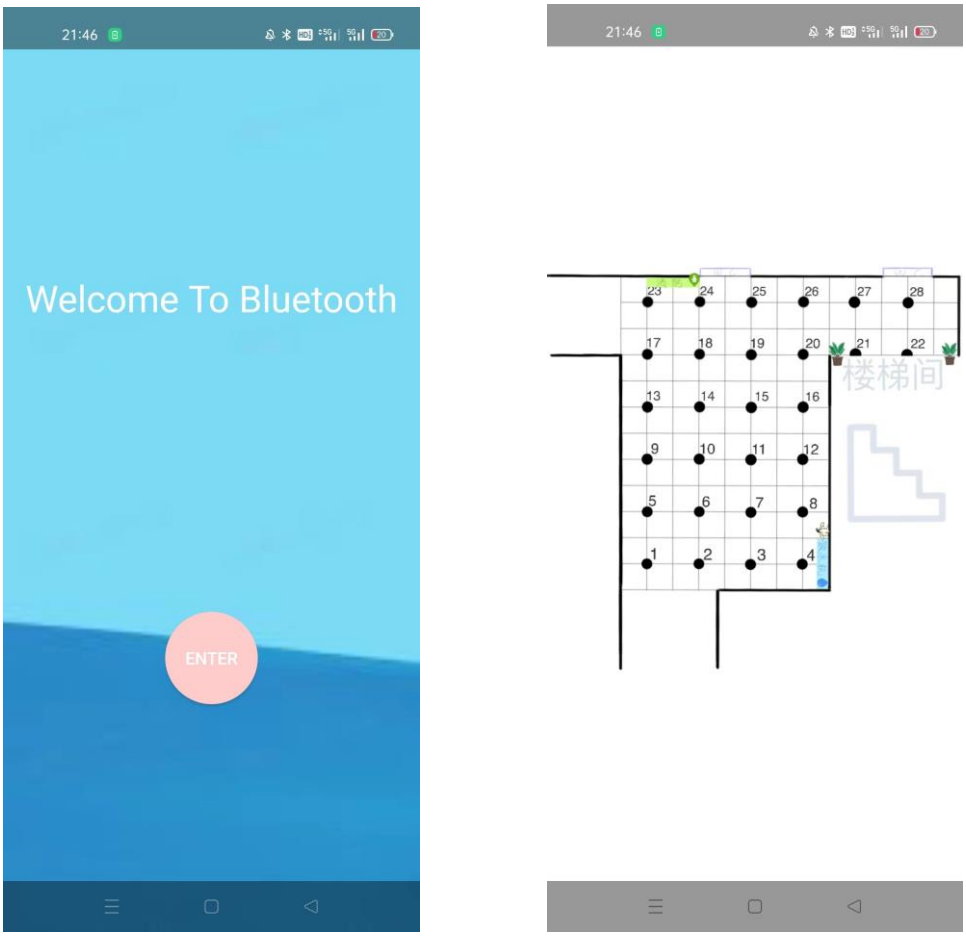


图 4.4 Android APP UI 设计

4.3.2 MQTT 客户端

Android APP 端为了能够接收 MQTT 服务器推送的数据，需要实现 MQTT 订阅客户端。Android MQTT 客户端的实现依赖于 Paho-MQTT 库，这是一个提供多语言支持的完整实现 MQTT 协议的三方库。同时，后台推送服务需要基于 Android Service 实现。由此，这里使用 Java 语言编写 MQTT Service 实现自动处理数据推送并通过 Android 本地广播机制发送数据。程序流程图如图 4.5 所示。

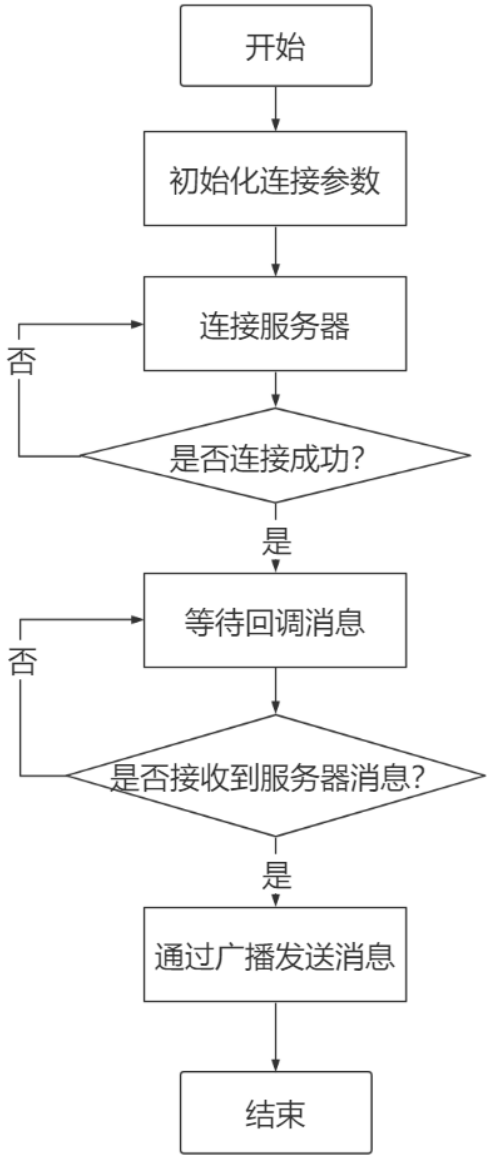


图 4.5 MQTT Client 程序流程图

程序的具体执行过程如下：首先，需要初始化 MQTT 连接参数，初始化参数的过程如图 4.6 所示。

```
String serverURI = "tcp://broker-cn.emqx.io:1883";   //服务器地址（协议+地址+端口号）
String clientId = "ANDROID_BLE_DEVICE_01";     // 由于只有一个设备，故设置固定设备号
mMqttAndroidClient = new MqttAndroidClient( context: this, serverURI, clientId);
mMqttAndroidClient.setCallback(mqttCallback);       // 设置监听订阅消息的回调
mMqttConnectOptions = new MqttConnectOptions();     // 参数配置对象
mMqttConnectOptions.setCleanSession(true);          // 设置是否清除缓存
mMqttConnectOptions.setConnectionTimeout(10);    // 设置超时时间，单位：秒
mMqttConnectOptions.setKeepAliveInterval(120);    // 设置心跳包发送间隔，单位：秒
// last will message
boolean doConnect = true;
String message = "{\"terminal_uid\":\"" + clientId + "\",\"msg\":\"Client offline\"}";
try {
    // 遗嘱发布主题
    String endWillMsgPublishTopic = "EndWillMsg";
    mMqttConnectOptions.setWill(endWillMsgPublishTopic, message.getBytes(), qos: 0, retaine
} catch (Exception e) {
    Log.d(TAG, msg: "Exception Occurred", e);
    doConnect = false;
    iMqttActionListener.onFailure( asyncActionToken: null, e);
}
```

图 4.6 初始化 MQTT 连接参数

在初始化参数后，开始连接 MQTT 服务器，连接服务器的过程如图 4.7 所示。

```
// 连接MQTT服务器
private void doClientConnection() {
    if(null != mMqttAndroidClient) {
        if (!mMqttAndroidClient.isConnected() && isConnectIsNormal()) {
            try {
                mMqttAndroidClient.connect(mMqttConnectOptions, userContext: null, iMqttActionListener);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        }
    }
}
```

图 4.7 连接 MQTT 服务器

在连接服务器成功后，需要重写接收消息回调函数，在该回调函数通过 Android 本地广播机制将接收到的推送服务转发出去。消息回调函数如图 4.8 所示。

```
@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    // Log.d(TAG, "收到" + topic + "发来的消息: " + new String(message.getPayload()));
    // 收到消息，这里弹出 Toast表示，如果需要更新 UI，可以使用广播或者 EventBus 进行发送
    // Toast.makeText(getApplicationContext(), "messageArrived: " + new String(message.getPayload()), Toast.LENGTH_LONG).show();
    String msgForTransfer = new String(message.getPayload());

    // intent 包装 bundle 对象并通过 broadcast 发送
    Intent intent = new Intent(SEND_MQTT_PAYLOAD);
    Bundle bundle = new Bundle();
    bundle.putString("MqttRevMsg", msgForTransfer);       // 键值对
    intent.putExtras(bundle);
    LocalBroadcastManager.getInstance(MyMqttService.this).sendBroadcast(intent); // 发送广播
}
```

图 4.8 接收消息回调函数

4.3.3 数据解析模块

数据解析模块主要负责从数据库和从服务器接收到的数据中获取 RSSI 数值数组。分为以下两个模块：

1）数据库端数据解析

数据库端数据解析模块负责从位置指纹数据库中获取所有定位区域的 RSSI 数值数组并组装为以数组索引值代表定位区域编号的二维数组形式便于后续的进一步处理。

获取数据库中的所有 RSSI 数值并组装为数组的函数如图 4.9 所示。

```
// 提取数据：从数据库中分别获取 4 个位置的 Rss信息数组
@SuppressLint("Range")
public static ArrayList<Integer> getRssByDataBase(SQLiteDatabase locationInfoDb) {
    ArrayList<Integer> rssList = new ArrayList<>(); // 存储查询结果的（每个位置的 Rss数值数组）
    Cursor cursor = locationInfoDb.query( table: "location_info",
            columns: null, selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: null);
    while (cursor.moveToNext()) {
        rssList.add(cursor.getInt(cursor.getColumnIndex( columnName: "beacon_rss_01")));
        rssList.add(cursor.getInt(cursor.getColumnIndex( columnName: "beacon_rss_02")));
        rssList.add(cursor.getInt(cursor.getColumnIndex( columnName: "beacon_rss_03")));
        rssList.add(cursor.getInt(cursor.getColumnIndex( columnName: "beacon_rss_04")));
        rssList.add(cursor.getInt(cursor.getColumnIndex( columnName: "beacon_rss_05")));
        rssList.add(cursor.getInt(cursor.getColumnIndex( columnName: "beacon_rss_06")));
    }
    cursor.close();
    return rssList;
}
```

图 4.9 从数据库中获取 RSSI 数值数组函数

将 RSSI 数值数组组装为二维数组的函数如图 4.10 所示。

```
// 将所有位置的 Rss数组拆分并重新组装为根据位置划分的二维数组
public static ArrayList<ArrayList<Integer>> oneDiDivideToTwoDi(ArrayList<Integer> rssListFromDataBase) {
    ArrayList<ArrayList<Integer>> rssListOfLocationList = new ArrayList<>();
    // 两层循环给二维数组赋值
    for (int i = 0; i < 28; i++) {
        ArrayList<Integer> rssTemp = new ArrayList<>();
        for (int j = 0; j < 6; j++) {
            rssTemp.add(rssListFromDataBase.get(i * 6 + j));
        }
        rssListOfLocationList.add(rssTemp);
    }
    return rssListOfLocationList;
}
```

图 4.10 组装二维数组函数

2）服务器端数据解析

服务器端推送来的数据是 Json 格式的字符串，所以需要使用 Gson 类库解析该字符串获取 RSSI 数值数组。Json 格式的字符串示例如图 4.11 所示。

```
2022-04-04 14:57:27.016 26099-26099/com.bluetooth.test D/MyMqttService: 收到RssUpload发来的消息:
{"v":1,"mid":2019,"time":4161,"ip":"192.168.41.196","mac":"E0E2E69D235C","devices":
[[0,"E0FC523BDC4C",-63,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D900014CDCC5"],
[0,"C4E497072BEF",-59,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001EF2BC5"],
[0,"FEF5C4FE3DE7",-63,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001E73DC5"],
[0,"D10D835390A2",-60,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001A290C5"],
[0,"EC9C1499D5F31",-57,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001F3D5C5"],
[0,"E0FC523BDC4C",-62,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D900014CDCC5"],
[0,"F7CA277A3602",-67,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D900010236C5"],
[0,"C4E497072BEF",-54,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001EF2BC5"],
[0,"D10D835390A2",-65,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001A290C5"],
[0,"EC9C1499D5F3",-52,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001F3D5C5"],
[0,"FEF5C4FE3DE7",-63,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001E73DC5"],
[0,"D10D835390A2",-57,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001A290C5"],
[0,"EC9C1499D5F3",-55,"0201061AFF4C000215B5B182C7EAB14988AA99B5C1517008D90001F3D5C5"]]}
```

图 4.11 Json 格式字符串示例

Gson 类库是由 Google 官方推出的用于 Json 格式字符串解析的库，该库支持的功能多且解析速度快，在实际开发中得到了广泛的应用。使用 Gson 类库解析字符串，需要先创建 Json 字符串对应的 JavaBean 类，此类的成员变量需要和 Json 字符串所包含的数据一一对应，在这里，此类的成员变量包括消息版本号、消息编号、启动时间、网关 IP 地址、网关 MAC 地址以及 BLE 信标节点的广播数据数组。JavaBean 类的定义如图 4.12 所示。

```
private int v;          // 版本号
private int mid;        // message id
private int time;       // 启动时长，以秒计算
private String ip;      // 网关的 IP
private String mac;     // 网关的 MAC地址
// 将混合数组中的所用类型都认为是字符串将变得简单
private ArrayList<ArrayList<String>> devices;    // 由 BLE广播包组成的数组
```

图 4.12 Json 格式字符串对应的 JavaBean 类

在创建 Json 字符串对应的 JavaBean 类后，需要将字符串转换为 JavaBean 对象，并从该对象中获取广播数据数组。其转换过程如图 4.13 所示。

```
// Gson解析 Json字符串，由 msgReceiver字符串转化为 beaconJsonBean对象
Gson gson = new Gson();
BeaconJsonBean beaconJsonBean = gson.fromJson(msgReceiver, BeaconJsonBean.class);
// Log.d(TAG, beaconJsonBean.toString());

// 解析 beaconJsonBean对象得到 Rss数值数组
ArrayList<Integer> rssList = new ArrayList<>(
        Arrays.asList(-100, -100, -100, -100, -100, -100));     // 解析得到的 Rss数值存储数组
ArrayList<ArrayList<String>> devices = beaconJsonBean.getDevices();  // 解析得到的 devices数组
```

图 4.13 Json 字符串转换 Javabean 类的过程

在获取广播数据数组后，需要从该数组中解析得到 RSSI 数值数组。BLE 信标节点的广播数据格式如表 4.2 所示。

表 4.2 数据格式说明

数据位数	说明	示例
Byte1	数据类型	00
Byte2-7	BLE 设备 MAC 地址	C4BE84D7770A
Byte8	RSSI 值	BA: rssi=0xBA-256=-68
Byte9-最后	广播数据	0201061AFF4C000215000018034 94C4F47 4943544543480000010002D8

按照上图所示的广播数据格式，遍历广播数据数组即可得到 RSSI 数值的数组。数据格式解析的示例如图 4.14 所示。

```
// 内层循环找到每一个 beacon广播数组
if (0 == i) {
    if (array.get(1).equals("D10D835390A2")) {
        rssList.set(0, Integer.valueOf(array.get(2)));
        break;
    }
} else if (1 == i) {
    if (array.get(1).equals("FEF5C4FE3DE7")) {
        rssList.set(1, Integer.valueOf(array.get(2)));
        break;
    }
} else if (2 == i) {
    if (array.get(1).equals("F7CA277A3602")) {
        rssList.set(2, Integer.valueOf(array.get(2)));
        break;
    }
```

图 4.14 广播数据格式解析示例

4.3.4 WKNN 定位算法

为了实现较高的定位精度，本系统选用 WKNN 定位算法。WKNN 算法在 KNN 算法的基础上，使用欧氏距离的倒数为权重，提高了欧氏距离较小的定位区域对于整体定位结果的影响，有效地提高了定位精度。WKNN 算法的程序流程图如图 4.15 所示。

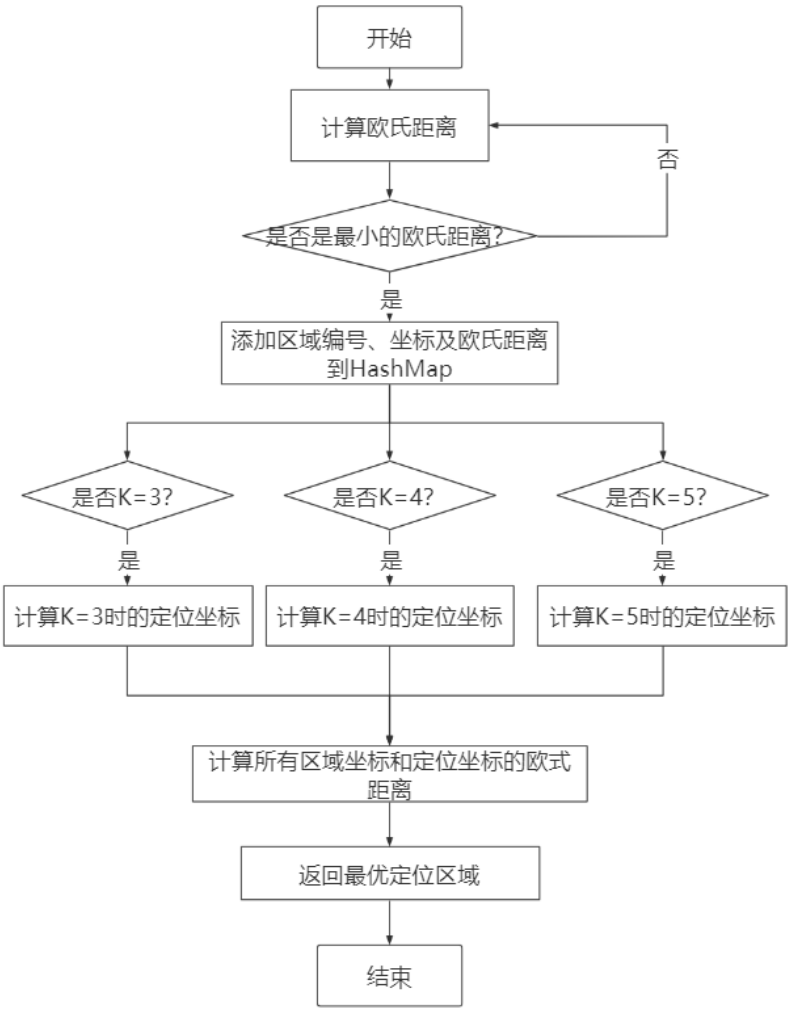


图 4.15 WKNN 算法程序流程图

WKNN 算法的具体执行过程如下：首先需要计算所有定位区域的指纹和实时收集的 RSSI 数据的欧氏距离。欧氏距离的计算函数如图 4.16 所示。所有组的欧氏距离的计算过程如图 4.17 所示。

```java
// 欧式距离的计算：计算得到距离值，并保留两位小数
private static Double calculateEuclideanDistance(ArrayList<Integer> rssListFromLocation, ArrayList<Integer> rssListFromServer) {
    double euclideanDistanceTemp;           // 累加的中间值
    double euclideanDistanceSum = 0.0;      // 每个坐标差的平方和
    double euclideanDistance;               // 欧氏距离
    for (int i = 0; i < 6; i++) {
        euclideanDistanceTemp = Math.pow(rssListFromLocation.get(i) - rssListFromServer.get(i), 2);
        euclideanDistanceSum += euclideanDistanceTemp;
    }
    euclideanDistance = Math.sqrt(euclideanDistanceSum);
    // 保留两位小数
    BigDecimal formatHelper = new BigDecimal(euclideanDistance);
    return formatHelper.setScale( newScale: 2, RoundingMode.HALF_UP).doubleValue();
}
```

图 4.16 欧式距离的计算函数

```java
// 1. 计算最小距离
ArrayList<Double> distanceList = new ArrayList<>();
for (int i = 0; i < 28; i++) {
    Double distance = calculateEuclideanDistance(rssListOfLocationList.get(i), rssListFromServer);
    distanceList.add(distance);
}
```

图 4.17 所有组的欧式距离的计算过程

在计算得到所有组的欧式距离后，需要对所有的欧氏距离进行排序得到最小的 K 个值及其定位点编号，将编号及其欧氏距离的值作为键值对添加到 HashMap 中。其计算过程和添加过程如图 4.18 所示。

```java
// 2. 找到 distanceList数组中最小的 K个值及其编号（index + 1）
HashMap<Integer, Double> minDistanceMap = new HashMap<>();
for (int i = 0; i < K; i++) {
    Double minElement = Collections.min(distanceList);
    int indexOfMinElement = distanceList.indexOf(Collections.min(distanceList));
    minDistanceMap.put(indexOfMinElement + 1, minElement);
    distanceList.set(indexOfMinElement, 1000.0);    // 将 distanceList中已找到的最小值设为一个相对无穷大
}
Log.d(TAG, minDistanceMap.toString());

// 3. 取到最小的 K个点及其定位坐标和 NN距离
ArrayList<CoordinateBean> minCoordinateList = new ArrayList<>();
for (Integer key : minDistanceMap.keySet()) {
    minCoordinateList.add(CoordinateRecord.coordinateRecord.get(key - 1));
}
Log.d(TAG, String.valueOf(minCoordinateList.size()));

ArrayList<Double> minValues = new ArrayList<>(minDistanceMap.values());
Log.d(TAG, minValues.toString());
```

图 4.18 HashMap 的添加过程

在得到包含了 K 个最小定位点及其欧氏距离的 HashMap 后，根据定位点编号获取对应的坐标，并传入 WKNN 计算函数计算可得最终的定位坐标。WKNN 计算函数示例如图 4.19 所示。最终待定位点坐标的计算示例如图 4.20 所示。

```
// WKNN K = 3的计算
private static double calculateWKNN3(double minValue01, double minValue02, double minValue03,
                                     float p01, float p02, float p03) {
    double temp = (1 / (1 / minValue01 + 1 / minValue02 + 1 / minValue03));
    // Log.d(TAG, String.valueOf(temp));
    double result = ((1 / minValue01) * p01 + (1 / minValue02) * p02 + (1 / minValue03) * p03) * temp;
    // Log.d(TAG, String.valueOf(result));
    BigDecimal formatHelper = new BigDecimal(result);
    return formatHelper.setScale( newScale: 2, RoundingMode.HALF_UP).doubleValue();
}
```

图 4.19 WKNN 的计算过程示例（K=3）

```
// 4. WKNN的计算：得到最小坐标（欧氏距离的倒数加权）
double x = 0.0;
double y = 0.0;
if (3 == K) {
    x = calculateWKNN3(minValues.get(0), minValues.get(1), minValues.get(2),
            minCoordinateList.get(0).getX(), minCoordinateList.get(1).getX(), minCoordinateList.get(2).getX());
    y = calculateWKNN3(minValues.get(0), minValues.get(1), minValues.get(2),
            minCoordinateList.get(0).getY(), minCoordinateList.get(1).getY(), minCoordinateList.get(2).getY());
```

图 4.20 最终待定位点坐标的计算过程示例（K=3）

在计算得到最终的待定位点坐标后，通过遍历计算所有定位点和待定位点坐标之间的欧氏距离，找到最接近待定位点坐标的定位点编号，这就是最终将在地图上展示的人员感知位置。寻找最优定位点的过程如图 4.21 所示。

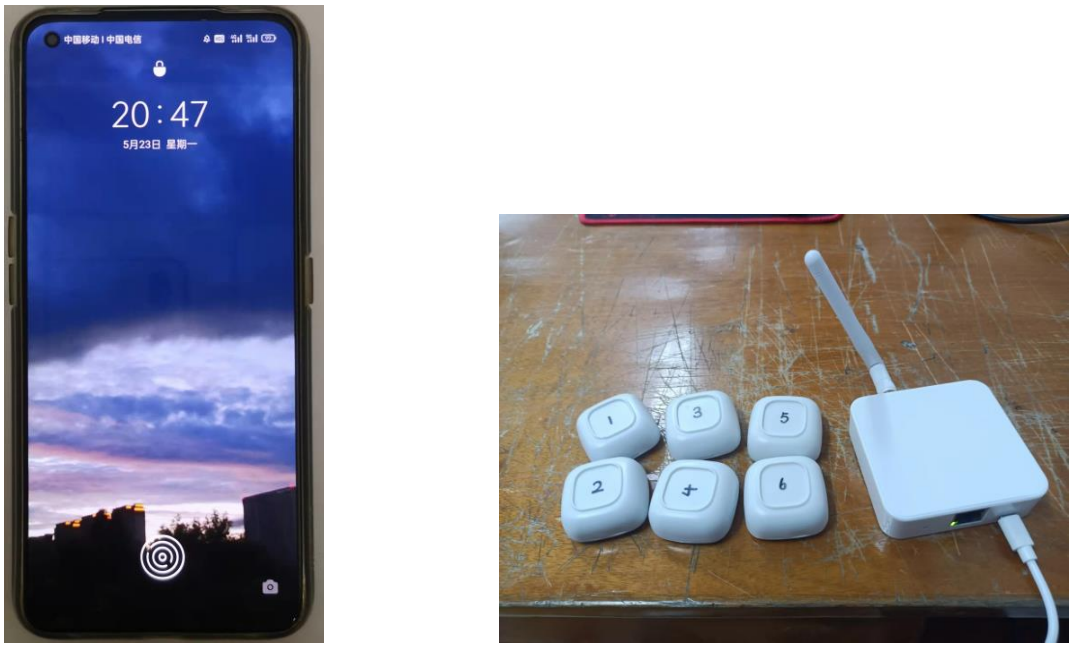
```
// 5. 根据得到的最小坐标计算最终的定位点编号
double[] minDistanceForId = new double[28];
for (int i = 0; i < 28; i++) {
    minDistanceForId[i] = calculateDistance(x, y, CoordinateRecord.coordinateRecord.get(i).getX(),
            CoordinateRecord.coordinateRecord.get(i).getY());
}
Log.d(TAG, Arrays.toString(minDistanceForId));
Log.d(TAG, String.valueOf(getMinIndex(minDistanceForId) + 1));
return getMinIndex(minDistanceForId) + 1;
```

图 4.21 寻找最优定位点的计算过程

第 5 章 系统测试及结果分析

5.1 测试设备与环境

系统测试基于上述搭建的室内人员位置感知实验平台。所采用的移动端设备为搭载 Android 操作系统的 Realme Q2 Pro 智能手机，蓝牙设备采用国内公司四月兄弟出品的 aBeacon 信标节点和 BLE 网关设备。测试的室内环境为重庆邮电大学第八教学楼一楼楼梯间拐角处。测试所采用的移动设备和蓝牙设备如图 5.1 所示。其余部分详见第 4 章室内人员位置感知实验平台的搭建部分。



(a)移动测试设备　　(b)蓝牙设备

图 5.1 测试设备实物图

5.2 测试结果与分析

为了了解系统的运行效果并对比各位置感知算法的精度，在开发完成整个系统后，需要在搭建的室内人员位置感知实验平台对系统进行测试。

本次测试分为三组进行，分别测试 NN 算法、KNN 算法（K=3/4/5）、WKNN 算法（K=3/4/5）的位置感知精度。对于每组实验，选取了 20 个测试点，包括 16 个指纹采集区域中心所在点和 4 个指纹采集区域交叉点，具体的测试点信息如表 5.1 所示。算法测试的过程示例如图 5.2 所示。

(a)定位偏差示意图　　(b)定位准确示意图

图 5.2 测试过程示例图

图 5.2(a)表示位置感知结果偏差示意图，实际所在位置由定位图标标识为 3 号区域，而位置感知结果为红色圆圈标识为 2 号区域。图 5.2(b)表示位置感知结果准确示意图，实际所在位置和位置感知结果均为 27 号区域。

实验数据处理阶段需要计算每组实验的感知精度，感知精度由位置感知平均误差 e 和区域感知精度 u 组成。其计算如公式 5.1 和公式 5.2 所示。

$$e=\frac{1}{20}\sum_{i,j=1}^{20}\sqrt{(x_i-x_j)^2+(y_i-y_j)^2} \tag{5.1}$$

$$u=\frac{p}{p+q} \tag{5.2}$$

公式(5.1)中，(x_i,y_i)表示测试参考点坐标，(x_j,y_j)表示定位结果点坐标；公式(5.2)中，p 表示定位准确的测试点数量，而 q 表示定位偏差的测试点数量。

NN 算法的测试结果如表 5.1 所示。

表 5.1 NN 算法测试结果

测试点编号	实际坐标(m)/区域	定位区域
1	(1, 1)/1	1
2	(5, 1)/3	4
3	(3, 3)/6	12
4	(7, 3)/8	8
5	(1, 5)/9	18
6	(5, 5)/11	16
7	(3, 7)/14	19
8	(7, 7)/16	16
9	(1, 9)/17	17
10	(5, 9)/19	27
11	(3, 11)/24	23
12	(7, 11)/26	20
13	(9, 9)/21	21
14	(11, 9)/22	22
15	(9, 11)/27	27
16	(11, 11)/28	21
17	(4, 2)/(2/3/6/7)	2
18	(4, 10)/(18/19/24/25)	18
19	(6, 4)/(7/8/11/12)	11
20	(6, 8)/(15/16/19/20)	19
e(m)/u(%)	-	-/55

KNN 算法的测试结果如表 5.2 所示。

表 5.2 KNN 算法测试结果

测试点编号	实际坐标(m)/区域	定位结果(m)/定位区域		
		K=3	K=4	K=5
1	(1, 1)/1	(1.98, 1.74)/1	(1.05, 1.94)/1	(1.77, 1.74)/1
2	(5, 1)/3	(4.82, 2.06)/7	(4.15, 1.93)/3	(1.94, 3.41)/5
3	(3, 3)/6	(3.93, 3.84)/6	(2.02, 3.61)/6	(1.79, 5.97)/9
4	(7, 3)/8	(6.04, 2.92)/7	(6.25, 2.08)/8	(6.17, 3.99)/8
5	(1, 5)/9	(1.19, 4.72)/9	(1.67, 1.99)/9	(1.81, 5.62)/9

续表 5.2 KNN 算法测试结果

测试点编号	实际坐标(m)/区域	定位结果(m)/定位区域		
		K=3	K=4	K=5
6	(5, 5)/11	(7.02, 3.94)/8	(4.38, 2.06)/6	(2.79, 5.96)/10
7	(3, 7)/14	(2.24, 6.35)/14	(2.24, 6.28)/14	(2.88, 6.32)/14
8	(7, 7)/16	(9.18, 9.85)/20	(7.74, 6.05)/16	(6.09, 8.29)/20
9	(1, 9)/17	(1.92, 8.03)/17	(1.13, 10.75)/17	(2.75, 10.64)/23
10	(5, 9)/19	(3.17, 6.64)/14	(3.30, 8.12)/14	(4.47, 7.95)/15
11	(3, 11)/24	(2.26, 10.56)/24	(2.0, 8.19)/13	(2.66, 8.87)/18
12	(7, 11)/26	(5.31, 9.74)/19	(6.17, 10.19)/26	(7.61, 10.12)/26
13	(9, 9)/21	(8.43, 10.03)/21	(8.02, 9.81)/21	(8.03, 9.91)/无
14	(11, 9)/22	(10.16, 9.59)/22	(10.24, 7.1)/无	(9.03, 8.47)/无
15	(9, 11)/27	(8.58, 10.13)/27	(8.11, 10.91)/27	(8.18, 8.85)/无
16	(11, 11)/28	(10.12, 10.83)/28	(10.39, 10.51)/28	(9.05, 10.93)/28
17	(4, 2)/(2/3/6/7)	(3.15, 2.96)/6	(1.92, 0.78)/1	(3.49, 2.42)/6
18	(4, 10)/(18/19/24/25)	(4.55, 10.68)/24	(3.06, 9.01)/18	(4.62, 9.89)/19
19	(6, 4)/(7/8/11/12)	(4.01, 4.45)/6	(5.77, 4.23)/11	(6.55, 4.97)/12
20	(6, 8)/(15/16/19/20)	(5.16, 8.89)/19	(8.27, 10.41)/无	(6.86, 7.34)/16
e(m)/u(%)	-	1.585/70	1.688/65	1.982/55

WKNN 算法的测试结果如表 5.3 所示。

表 5.3 WKNN 算法测试结果

测试点编号	实际坐标(m)/区域	定位结果(m)/定位区域		
		K=3	K=4	K=5
1	(1, 1)/1	(1.88, 1.62)/1	(1.10, 1.87)/1	(1.21, 1.77)/1
2	(5, 1)/3	(4.82, 2.06)/3	(4.11, 1.93)/3	(5.87, 1.21)/3
3	(3, 3)/6	(3.65, 3.79)/6	(2.87, 3.49)/6	(3.26, 3.51)/6
4	(7, 3)/8	(6.11, 3.32)/8	(7.53, 3.78)/8	(5.47, 3.89)/7
5	(1, 5)/9	(1.87, 5.33)/9	(1.23, 2.48)/5	(1.43, 4.23)/9
6	(5, 5)/11	(2.02, 4.0)/6	(4.13, 4.16)/11	(4.21, 5.98)/11
7	(3, 7)/14	(2.24, 6.35)/14	(2.15, 7.79)/14	(1.88, 8.21)/13
8	(7, 7)/16	(5.32, 7.64)/15	(6.32, 7.6)/16	(6.09, 7.83)/16
9	(1, 9)/17	(1.43, 9.62)/17	(1.22, 10.75)/23	(2.73, 9.21)/18
10	(5, 9)/19	(4.37, 9.97)/19	(2.7, 9.89)/18	(4.34, 8.18)/19
11	(3, 11)/24	(3.42, 10.55)/24	(3.82, 10.11)/24	(1.66, 8.87)/23
12	(7, 11)/26	(6.53, 10.24)/26	(6.03, 10.87)/26	(7.96, 10.31)/26
13	(9, 9)/21	(8.26, 9.93)/21	(8.18, 10.92)/21	(7.1, 10.12)/26
14	(11, 9)/22	(11.75, 8.35)/22	(10.53, 8.11)/22	(10.63, 9.76)/22
15	(9, 11)/27	(9.28, 10.21)/27	(9.86, 10.25)/27	(8.12, 10.74)/27
16	(11, 11)/28	(10.57, 10.29)/28	(9.21, 9.82)/28	(9.05, 10.61)/28
17	(4, 2)/(2/3/6/7)	(3.26, 1.52)/2	(4.82, 2.54)/7	(3.1, 3.71)/6
18	(4, 10)/(18/19/24/25)	(3.39, 10.71)/24	(3.26, 10.71)/24	(2.98, 8.87)/18
19	(6, 4)/(7/8/11/12)	(5.26, 4.57)/11	(6.67, 3.21)/8	(5.42, 4.76)/11
20	(6, 8)/(15/16/19/20)	(7.81, 8.86)/16	(6.72, 10.28)/21	(5.21, 9.48)/19
e(m)/u(%)	-	1.389/85	1.430/80	1.424/75

上述表 5.1、表 5.2 和表 5.3 分别详细记录了 NN 算法、KNN 算法和 WKNN 算法的测试结果，并给出了每组测试的平均定位误差和区域感知精度。KNN 算法和 WKNN 算法的测试点定位误差统计图分别如图 5.3 和图 5.4 所示。

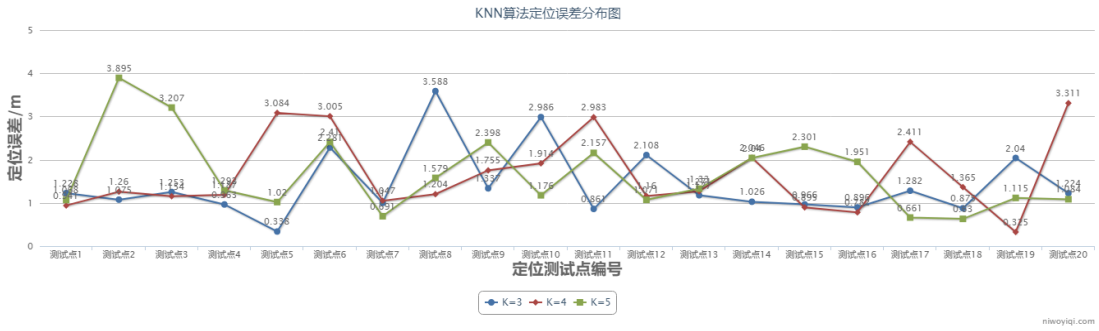


图 5.3 KNN 算法测试点定位误差统计图

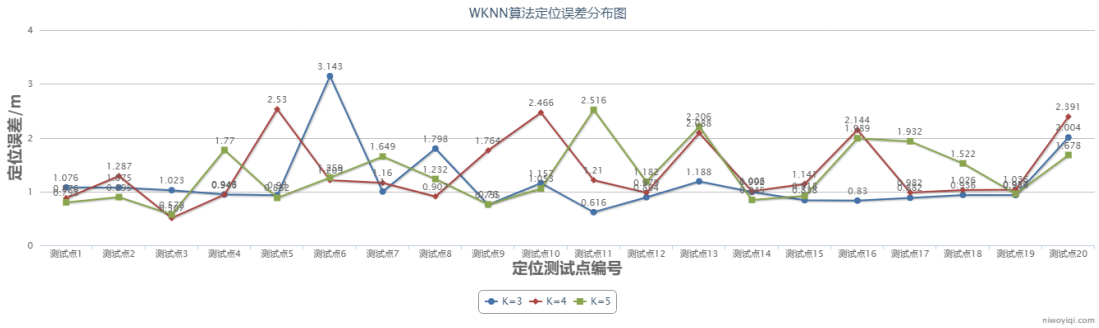


图 5.4 WKNN 算法测试点定位误差统计图

KNN 算法和 WKNN 算法在分别取最优 K 值时的定位误差分布如图 5.5 所示。

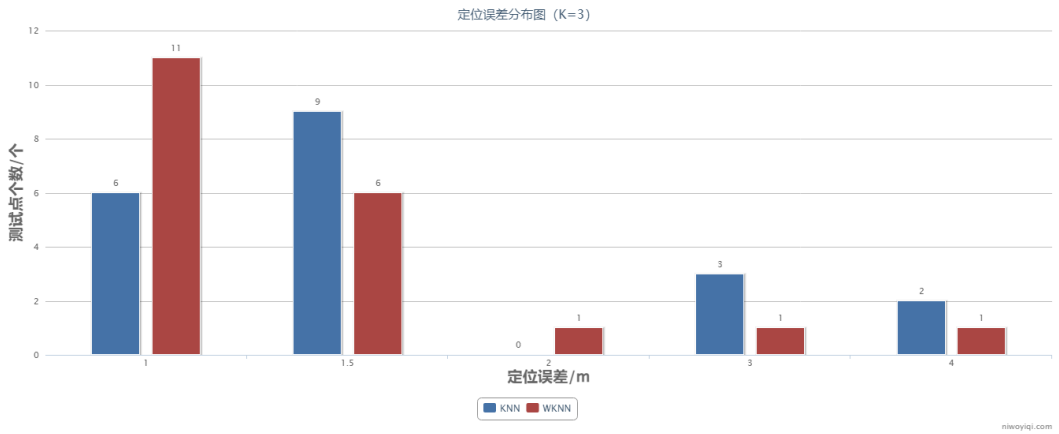


图 5.5 最优 K 值下定位误差分布图

由上述表格可知：在本次实验中，NN 算法的区域感知精度较低，只有 55%；KNN 算法在 K=3 和 K=4 时的位置感知精度均达到了 60%以上，分别为 70%和 65%，而在 K=5 时的位置感知精度不高，仅仅为 55%；WKNN 算法在 KNN 算法的基础上位置感知精度有了较大的提高，在 K=3 和 K=4 时，精度达到了 80%以上，分别为 85%和 80%，在 K=5 时精度也有 75%。

综合上述表格、定位误差统计图及最优 K 值时定位误差分布图可得如下结论：KNN 算法在 K=3 时的平均定位误差最小，为 1.585m，最大定位误差 3.588m，为测试点 8 的测试结果，最小定位误差 0.338m，为测试点 5 的测试结果；定位误差在 2m 以上的测试点有 5 个，定位误差在 1.0m~2m 之间的测试点有 9 个，定位误差在 1m 以下的测试点有 6 个。由此可知 KNN 算法在 K=3 时的测试点定位误差有 70%在 2m 以下。WKNN 算法在 K=3 时的平均定位误差最小，为 1.389m，最大定位误差 3.143m，为测试点 6 的测试结果，最小定位误差 0.616m，为测试点 11 的测试结果；定位误差在 2m 以上的测试点仅有 2 个，定位误差在 1.5m~2m 之间的测试点有 1 个，定位误差在 1.0m~1.5m 之间的测试点有 6 个，定位误差在 1m 以下的测试点有 11 个。由此可知 WKNN 算法在 K=3 时的测试点定位误差有 85%在 1.5m 以下，有 90%在 2m 以下。

由上述对测试结果的分析可知，在该室内环境下，选用 WKNN 算法且 K=3 时的人员位置感知效果最佳，平均误差仅为 1.389m，位置感知精度可达 85%。由此可见，WKNN 算法使用欧氏距离倒数加权的方法有效地提高了系统精度；在室内环境下划分的定位区域数量适中的情况下，选用 K=3 能够得到较低的平均定位误差。该系统满足一般室内人员位置感知系统的精度要求，通过实验测试了常用的位置感知算法的性能，可为相关系统的设计实现与参数的选取提供一定的参考。

第 6 章 总结与展望

6.1 主要工作与创新点

本文基于 BLE 信标节点和 BLE 网关搭建室内人员位置感知实验平台，采用无源位置指纹法构建指纹数据库，并采用 WKNN 定位匹配算法计算待感知人员的位置坐标，最后实现了可实时刷新待感知人员位置的 APP。本文的主要工作如下：

1）研究了人员感知技术的发展，对各种人员感知技术进行了总结和分析，并重点介绍了基于 BLE 的室内人员感知技术及其感知方法并分析了各种方法的优缺点和使用场景，最后给出了室内人员感知系统的评价指标。

2）详细分析了室内人员感知系统的设计需求并以此为标准设计实现了基于 BLE 的无终端室内人员感知系统。该系统包含人员感知实验平台、位置指纹数据库和 Android APP 三个部分。人员感知实验平台部署于典型的室内环境下，其所取得的实验结果经过一定的处理后可推广应用于其他室内场景；位置指纹数据库记录单人员场景下不同位置的 RSSI 信号抖动；Android APP 应用绘制了典型室内场景下的地图，并可通过 WKNN 定位算法实时匹配刷新待感知人员位置，极大地方便了用户的使用，提高了用户的体验。

3）基于人员位置感知实验平台测试了系统的主要性能指标。其位置感知精度可达 80%以上，基本满足室内位置服务的精度要求。且该系统具有功耗低、覆盖范围广、部署简单且成本低等优点，为室内人员感知系统的设计与实现提供了一个可靠实用的范例。