

# 中南大学

## 《物联网定位技术》

### 课程实验报告



实验名称 \_\_\_\_\_ 物联网定位技术实验

学生姓名 \_\_\_\_\_ 白明强、杜斯博、王云鹏

专业班级 \_\_\_\_\_ 物联网 1802 班

指导教师 \_\_\_\_\_ 张士庚

学 院 \_\_\_\_\_ 计算机学院

# 实验一：GPS 和北斗实验

## 一、实验目标及环境

掌握 GPS/北斗定位的基本原理，了解相关卫星信号的格式。通过上位机软件与实验平台通信，了解 NMEA0183 数据格式，以及每条语句的组成和意义。

## 二、实验要求

1. (课堂完成) 掌握基于指纹的无线室内定位技术的基本原理；
2. (撰写实验报告) 实验可以分小组合作完成，每个小组提交一份实验报告，包括系统设计方案，程序源码或伪码，实验结果。

## 三、实验平台与仪器

1. 实验室北斗实验箱，北斗/GPS 综合实验箱 BGE2200A，GNSS 卫星信号转发器 RT380，实验测控计算机（通用）

## 四、实验内容

GPS/北斗天线接收来自卫星的高频载波信号，信号通过电缆传送到实验平台，实验平台对信号进行处理，解析出导航电文、载波相位等数据，并通过 RS232 串口发送给上位机。用户可以通过上位机输出的数据，对 NMEA0183 语句进行解析，并对每条语句中所涉及的信息有所了解，有针对性的提取自己有用的数据信息，比如时间信息，卫星信息，定位信息、速度信息等。NMEA 0183 是美国国家海洋电子协会（National Marine Electronics Association）为海用电子设备制定的标准格式。目前已成了 GPS 导航设备统一的 RTCM（Radio Technical Commission for Maritime services）标准协议。GPS 接收机上电后，会自动通过串口或 USB 口发送 NMEA0183 格式的数据包，它是一组包含有各种地理位置信息的字符串。

字符串格式为：

\$信息类型，xxx，xxx，xxx，xxx，xxx，xxx，xxx，信息类型为：

GPGLL：可见卫星信息；

GPGLL：地理定位信息；

GPRMC：推荐最小定位信息；

GPVTG：地面速度信息；

GPGLL：GPS 定位信息；

GPGLL：当前卫星信息；

每行开头的字符都是 '\$'，接着是信息类型，后面是数据，以逗号分隔开。一行完整的数据如下：

下：

\$GPRMC,062363.00,A,2236.33923,N,11402.35855,E,0.304,306.80,020411,,A\*

目前的北斗的数据格式，也是以 NMEA0183 数据格式方式进行输出，为了区分北斗卫星的输出格式，在语句前面以\$BD开头的，而 GPS 的语句前面是以\$GP开头的，组合导航的话，语句前面是以\$GN开头的。

首先，在电脑上安装好实验箱所需要的驱动程序。将实验台接上电源，并通过串口连接到上位机，打开实验台电源开关。运行软件程序，进入软件主界面；在试验箱上选择“联合定位”。

打开北斗实验软件，选择实验二，点击开始按钮即可看到原始的 NMEA 数据、星空图及柱状图。

点击开始记录，相关的原始数据被保存到 DataRecord 目录下。

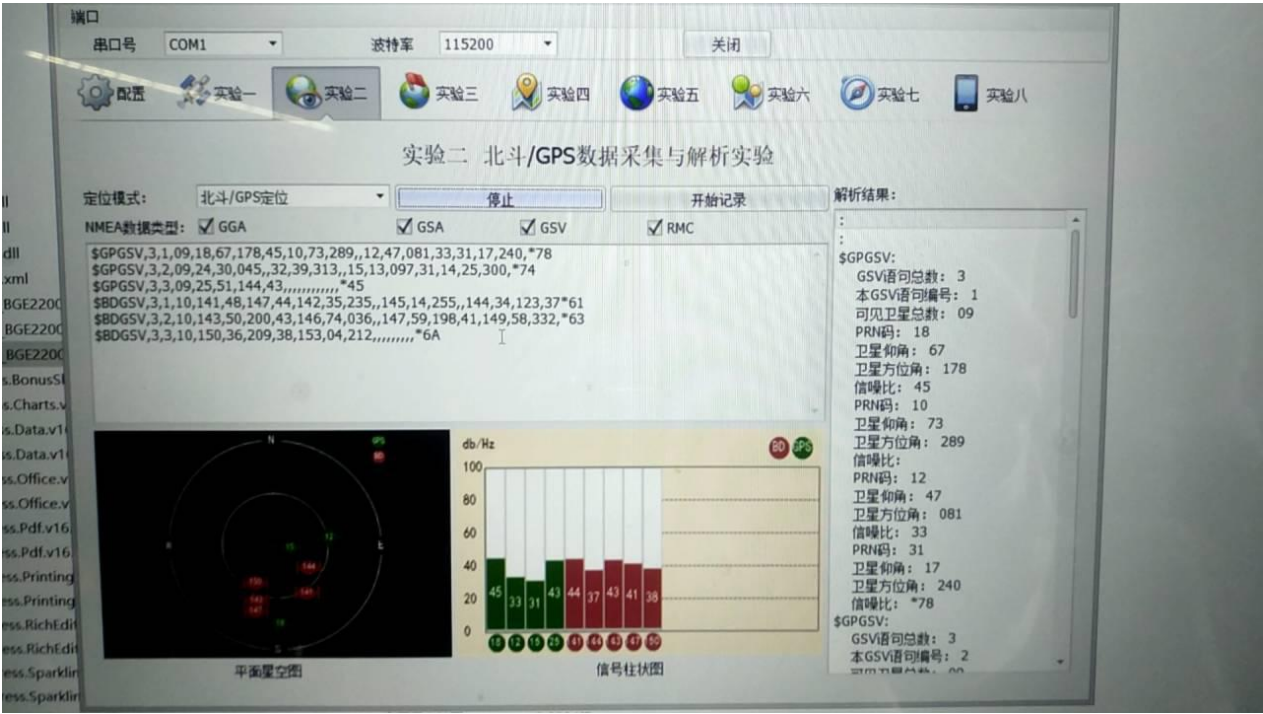


图 1 记录北斗实验软件获取的原始数据

## GPGGA GPS 定位数据:

数据详解: ji

Field	Meaning
0	Message ID \$GPGGA
1	UTC of position fix                      UTC 时间, 格式为 hhmmss.sss;
2	Latitude                                      纬度, 格式为 ddmm.mmmm(第一位是零也将传送);
3	Direction of latitude: N: North S: South                                      纬度半球, N 或 S(北纬或南纬)
4	Longitude                                    经度, 格式为 dddmm.mmmm(第一位零也将传送);
5	Direction of longitude: E: East W: West                                    经度半球, E 或 W(东经或西经)
6	GPS Quality indicator:                      定位质量指示, 0=定位无效, 1=定位有效; 0: Fix not valid 1: GPS fix 2: Differential GPS fix, OmniSTAR VBS 4: Real-Time Kinematic, fixed integers 5: Real-Time Kinematic, float integers, OmniSTAR XP/HP or Location RTK
7	Number of SVs in use, range from 00 through to 24+ 使用卫星数量, 从 00 到 12(第一个零也将传送)
8	HDOP    水平精确度, 0.5 到 99.9
9	Orthometric height (MSL reference)      天线离海平面的高度, -9999.9 到 9999.9 米 M 指单位米
10	M: unit of measure for orthometric height is meters      高度单位
11	Geoid separation                              大地水准面高度, -9999.9 到 9999.9 米 M 指单位米
12	M: geoid separation measured in meters                      高度单位
13	Reference station ID, range 0000-4095. A null field when any reference station ID is selected and no corrections are received1. 差分参考基站标号
14	The checksum data, always begins with *      数据校验和

GPRMC 最小定位信息:

数据详解: (加上三项表示方向的数据, 共 11 项)

Field	Meaning
0	Message ID \$GPRMC
1	UTC of position fix      UTC 时间, hhmmss(时分秒)格式
2	Status A=active or V=void      定位状态, A=有效定位, V=无效定位
3	Latitude      纬度 ddm. mmm(度分)格式(前面的 0 也将被传输)
4	Longitude      经度 dddmm. mmm(度分)格式(前面的 0 也将被传输)
5	Speed over the ground in knots 地面速率(000.0~999.9 节, 前面的 0 也将被传输)
6	Track angle in degrees (True) 地面航向(000.0~359.9 度, 以真北为参考基准, 前面的 0 也将被传输)
7	Date UTC 日期, ddmmyy(日月年)格式
8	Magnetic variation in degrees 磁偏角(000.0~180.0 度, 前面的 0 也将被传输)
9	The checksum data, always begins with *      校验和

GPVTG 地面速度信息

Field	Meaning
0	Message ID \$GPVTG
1	Track made good (degrees true) 以正北为参考基准的地面航向(000~359 度, 前面的 0 也将被传输)
2	T: track made good is relative to true north      T=真北参照系
3	Track made good (degrees magnetic) 以磁北为参考基准的地面航向(000~359 度, 前面的 0 也将被传输)
4	M: track made good is relative to magnetic north M=磁北参照系
5	Speed, in knots 地面速率(000.0~999.9 节, 前面的 0 也将被传输)
6	N: speed is measured in knots      N 速度单位 节

Field	Meaning
-------	---------

7 Speed over ground in kilometers/hour (kph)  
地面速率(0000.0~1851.8 公里/小时, 前面的 0 也将被传输)

8 K: speed over ground is measured in kph K 速度单位 公里  
每小时

9 The checksum data, always begins with \* 检验和  
GPGSV 可视卫星状态

Field	Meaning
-------	---------

0 Message ID \$GPGSV

1 Total number of messages of this type in this cycle  
总的 GSV 语句电文数

2 Message number 当前 GSV 语句号

3 Total number of SVs visible 可视卫星总数

4 SV PRN number  
PRN 码 (伪随机噪声码) 也可以认为是卫星编号

5 Elevation, in degrees, 90° maximum  
仰角(00~90 度)

6 Azimuth, degrees from True North, 000° through 359°  
方位角(000~359 度)

7 SNR, 00 through 99 dB (null when not tracking)  
信噪比(00~99dB)

8 - 11 Information about second SV, same format as fields 4  
through 7

12 - 15 Information about third SV, same format as fields 4 through  
7

16 - 19 Information about fourth SV, same format as fields 4  
through 7

20 The checksum data, always begins with \* 总和校验域

#### GPGSA 当前卫星信息

Field	Meaning
-------	---------

0 Message ID \$GPGSA

1 Mode 1, M = manual, A = automatic  
定位模式, A=自动手动 2D/3D, M=手动 2D/3D

2 Mode 2, Fix type, 1 = not available, 2 = 2D, 3 = 3D

Field Meaning	
	定位类型，1=未定位，2=2D 定位，3=3D 定位
3	PRN number, 01 through 32 for GPS, 33 through 64 for SBAS, 64+ for GLONASS
4	PDOP: 0.5 through 99.9 PDOP 综合位置精度因子（0.5 - 99.9）
5	HDOP: 0.5 through 99.9 HDOP 水平精度因子（0.5 - 99.9）
6	VDOP: 0.5 through 99.9 VDOP 垂直精度因子（0.5 - 99.9）
7	The checksum data, always begins with * 校验值

我们尝试解析 GPGSA 等数据，采用 Python 语言。

## 五、源码展示

```
import time

def getTime(string, format, returnFormat):
    return time.strftime(returnFormat, time.strptime(string, format)) # Convert date and
time to a nice printable format

def getLatLng(latString, lngString):
    lat = latString[:2].lstrip('0') + "." + "%.7s" %
str(float(latString[2:])*1.0/60.0).lstrip("0.")
    lng = lngString[:3].lstrip('0') + "." + "%.7s" %
str(float(lngString[3:])*1.0/60.0).lstrip("0.")
    return lat,lng

class NMEA0183():

    def parseGGA(self, sGsa):

print("=====GGA=====")
print("UTC 时间: "+getTime(sGsa[1], "%H%M%S.%f", "%H:%M:%S"))
lat,lng = getLatLng(sGsa[2], sGsa[4])
print("纬度: "+lat)
print("纬度半球: "+sGsa[3])
print("经度: "+lng)
print("经度半球: "+sGsa[5])
print("定位质量指示（0=定位无效，1=定位有效）: "+sGsa[6])
```

```

print("使用卫星数量: "+sGsa[7])
print("水平精确度: "+sGsa[8])
print("天线离海平面的高度（单位米）: "+sGsa[9])
print("大地水准面高度（单位米）: "+sGsa[11])

def parseRMC(self, sRMC):

print("=====RMC=====
=====")
    print("UTC 时间: "+getTime(sRMC[1]+sRMC[9], "%H%M%S.%f%d%m%y",
"%a %b %d %H:%M:%S %Y"))
    print("状态 (A=OK,V=FAILED):", sRMC[2])
    lat,lng = getLatLng(sRMC[3], sRMC[5])
    print("纬度: "+lat)
    print("纬度半球: "+sRMC[4])
    print("经度: "+lng)
    print("经度半球: "+sRMC[6])
    print("速度 (节):", sRMC[7])
    print("航向角(deg):", sRMC[8])
    print("磁偏角: ", sRMC[10])

def parseVTG(self, lines):

print("=====VTG=====
=====")

    print("以正北为参考基准的地面航向 (deg):", lines[1], lines[2])
    print("以磁北为参考基准的地面航向 (deg):", lines[3], lines[4])
    print("地面速率 (knots):", lines[5], lines[6])
    print("地面速率 (km/h):", lines[7], lines[8].partition("*")[0])

def parseGSV(self, lines):
    if lines[2] == '1': # 第一句

print("=====GSV=====
=====")
        else:

print("=====
=====")
        print("总的 GSV 语句电文数 :", lines[1])
        print("当前 GSV 语句号:", lines[2])
        print("可视卫星总数:", lines[3].lstrip("0"))
        for i in range(0, int(len(lines) / 4) - 1):

```



```

        print("PRN 码（伪随机噪声码）:", lines[4 + i * 4].lstrip("0"))
        print("仰角(00~90度):", lines[5 + i * 4].lstrip("0"))
        print("方位角(000~359度):", lines[6 + i * 4].lstrip("0"))
        print("信噪比(00~99dB):", lines[7 + i * 4].partition("*")[0])

def parseGSA(self, lines):

print("=====GSA=====")
=====
    print("定位模式, A=自动手动 2D/3D, M=手动 2D/3D :", lines[1])
    print("定位类型, 1=未定位, 2=2D 定位, 3=3D 定位 :", lines[2])
    print("卫星 PRN: ", end='')
    for i in range(0, 12):
        prn = lines[3 + i].lstrip("0")
        if prn:
            print(" ", prn, end='')
    print("\nPDOP 综合位置精度因子: ", lines[15])
    print("水平精度因子: ", lines[16])
    print("垂直精度因子: ", lines[17].partition("*")[0])

if __name__ == "__main__":
    nmea = NMEA0183()
    # nmea_string = input("请输入原始数据: ")
    nmea_string = """
        $GPRMC,032750.718,V,3554.928,N,07402.498,W,51.7,2.42,061220,,E*4E
        $GPGGA,032751.718,3554.928,N,07402.498,W,0,00,,M,M,,*5D
        $GPGLL,3554.928,N,07402.498,W,032752.718,V*3E
        $GPVTG,2.42,T,M,51.7,N,95.8,K*53
        $GPRMC,032754.718,V,3554.927,N,07402.500,W,30.0,2.52,061220,,E*44
        """
    for i in nmea_string.splitlines():
        if (i.find("GGA") != -1):
            nmea.parseGGA(i.split(","))
        if (i.find("RMC") != -1):
            nmea.parseRMC(i.split(","))
        if (i.find("VTG") != -1):
            nmea.parseVTG(i.split(","))
        if (i.find("GSV") != -1):
            nmea.parseGSV(i.split(","))
        if (i.find("GSA") != -1):
            nmea.parseGSA(i.split(","))

```

结果如下：

```
nmea0183 x
C:\Python37\python.exe C:/Users/7zq121vm/Desktop/物联网定位技术实验2020/nmea0183.py
=====RMC=====
UTC 时间: Sat Feb 01 03:27:50 2020
状态 (A=OK,V=FAILED): V
纬度: 35.9154666
纬度半球: N
经度: 74.4163333
经度半球: W
速度 (节): 51.7
航向角(deg): 2.42
磁偏角:

=====GGA=====
UTC时间: 03:27:51
纬度: 35.9154666
纬度半球: N
经度: 74.4163333
经度半球: W
定位质量指示 (0=定位无效, 1=定位有效): 0
使用卫星数量: 00
水平精确度:
天线离海平面的高度 (单位米):
大地水准面高度 (单位米):

=====VTG=====
以正北为参考基准的地面航向 (deg): 2.42 T
以磁北为参考基准的地面航向 (deg): M
地面速率 (knots): 51.7 N
```

## 五、实验收获与总结

通过本次实验我们实际上手对北斗实验平台进行操作，近距离地接触了卫星定位。通过实验课后对数据的解析处理，我们更加清晰的了解到了卫星星历文件的数据信息及其含义。对卫星的导航文件的格式信息也有了更加深入的认识，也了解到了NMEA0183在卫星通讯中的广泛应用。通过查找关于此协议的英文资料，锻炼了信息搜集的能力。

# 实验二：基于 KNN 算法室内定位系统的实现

## 一、实验背景

目前全球定位系统(GPS, Global Positioning System)是获取室外环境位置信息通过实施项目的最常用方式。但由于卫星信号容易受到各种障碍物遮挡，GPS/APGS 等卫星定位技术并不适用于室内或高楼林立的场合。目前由于室内环境下都普及了 WiFi，因此利用 WiFi 进行定位无需额外部署硬件设备，是一个非常节省成本的方法。然而 WiFi 并不是专门为定位而设计的，传统的基于时间和角度的定位方法并不适用于 WiFi。近十年来，在室内 WiFi 场景下的定位中，位置指纹法被广泛研究和采用。

基于无线信号的定位方法首先考虑的是使用 WiFi（基于 IEEE802.11 标准的 WLAN）作为基础定位设施。现在，包括智能手机、笔记本电脑在内的大部分移动通信设备都内嵌了 WiFi 模块。实际上，WiFi 已经被广泛地在室外定位与导航中使用（通过智能手机以及被维护的 wifi 热点位置与其对应的 mac 地址的数据库进行查找，很多公司有维护这样的数据库，包括 Google、Apple、Microsoft，以及 Skyhook 这样的定位服务提供商等等）。其他还有一些技术，比如用蓝牙、RFID、移动电话基站信号等，也可以用来实现室内定位，但是它们不像 WiFi 这样到处都有，因此流行程度不如 WiFi。移动电话信号并不能在所有的室内场景下都能稳定传播，使用 RFID 需要额外的安装硬件的花费，此外，基于超声波的定位技术使用在一些实验性的工作中，而实际利用超市波的商用设备很少，因此实际应用并不多。

因此，本次室内定位模拟实验，计划通过手机 app 来测试室内规定区域内不同位置的 WIFI 信号强度，建立指纹数据库，最后利用 python 提供的科学计算工具来实现 KNN 算法，验证室内位置定位的准确程度。

## 二、实验目的

1. 掌握基于指纹的无线室内定位方法，设计和实现一个基于 WIFI 的无线室内定位系统。

- 2.软件平台：Android Studio ， Python

- 3.硬件平台： 安卓智能手机及笔记本电脑

## 三、实验原理

### 1.常用的室内定位技术方案

目前主流的室内定位技术方案有：超宽带（UWB）室内定位技术、射频识别（RFID）技术、WI-FI 技术、蓝牙室内定位技术、超声波室内定位技术，也可以用来实现室内定位，但它们不像 WiFi 这样到处都有，因此流行程度不如 WiFi。移动电话信号并不能在所有的室内场景下都能稳定传播，使用 RFID 需要额外的安装硬件的花费，此外，基于超声波的定位技术使用在一些实验性的工作中，而实际利用超市波的商用设备很少，因此实际应用并不多。WiFi 广泛使用在家庭、

旅馆、咖啡馆、机场、商场等各类大型或小型建筑物内，这样使得 WiFi 成为定位领域中一个最引人注目的无线技术。通常，一个 WiFi 系统由一些固定的接入点（AP）组成，它们部署在室内一些便于安装的位置，系统或网络管理员通常知道这些 AP 的位置。能连接 WiFi 的移动设备（比如笔记本电脑、移动电话）相互之间可以直接或间接地（通过 AP）通信，因此可以考虑在通信功能外同时实现定位功能。

### **1.1 超宽带（UWB）室内定位技术**

超宽带(UWB)无线定位技术由于功耗低、抗多径效果好、安全性高、系统复杂度低，尤其是能提供非常精确的定位精度等优点，而成为未来无线室内定位技术的热点和首选。UWB 技术为一种发射功率较弱，传输速率惊人(上限达到 1000Mbps 以上)，穿透能力相对优秀，空间容量充足，而且是根据极窄脉冲下的一种无线技术，且无载波。通过这些优势，在室内定位中发挥的淋漓尽致，起到了很好的效果。

### **1.2 射频识别（RFID）技术**

它是利用电磁感应原理，通过无线激发近距离无线标签，实现信息读取的技术。射频识别距离从几厘米到十几米。RFID 用于人员定位的典型应用来自人员考勤系统的拓展，相比 UWB 定位技术，RFID 主要用于人员是否存在于某个区域的辨识，不能做到实时跟踪，并且定位应用还没有标准的网络体系。因此，不适用于大型设备的巡检、人员安全的确认等用途。

### **1.3 Wi-Fi 技术**

Wi-Fi 定位应用采用在区域内安置无线基站，根据待定位 Wi-Fi 设备的信号特征，结合无线基站的拓扑结构，综合确定待定位 Wi-Fi 设备的坐标。Wi-Fi 定位技术便于利用现有的无线设备实现定位功能。但相比于 UWB 定位来说，Wi-Fi 的安全性较差，功耗较高，频谱资源已趋近饱和，因此，不利于终端设备的长期携带和大规模应用。

### **1.4 蓝牙室内定位技术**

其具体定位原理是基于 RSSI 信号强度定位，首先在区域内铺设蓝牙信标，由 Beacon 发射信号，蓝牙设备接收信号并反馈，当设备进入范围内时，估算系统中各蓝牙设备之间的距离。通过这种技术，定位系统在确定特定设备的位置时，精确度可达到米级。蓝牙存在的问题是，蓝牙系统的稳定性跟不上，在复杂的环境下很容易被干扰，特别是声音、其他信号，还有蓝牙设备的价格一直是处于考虑的地方。

### **1.5 超声波室内定位技术**

采用反射式测距法是超声波定位最常采用的方法。该系统由一个主测距器与多个电子标签组成，主测距器一般布置于移动机器人本体上，各个电子标签则较固定一些，布置于室内空间的固定位置。定位过程如下：先由上位机发送同频率的信号给各个电子标签，电子标签接收到后又反射传输给主测距器，从而可以确定各个电子标签到主测距器之间的距离，并得到定位坐标。

## 2. WIFI 室内定位方案

本系统选用 WIFI 室内定位技术实现室内定位，有两种思路：

### 2.1 通过信号到达的时间

即通过无线 ap 发送的光电信号到达定位点的时间，获取当前位置到达各个 ap 之间的距离，从而获取定位。实际应用中，由于室内空间相对狭小，范围比较小，所以各个点到无线 ap 之间的距离差异没有那么大，导致无线信号到达各个点的时间会是一个极小的值，即使可以获取到，但是误差会很大，不能对不同点的信号到达时间做到有效的区分，至于使用此方案进行定位，更是不切实际。

### 2.2 通过接收信号的强度

通过在室内空间内根据无线接收设备接收到的来自不同无线接入点（ap）的不同 wifi 信号强度，对接收设备进行定位。具体来说：通过射线跟踪技术获得室内位置的接收信号强度（RSS），得到离线指纹库，其中指纹是指一个 RSS 向量与一个位置的对应，然后通过模拟目标在室内的运动轨迹，以及轨迹点处对应的 RSS，得到在线测试数据，最后通过 wifi 指纹结合 K 最近邻算法（KNN），获取与当前指纹最邻近的 K 个指纹估计当前位置，通过最邻近的 k 个指纹的位置，预估当前轨迹点所在的位置。

基于实际来看，方案二可行性较大，本文讨论的正是使用此方案实现的 wifi 室内定位系统。

## 3. WIFI 室内定位系统需求分析

基于 WIFI 指纹的室内定位系统，通过无线设备到从各个 ap 接收到的信号强度，在室内环境中实现定位：针对一个实际场所的测试结果（包括实际场所的 AP 的布置状况，如果针对不清楚位置的布置，要说明 ap 的布置数量），最后实际测试的定位精度或者准确度要加以分析。具体分为以下三个步骤：

### 3.1 WIFI 样本采集

WIFI 样本数据库的建立是 WIFI 定位的基础，因而系统应该能够采集 WIFI 场强信号，并将采集到的 WIFI 数据进行存储。

### 3.2 WIFI 定位的需求分析

WIFI 定位是系统的核心模块，包括室内固定事物定位和移动用户定位两部分。都需要用到 WIFI 指纹数据库，定位时，对采集到的数据进行过滤，根据过滤后定位数据从样本数据库中取得相应样本数据，调用定位算法得到定位结果。

### 3.3 动态追踪的需求分析

动态追踪模块的实现基于 WIFI 定位算法，能够实时显示用户动态轨迹，因而系统需要采集用户当前位置的 WIFI 场强信号，并根据采集到的 WIF 数据调用 WIFI 定位算法，得到用户当前位置的估计值，然后通过用户当前定位结果以及前几次的定位结果，调用动态追踪算法，从而估计出用户的动态轨迹。

另外，WIFI 室内定位对定位精度有一定的要求，要求定位精度应该达到 1 到 3 米，在基于 WIFI 的室内定位中，定位精度是挑战。要达到一定的精度，主要依赖两方面：（1）选择合适的算法；（2）大量的数据采样，对采样的数据准确度要有很高的要求。以上两点是实现准确定位的关键。

## 4. 基于 K 近邻算法的 WIFI 指纹定位

“位置指纹”把实际环境中的位置和某种“指纹”联系起来，一个位置对应一个独特的指纹。这个指纹可以是单维或多维的，比如待定位设备在接收或是发送信息，那么指纹可以是这个信息或信号的一个特征或多个特征（最常见的是信号强度）。

位置指纹可以是多种类型的，任何“位置独特”的（对区分位置有帮助的）特征都能被用来做为一个位置指纹。比如某个位置上通信信号的多径结构、某个位置上是否能检测到接入点或基站、某个位置上检测到的来自基站信号的 RSS（接收信号强度）、某个位置上通信时信号的往返时间或延迟，这些都能作为一个位置指纹，或者也可以将其组合起来作为位置指纹。

位置指纹定位技术主要包括两部分：离线训练阶段和在线定位阶段。

### 4.1 离线训练阶段

离线训练阶段的主要任务是建立一个位置指纹数据库要建立合适的指纹数据库。必须首先选择参考节点 RP(Rference Point)的位置，然后将每个参考节点处测量到的来自各个 AP(Access Point)的信号特征参数记录在数据库中，这个数据库也可以称作位置指纹地图(Radio Map)。我们选择接收信号强度 RSS(Received Signal Strength)均值作为信号特征参数。

### 4.2 在线定位阶段

在线定位阶段利用移动站 MS(Mobile Station)测得在某一位置处的信号特征参数(本文中选用 Rss 均值)，

通过相应的匹配算法，根据实测数据与 Radio Map 中存储数据的比较分析，搜索出一组和测量点相匹配的存储散据，进而估计用户的实际位置。

位置指纹法可以看作是分类或回归问题，监督式机器学习方法可以从数据中训练出一个从特征到标签的映射关系模型。KNN 是一种很简单的监督式机器学习算法，可以用来做分类或回归。

在利用 KNN 定位法之前，首先要计算测量得到的 RSS 均值矩阵 $[s_1, s_2, \dots, s_n]$ 和数据库中的 RSS 均值矩阵 $[S_1, S_2, \dots, S_N]$ 之间的距离。设 RadioMap 中共有  $m$  个参考点。共有  $n$  个 AP， $S_{ij}$  为 RadioMap 中第  $i$  个参考点处的第  $j$  个 AP 的 RSS 均值， $S_j$  为在线定位阶段测量得到的第  $j$  个 AP 的 RSS 均值， $i=1, 2, \dots, m, j=1, 2, \dots, n$ 。向量之间的距离定义如式(1)所示

$$L_q = \left( \sum_{j=1}^n |s_j - S_{ij}|^q \right)^{\frac{1}{q}}$$

公式 (1)

其中， $q=1$  时称  $L_{q1}$  为曼哈顿距离， $q=2$  时称  $k$  为欧几里德距离。

KNN(算法就是在上面的  $L_{qi}$  中从小到大选择  $K$  个参考点。然后利用式(2)计算其平均坐标作为测试点的估计位置。

$$(\hat{x}, \hat{y}) = \frac{1}{K} \sum_{i=1}^K (x_i, y_i)$$

公式 (2)

其中,  $(x_i, y_i)$  表示  $K$  个参考点中的第  $i$  个所对应的物理坐标;  $(x^{\wedge}, y^{\wedge})$  为测试点的估计坐标。当  $K=1$  时, 算法退化为最近邻(NN)算法。

### 三、实验内容

实验分为两步:

1. 获取位置指纹信息、建立指纹数据库
2. 利用 KNN 算法估算某位置的具体坐标。

#### 1.第一阶段-通过终端获取指纹特征

第一个阶段我们采用基于射频指纹的定位方法, 移动终端需要获得周围 AP 的 RSSI 指纹特征, 小组采用了 Android 编程的形式来完成, 因为目前的智能手机都提供了 WIFI 功能, 其系统接口也相对简单。

在初始时, 选择当前场景的地图。并输入地图尺寸, 程序会在地图上标记每次收集指纹的位置。在收集数据的阶段, 需要手动输入每次的位置, 当前位置的 AP 数量和信号强度就会被指纹数据库记录。



图 1 选择当前场景的地图

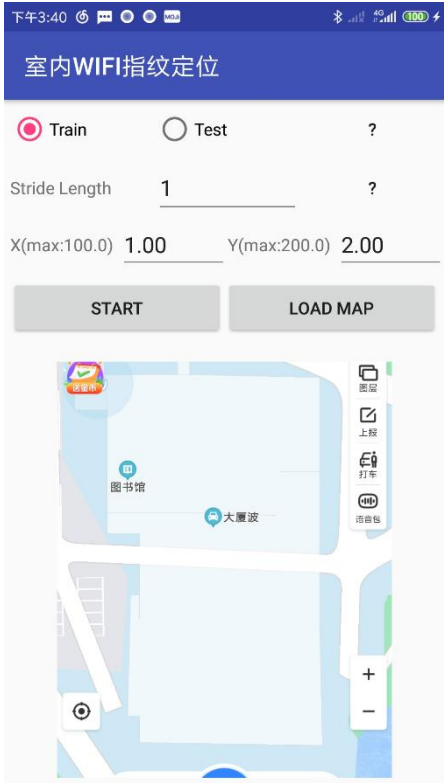
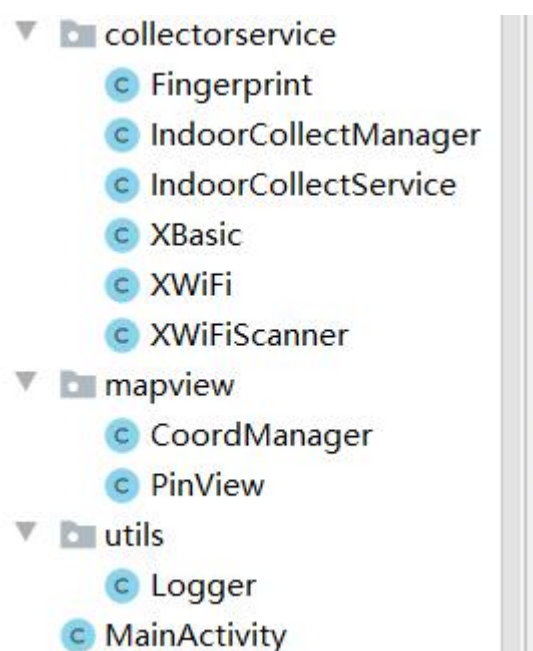


图 2 手动输入每次的位置

程序包结构如图所示：

主要采用后台 Service 的形式在另一个线程中对 WIFI 进行扫描。MainActivity 是启动界面，Logger 类负责将获取到的数据写入文件，mapview 包下的两个类负责将每次定位输入的位置解析并渲染到地图上，XWiFiScanner 是执行扫描工作的工具类，在这个类中调用了 Android 提供的 WifiManger 对象来获取扫描结果，采用 Handler-Listener 模型。IndoorCollectService 是 Service 的子类，会被注册为后台服务。Fingerprint 则是指纹信息的包装类，会在 Logger 中解析为字符串写入到文件。



采集到的数据样式如下：

```
10.00 5.00 14 2020-12-10-14:10:15
76:44:a4:a4:4a:02 -48 w | -48
78:44:fd:69:8b:bc -75 w | -75
3c:e5:a6:94:18:a0 -67 w | -67
3c:e5:a6:94:18:b0 -45 w | -45
88:10:8f:72:76:43 -81 w | -81
5a:7a:6a:a1:4e:d5 -61 w | -61
3c:e5:a6:94:2a:f0 -86 w | -86
78:44:fd:69:86:c6 -71 w | -71
fa:59:71:05:b2:bb -68 w | -68
3c:e5:a6:94:2b:a0 -82 w | -82
3c:e5:a6:94:2b:b0 -71 w | -71
4c:d1:a1:f7:32:b4 -66 w | -66
06:69:6c:4b:23:ee -91 w | -91
78:44:fd:69:86:86 -76 w | -76
```



## 2. 第二阶段-利用 KNN 算法具体估算坐标

根据 KNN 算法，我们需要先建立指纹数据库，也就是每一个坐标点的实测 AP 集的信号强度。建立 RadioMapItem 对象如下：

```
public class RadioMapItem {  
    private double x;  
    private double y;  
    private Map<String,Integer> mac2rssi;  
    // getter setter 省略  
}
```

（备注：x、y 是指纹坐标，mac2rssi 是 mac 地址到信号强度的映射集合。）

然后需要读取上面写入的 txt 文本文件，将数据转化为 java 对象。建立 TrainReader 类。在构造函数中读取存储空间中的数据文件。

```
trainScanner = new Scanner(new File(TARIN_PATH));  
testScanner = new Scanner(new File(TEST_PATH));
```

在后续的计算过程中，需要一个 List<RadioMapItem>集合，这个集合可以通过调用 getTrainRadioMap 方法来获取。按照上图中的数据格式，每个坐标的 AP 集之间由一个空行分割，在 getTrainRadioMap 中我们按空行分割训练集文件，对每一段数据调用 getRadioMapItem 方法来获取一个 RadioMapItem 对象。将对象组织成集合后返回。

getRadioMapItem 方法按照数据格式解析出 xy 坐标以及 mac 和信号强度，构造成 RadioMapItem 返回。

Caculator 类通过上述的 getTrainRadioMap 方法获取到 AP 集合。

```
private List<RadioMapItem> trainList = new TrainReader().getTrainRadioMap();
```

calculateLocation 方法先通过：new TrainReader().getLatestTestItem() 读取出 test 文件中最后一个点的数据。之后建立距离集合 distance，然后对指纹库中每一个 AP 的信号集进行循环，计算测试 AP 集和指纹 AP 的距离，计算后存入距离集合 distance。

然后对 distance 排序，取出最小的 K 个距离值，找到这 K 个距离对应的指纹 AP 的坐标，取平均值即为预估的定位结果。

## 四、实验结果展示

小组在科技楼四楼的实验室，取了 10\*10 范围内，52 个坐标的信号值之后进行定位测试：

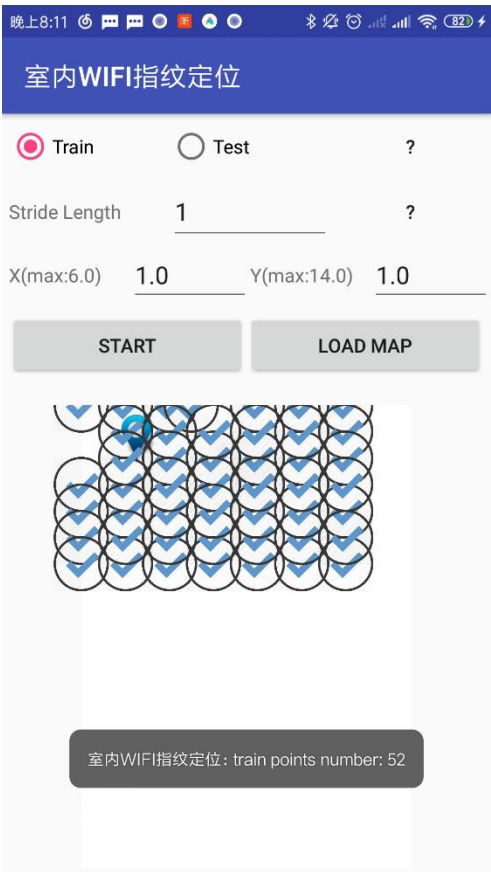


图 3.进行数据收集，建立指纹库

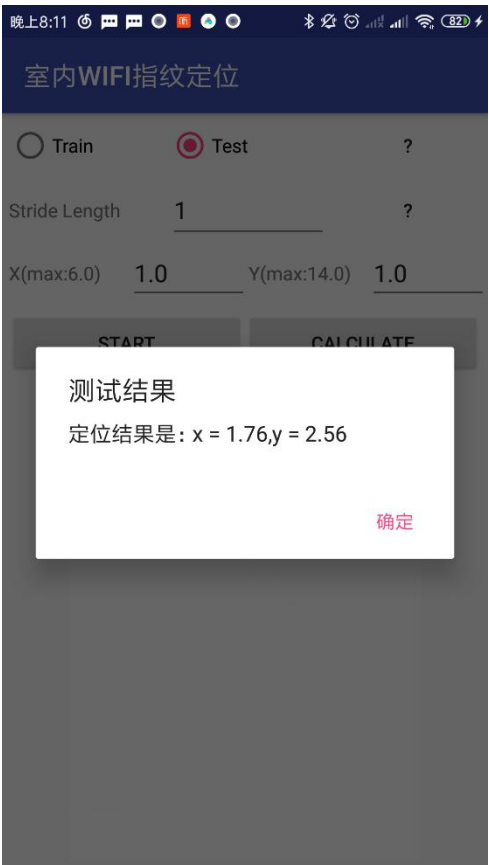
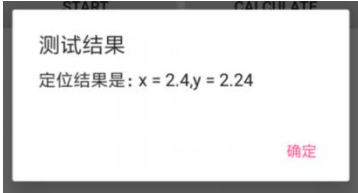
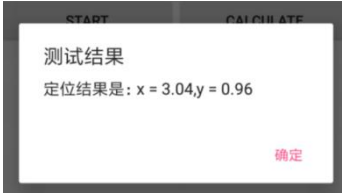
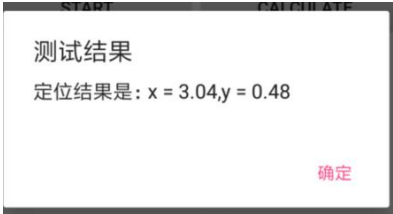
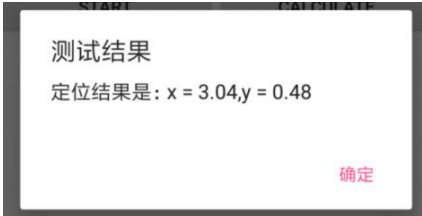
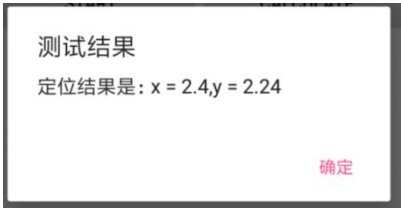
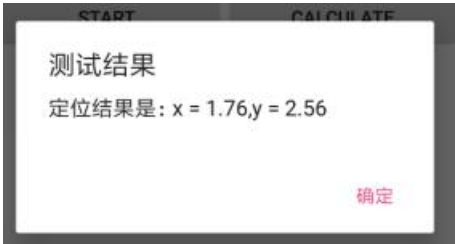


图 4.选取某一点进行定位测试

APP 编译后，取  $K=5$ ，然后实验了 6 个测试点的数据，如上右图所示，6 个点的真实坐标和预估坐标分别为下图所示

- (1)  真实坐标  $x=4.4, y=2.0$
- (2)  真实坐标  $x=3.6, y=2.0$
- (3)  真实坐标  $x=2.8, y=2.0$
- (4)  真实坐标  $x=2.8, y=1.2$
- (5)  真实坐标  $x=1.2, y=1.2$
- (6)  真实坐标  $x=2.0, y=2.5$

## 五、本次实验难点以及解决方案

遇到的第一个困难是，Android 在 6.0 版本之后限制了开发者使用 WLAN 功能的权限，不仅需要 APP 第一次打开时静态授权，还需要每一次使用相关功能的时候再由用户进行一次动态授权。另外，Google 还要求，获取 WIFI 信息的时候还需要用户开启 GPS。

```
minSdkVersion 21
targetSdkVersion 22
versionCode 1
versionName "1.0"
```

经研究，解决办法有两个：降低编译版本或者添加动态权限。在 build.gradle 中降低 target 版本到 23 以下，就可以绕过 6.0 版本的限制。或者在开始按钮的 ClickListener 中，弹出动态授权，在 onRequestPermissionsResult 回调方法中再调用相关扫描 API。

```
ActivityCompat.requestPermissions(this,
    new String[]{Manifest.permission.ACCESS_COARSE_LOCATION,
        Manifest.permission.ACCESS_FINE_LOCATION,
        Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_PERMISSION_CODE);
```

第二个 BUG 是，在单击第二次 CALCULATE 按钮之后，程序报错退出，显示 IOException。经检查，是因为将 Cacultaor 类完全变成了静态类，导致每次只在初始化的时候读取一次数据文件，当进行第二次位置测试时，新写入 test.txt 的文本并未被读取，导致调用 Scanner.next 的时候，遇到文件结尾，解析错误而退出。

解决办法就是将 Calcutor 类去掉静态标签，每次调用时都新建一个对象，也就会重新读取一次数据文件。

## 六、实验总结

从六个测试点的结果可以看出，KNN 算法进行室内定位的准确性较差，在 10 米的范围内，有 2 米左右的误差。实验结果难以称为理想。其中，指纹数据库坐标数量较少、数据地理范围小导致信号强度变化不大、依靠手机收集数据时人体不规范的遮挡等因素都造成了定位不准确。另外，经查找资料，KNN 算法作为较为简单的定位算法，其准确性本来也不高。

通过本次实验，我们小组体会了室内定位技术的复杂性和困难性，熟悉了 KNN 这种定位算法，也了解了 android 开发的基本流程，对其 wifi API 有了初步的认识。通过阅读原始论文，体会了在室内定位这种相对前沿的领域有所发明创造的不易，在查找资料的过程中提升了搜集信息的能力。另外，小组成员一起完成本次实验，也提高了我们团队合作的意识。总的来看，本次实验收获颇丰。

## 七、核心源代码展示（数据收集模块省略）

RadioMapItem.java:

```
import java.util.Map;
public class RadioMapItem {
    private double x;
    private double y;
    private Map<String,Integer> mac2rssi;
    public RadioMapItem(double x, double y, Map<String, Integer> mac2rssi) {
        this.x = x;
        this.y = y;
        this.mac2rssi = mac2rssi;
    }
    public double getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public double getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
    public Map<String, Integer> getMac2rssi() {
        return mac2rssi;
    }
    public void setMac2rssi(Map<String, Integer> mac2rssi) {
        this.mac2rssi = mac2rssi;
    }
    @Override
    public String toString() {
        return "RadioMapItem{" +
            "x=" + x +
            ", y=" + y +
            ", mac2rssi=" + mac2rssi +
            '}';
    }
}
```

TrainReader.java:

```
package com.example.lesliexong.opencollector.calc;

import com.example.lesliexong.opencollector.utils.Logger;
```

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class TrainReader {

    private String TARIN_PATH =
Logger.getRootDir()+"/train"+Logger.getDateUnderLine()+".txt";
    private String TEST_PATH =
Logger.getRootDir()+"/test"+Logger.getDateUnderLine()+".txt";

    private Scanner trainScanner;
    private Scanner testScanner;

    private List<RadioMapItem> trainRadioMap;

    public TrainReader() {
        try {
            trainScanner = new Scanner(new File(TARIN_PATH));
            testScanner = new Scanner(new File(TEST_PATH));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    public RadioMapItem getRadioMapItem(String data){
        // 接受一次扫描结果，返回解析对象
        String[] split = data.split("\n");
        String[] meta = split[0].split(" ");
        double x = Double.parseDouble(meta[0]);
        double y = Double.parseDouble(meta[1]);
        int apNum = Integer.parseInt(meta[2]);

        Map<String,Integer> mac2rssi = new HashMap<>();
        for (int i=0;i<apNum;i++){
            String mac = split[i+1].split(" ")[0];
            Integer rssi = Integer.parseInt(split[i+1].split(" ")[1]);
            mac2rssi.put(mac,rssi);
        }
        if (apNum!=0)
            return new RadioMapItem(x,y,mac2rssi);
        return null;
    }
}

```

```

    }

    public List<RadioMapItem> getTrainRadioMap(){

        List<RadioMapItem> result = new ArrayList<>();
        RadioMapItem temp;

        trainScanner.useDelimiter("\n\r\n");

        RadioMapItem each;

        while(trainScanner.hasNext()){
            each = getRadioMapItem(trainScanner.next());
            if (each!=null)
                result.add(each);
        }
        return result;
    }

    public RadioMapItem getLatestTestItem() throws InterruptedException {
        testScanner.useDelimiter("\n\r\n");
        String latestData = "";
        while (testScanner.hasNext()){
            latestData = testScanner.next();
        }
        return getRadioMapItem(latestData);
    }

}

Calculator.java:
import android.os.Build;
import android.support.annotation.RequiresApi;

import java.util.*;

/**
 * @author 7zq12lvm
 * @create 2020-12-15 9:07
 */

```

```

public class Caculator {

    private List<RadioMapItem> trainList = new
TrainReader().getTrainRadioMap();
    private int K = 5;

    @RequiresApi(api = Build.VERSION_CODES.N)
    public double[] calculateLocation() throws InterruptedException {

        // 得到测试点信息
        RadioMapItem testItem = new TrainReader().getLatestTestItem();

        // 测试点的AP 信号集合
        Map<String,Integer> testMac2rssi = testItem.getMac2rssi();

        ArrayList<Integer> distance = new ArrayList<>(trainList.size());

        for (int i=0;i<trainList.size();i++){
            // 对于radiomap 中的每一个热点, 计算test 与它的距离
            RadioMapItem ap = trainList.get(i);
            Map<String,Integer> apMac2rssi = ap.getMac2rssi();
            int distanceCounter = testMac2rssi.keySet().stream()
                .filter(s -> apMac2rssi.get(s) != null)
                .mapToInt(s -> Math.abs(apMac2rssi.get(s) -
testMac2rssi.get(s)))
                .sum();
            distance.add(i,distanceCounter);
        }

        ArrayList<Integer> dCopy = (ArrayList<Integer>) distance.clone();
        distance.sort(Comparator.comparing(Integer::intValue));

        double x = 0.0,y = 0.0;
        for (int i=0;i<K;i++){
            // Arrays.stream(distanceUnsorted).findFirst();
            x += trainList.get(dCopy.indexOf(distance.get(i))).getX();
            y += trainList.get(dCopy.indexOf(distance.get(i))).getY();
        }

        return new double[]{x/K,y/K};
    }
}

```



```
}

@RequiresApi(api = Build.VERSION_CODES.N)
public void main(String[] args) throws InterruptedException {
    double[] location = calculateLocation();
    System.out.println("x="+location[0]+",y="+location[1]);
}
}
```

# 实验三：无线室内定位系统

## 一、实验目标及环境

了解典型的无线传感器网络定位算法

在所给的网络中实现所讲授的无线传感器网络定位算法并进行比较

## 二、实验要求

- 1.（课堂完成）掌握典型的无线传感器网络定位算法基本原理，理解所讲的迭代式多边定位算法、DV-HOP 算法、PDM 定位算法、基于 MDS 的定位算法；
- 2.（课后完成）利用所给的网络数据，实现两种以上的定位算法并进行比较
- 3.（撰写实验报告）每个小组提交一份实验报告，包括算法原理，算法实现以及实验结果比较。
- 4.（实验结果展示）每个小组选派一名同学汇报自己小组的实验过程和结果，准备大概 5 分钟的 ppt。

## 三、实验资料介绍

### 1.节点位置数据

文件 net1\_pos 中给出了实验网络中节点的位置数据。每行表示一个节点的位置信息。格式如下：

节点序号 节点 x 坐标 节点 y 坐标 是否锚节点  
(1 代表锚节点，0 代表待定位节点)

```
net1_pos.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1 17.8977 106.2282 1
2 88.4522 43.1223 1
3 116.7474 169.6965 1
4 25.6446 96.3504 1
5 58.7169 157.5082 1
6 129.9749 50.8492 1
7 158.1535 34.7733 1
8 179.2871 30.6946 1
9 197.4612 60.6438 1
10 158.5115 90.1093 1
11 196.7114 32.8254 1
12 79.7605 32.5048 1
13 32.0705 30.7106 1
14 164.4605 107.9358 1
15 104.7398 74.5080 1
16 114.9813 162.7896 1
17 131.4193 62.3286 1
18 176.9411 105.3225 1
19 39.7566 154.4565 1
20 143.6027 117.2557 1
21 49.3941 89.9023 1
22 168.3337 119.6875 1
23 41.7215 168.8211 1
```

如 1 17.8977 106.2282 1：

表示节点 1，其真实位置为（17.8977,106,2282），该节点是锚节点

## 2. 节点之间的测距信息

文件 `net1_topo_error_free` 中给出了网络中相邻节点之间的距离信息。每一行表示两个节点之间的距离。格式如下：

节点 1 序号 节点 2 序号 节点之间距离测量值



```
net1_topo-error free.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1 4 12.5534
1 24 19.5705
1 42 15.7572
1 93 14.6704
1 129 5.6604
1 225 19.7107
1 269 13.0936
1 271 17.5454
1 305 18.0892
2 12 13.7214
2 99 6.8700
2 144 15.3077
2 164 13.6139
2 203 16.5905
```

如 1 4 12.5534:

表示节点 1 和节点 4 可以相互测量出彼此的距离,他们之间的距离是 12.5534

## 3. 文件夹中给出的 txt 文本说明

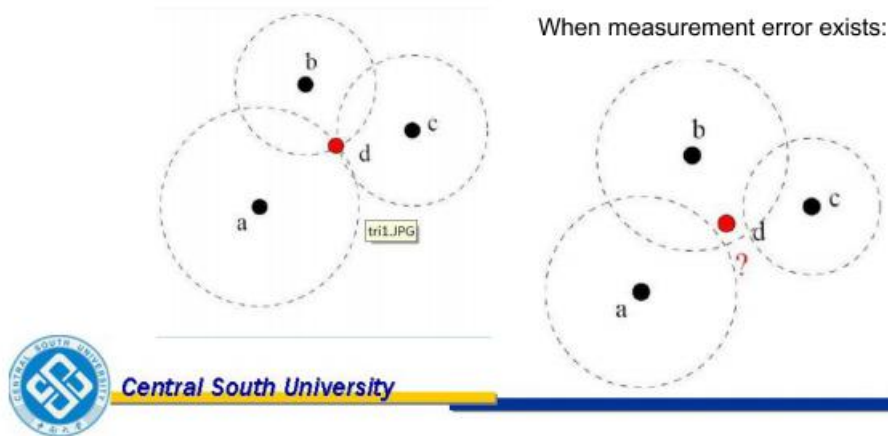
文件 `net1_topo_error_5` 给出的距离增加了 5%的误差之后的扰动值

文件 `net1_topo_error_10` 给出的距离增加了 10%的误差之后的扰动值

# 四、算法介绍

## 1.迭代多边定位算法

**1.1 算法思想:** 利用已知信标节点和非信标节点的几组对应关系(两点间距离)对非信标节点进行定位。(某个非信标节点只要已知 3 个及以上信标节点与其距离即可对其进行定位。)定位成功后的非信标节点转换为信标节点,可辅助对其他非信标节点定位。不断迭代定位过程,知道信标节点集合元素个数在前后两次迭代中数量不变,则迭代终止。



## 多边定位

$$\begin{cases} 2(x_1 - x_2)x + 2(y_1 - y_2)y = d_2^2 - x_2^2 - y_2^2 - (d_1^2 - x_1^2 - y_1^2) \\ \vdots \\ 2(x_1 - x_m)x + 2(y_1 - y_m)y = d_m^2 - x_m^2 - y_m^2 - (d_1^2 - x_1^2 - y_1^2) \end{cases}$$

$$A = \begin{bmatrix} 2(x_1 - x_2) & 2(y_1 - y_2) \\ \vdots & \vdots \\ 2(x_1 - x_m) & 2(y_1 - y_m) \end{bmatrix}, \mathbf{b} = \begin{bmatrix} d_2^2 - x_2^2 - y_2^2 - (d_1^2 - x_1^2 - y_1^2) \\ \vdots \\ d_m^2 - x_m^2 - y_m^2 - (d_1^2 - x_1^2 - y_1^2) \end{bmatrix}$$

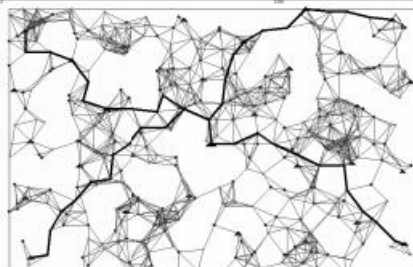
$$A \mathbf{x} = \mathbf{b} \longrightarrow \mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$$

**1.2 前提条件:** 已知位置的锚节点个数要超过 3 个; 对所有节点来说, 在添加完邻接矩阵关系后, 是一个连通图, 没有孤立节点。

**1.3 特别说明:** 因为在使用迭代多边定位算法对未知节点进行定位时需要利用该节点与 3 个以上信标节点的直线距离, 图是连通图但不是全连通图, 所以可能会 有部分节点无法定位

## 2.DV-HOP 算法

**2.1 算法思想：**同样使用待求节点到其他节点的距离来求待求节点位置，只是求距离的方法改变，改为由跳数乘以平均每跳距离来估算。跳数可以用 Floyd 算法求出。

$$\frac{\sum \text{Inter-Anchor Distances}}{\sum \text{Inter-Anchor Hop Counts}}$$


**2.2 前提条件：**已知位置的锚节点个数要超过 3 个；对所有节点来说，在添加完邻接矩阵关系后，是一个连通图，没有孤立节点。

**2.3 特别说明：**图是否为连通图对于迭代多边算法影响可能不会很大，但是在 DV-HOP 算法中，如果有孤立节点，则会导致程序出错。

## 3.PDM 算法

**3.1 算法思想：**PDM 算法全称为计算邻近度-距离转换矩阵算法 (Proximity-to-Distance Mapping)，也就是说通过计算节点间的邻近度-距离转换矩阵来给未知节点进行定位，本算法中涉及到：

$T$ ：距离转换矩阵

$\vec{p}$ ：各个点到锚节点的跳数矩阵

$\vec{d}$ ：各个点到锚节点的距离矩阵

$$\vec{d} = T \cdot \vec{p}$$

通过锚节点间协作构建邻近度-距离转换矩阵  $D=P \cdot T$ ，对于已知的锚节点，它们之间的跳数可以通过 Floyd 算法计算出来，即可获取关于所有锚节点的跳数矩阵  $P$ ，而锚节点的坐标已知，其相互之间的距离可以计算出来，即构造了距离矩阵  $D$ 。利用伪逆技术增强鲁棒性，计算出转换矩阵  $T$ ：

$$T = D \cdot P^T (PP^T)^{-1}$$

若锚节点的分布要能较准确的刻画网络拓扑性质，我们可以认为矩阵  $T$  也符合网络中的非锚节点。而非锚节点到锚节点的跳数可以用 Floyd 算出，即  $P$  可算出，通过  $D=T \cdot P$  可以算出某个点到锚节点的距离，再调用算法一可以算出坐标。

总结来说使通过计算节点间的邻近度-距离转换矩阵（也就是跳数-距离转换矩阵）来给未知节点进行定位。其中转换矩阵由锚节点集合求出，应用在普通节点上以求出锚节点到普通节点间距离，再根据距离求出待求节点位置。其中跳数（最小跳数）可由 Flyod 算法求出，距离采用欧式距离。

**3.2 前提条件：**已知位置的锚节点个数要超过 3 个；对所有节点来说，在添加完邻接矩阵关系后，是一个连通图，没有孤立节点

**3.3 特别说明：**图是否为连通图对于 PDM 算法影响很大，如果有孤立节点，矩阵运算就不会正确，无法定位。

## 4.MDS 算法

**4.1 算法思想：**对于  $n$  个节点间的距离矩阵（size:  $n \times n$ ），使用 MDS 降维置 2 维平面上，成为相对距离，转化为绝对坐标即可实现定位。计算所考虑区域中所有节点对之间的最短路径。

最短路径距离用于构造 MDS 的距离矩阵。将经典 MDS 应用于距离矩阵，保留前 2 个（或 3 个）最大特征值和特征向量，以构建 2-D（或 3-D）相对图。给定足够的锚点节点（2-D 为 3 个或更多，3-D 为 4 个或更多），根据锚点的绝对位置将相对图转换为绝对图。

**4.2 前提条件：**已知位置的锚节点个数要超过 3 个；对所有节点来说，在添加完邻接矩阵关系后，是一个连通图，没有孤立节点。

**4.3 特别说明：**MDS-MAP 的当前实现的缺点是，它需要网络的全局信息和集中式计算。解决此问题的一种方法是将网络划分为多个子网络，然后将 MDSMAP 分别应用于每个子网络。另一个缺点是，当锚节点的数量很大时，MDS-MAP 的性能不如以前的方法。

## 五、算法实现

### 1. 迭代多边定位算法

**第一步：**将数据读入内存。利用 Matlab 中已有的函数 load 将文件中的内容以矩阵的方式读入内存。

```
Data_post = load('net1_pos.txt');
Data_road1 = load('net1_topo-error free.txt');
Data_road2 = load('net1_topo-error 5.txt');
Data_road3 = load('net1_topo-error 10.txt');
culunm_post = size(Data_post);
```

**第二步：**判断锚节点的个数。

```
tempcount = 0;
for i = 1:culunm_post(1)
    if Data_post(i,4) == 1 tempcount = tempcount+1;
end
```

```

end
if tempcount < 3
disp('锚节点少于 3 个,迭代多边定位算法无法执行');
return;
end

```

**第三步：**初始化距离矩阵，将与锚节点有关的路径读入距离矩阵  
先将距离矩阵全部赋值为无穷大，再将每个节点到自身的距离规定为 0，  
在边的信息中，如果有端点为锚节点的，就更新其两点间的距离。

```


for i = 1:culunm_road(1)
if(Data_road(i,1) <= anchors_n)
matrix(Data_road(i,1),Data_road(i,2)) = Data_road(i,3);
elseif(Data_road(i,2) <= anchors_n)
matrix(Data_road(i,2),Data_road(i,1)) = Data_road(i,3);
end
end

```

**第四步：**迭代计算

逐一判断每一个非信标节点是否有 3 个以上锚节点与其相关，如果有 3 个以上的信标节点与其相关则根据：

$$\begin{cases} (x-x_1)^2 + (y-y_1)^2 = d_1^2 \\ (x-x_2)^2 + (y-y_2)^2 = d_2^2 \\ \vdots \\ (x-x_m)^2 + (y-y_m)^2 = d_m^2 \end{cases}$$

$$\mathbf{A} = \begin{bmatrix} 2(x_1-x_2) & 2(y_1-y_2) \\ \vdots & \vdots \\ 2(x_1-x_m) & 2(y_1-y_m) \end{bmatrix}, \mathbf{b} = \begin{bmatrix} d_2^2 - x_2^2 - y_2^2 - (d_1^2 - x_1^2 - y_1^2) \\ \vdots \\ d_m^2 - x_m^2 - y_m^2 - (d_1^2 - x_1^2 - y_1^2) \end{bmatrix}$$


$$\begin{bmatrix} x \\ y \end{bmatrix}$$

其中  $x, y$  为未定位的节点的实际位置， $(x_1, y_1), (x_2, y_2), \dots$  是已知的锚节点到该未知节点的距离。

根据以上两个矩阵的转换，可以求得：

，其中  $\mathbf{x}$  为一个一

行两列的矩阵，分别表示未知节点定位后的坐标位置  $x, y$ 。定位成功后，把该点的标号加入锚节点集合中并从非锚节点集合中删除，并在距离矩阵中加入与该节点相关的边的信息。

**不断迭代运行第四步，直至锚节点集合中元素个数不再变化。**

核心代码如下：

(1) 计算未知节点位置

```

point = temp(1,3)^2 - temp(1,1)^2 - temp(1,2)^2;
for ii = 2:k-1
A(ii-1,:) = 2*[temp(1,1) - temp(ii,1) temp(1,2) - temp(ii,2)];
b(ii-1,:) = [temp(ii,3)^2 - temp(ii,1)^2 - temp(ii,2)^2 - point];

```

```

end
Ans = inv(transpose(A)*A)*transpose(A)*b;
estimated(j,1) = Ans(1,1);
estimated(j,2) = Ans(2,1);
(2) 更新锚节点、非锚节点集合以及距离矩阵
anchors_n_t = anchors_n_t + 1;
anchors(anchors_n_t) = j;
Locate=find(all == j);
all(Locate) = [];
for m = 1:culunm_road(1)
if(Data_road(m,1) == j)
matrix(Data_road(m,1),Data_road(m,2)) = Data_road(m,3);
elseif(Data_road(m,2) == j)
matrix(Data_road(m,2),Data_road(m,1)) = Data_road(m,3);
end
End

```

**第五步：**计算误差，输出结果

**重点部分代码说明：**

根据待求节点到（已知位置的）锚节点间距离求待求节点位置：

```

A=[2 * ( x(1,1) - x(2,1) ), 2 * ( y(1,1) - y(2,1) );
2 * ( x(1,1) - x(3,1) ), 2 * ( y(1,1) - y(3,1) )];

b=[ ( d(2,1)^2 - x(2,1)^2 - y(2,1)^2 ) - ( d(1,1)^2 - x(1,1)^2 -
y(1,1)^2 );
( d(3,1)^2 - x(3,1)^2 - y(3,1)^2 ) - ( d(1,1)^2 - x(1,1)^2 -
y(1,1)^2 )];

Z=inv(A'*A)*A'*b;

Loc(i,1)=Z(1,1);

Loc(i,2)=Z(2,1);

```

这里是已知三个距离的情况，可以理解为三个圆形交叉求交点。

*for j=1:anchor\_n-1* %锚节点遍历

```

A(j,1)=2*(anchor(2,j)-anchor(2,anchor_n)); % 2(xj-xn)

```



```

A(j,2)=2*(anchor(3,j)-anchor(3,anchor_n)); % 2(yj-yn)

%bj = xj^2-xn^2+yj^2-yn^2+dn^2-dj^2

%dn^2 - dj^2 = (hop(i,n)*hopdis(i))^2-(hop(i,j)*hopdis(i))^2

b(j)=anchor(2,j)^2 - anchor(2,anchor_n)^2 + anchor(3,j)^2 ...
-anchor(3,anchor_n)^2 +
hopdis(i)^2*hop(i,anchor_n)^2-hopdis(i)^2*hop(i,j)^2;

end

%X=(A'*A)^-1*A'*b

X=inv(A'*A)*A'*b;

esxy(1,i)=X(1);

esxy(2,i)=X(2);

```

这里则是  $n$  个距离的情况，可以理解为超球求交点。  
将定位成功的待求节点转化为锚节点

```
NodePos(i,4)=1;
```

只要将标记位置 1 则可

## 2. DV-HOP 算法

**第一步：**将数据读入内存。利用 Matlab 中已有的函数 load 将文件中的内容以矩阵的方式读入内存。

**第二步：**判断锚节点的个数。

**第三步：**将所有的两点间的距离关系读入距离矩阵

```
for i =1:culunm_road(1)
matrix(Data_road(i,1),Data_road(i,2)) = Data_road(i,3);
matrix(Data_road(i,2),Data_road(i,1)) = Data_road(i,3);
end
shortest_path = matrix;
```

**第四步：**利用最短路径算法求得两点间的最短路径

```
for k=1:nodes_n
for i=1:nodes_n
for j=1:nodes_n
if
shortest_path(i,k)+shortest_path(k,j)<shortest_path(i,j)
shortest_path(i,j) =
shortest_path(i,k)+shortest_path(k,j);
jump_n(i,j) = jump_n(k,j);
end
end
end
End
```

**第五步：**求每个信标节点的校正值

利用函数：

```
anchor_to_anchor=shortest_path(1:anchors_n,1:anchors_n);
for i=1:anchors_n
hopsiz(i)=sum(sqrt(sum(transpose(( repmat(true(i,:),anchors_n,1)...
-
true(1:anchors_n,:)).^2))))/sum(anchor_to_anchor(i,:));
End
```

**第六步：**未知节点计算位置

先通过距离=跳数\*校正值得未知节点到每个锚节点的距离，再根据最小二乘法计算具体位置。

```
obtained_hopsiz=hopsiz(find(shortest_path(i,1:anchors_n)==...
min(shortest_path(i,1:anchors_n))));
unknown_to_anchors_dist=transpose(obtained_hopsiz(1)*...
shortest_path(i,1:anchors_n));
A=2*(estimated(1:anchors_n-1,:)-repmat(estimated(anchors_n,:),...
anchors_n-1,1));
anchors_location_square=transpose(sum(transpose...
```

```

    (estimated(1:anchors_n,:).^2));
    dist_square=unknown_to_anchors_dist.^2;
    b=anchors_location_square(1:anchors_n-1)-...
    anchors_location_square(anchors_n)-dist_square(1:anchors_n-1)+..
    dist_square(anchors_n);
    estimated(i,:)=transpose(A\b);

```

**第七步：** 计算误差，输出结果

**重点部分代码说明：**

Floyd 算法求跳数矩阵 hop

```

for k=1:node_n(1)

    for i=1:node_n(1)

        for j=1:node_n(1)

            if hop(i,k)+hop(k,j)<hop(i,j)

                hop(i,j)=hop(i,k)+hop(k,j);

            end

        end

    end

end

```

平均每跳距离 hopsize

```

for i=1:anchor_n

    distance = 0;

    h = 0;

    for j=1:anchor_n

        if (i~=j)

            distance=distance+dis(i,j);

            h=h+hop(i,j);

        end

    end

```

end

hopsizel(i)=distance/h;

End

全局平均每跳距离 hopAvg

hopAvg = 0;

for i=1:anchor\_n

hopAvg = hopAvg + hopsizel(i);

end

hopAvg = hopAvg/anchor\_n;

挑选最近锚节点的平均每跳距离 hopdis

hopdis=zeros(1,node\_n(1));

for i =anchor\_n+1:node\_n(1)%

min=inf;

index=0;

for j=1:anchor\_n%

if dis(i,j)<min)%获得普通节点于锚节点间最小距离

min=dis(i,j);

index=j;

end

end

if index ==0%若某普通节点没有到锚节点的直接连接

hopdis(i)=hopAvg;%则其平均每跳距离为全局平均每跳距离

else

```
hopdis(i)=hopsizex(index);%
```

```
end
```

```
End
```

### 3.PDM 算法

**第一步：**将数据读入内存。利用 Matlab 中已有的函数 load 将文件中的内容以矩阵的方式读入内存。

**第二步：**判断锚节点的个数。

**第三步：**将所有的两点间的距离关系读入距离矩阵

**第四步：**利用最短路径算法求得两点间的最短路径，以及最短路径下每对节点的上一跳信息。

**第五步：**构造节点跳数矩阵 P\_all

根据上一跳矩阵 jump\_n，通过循环判断可以得出一个 32\*320 的矩阵 P\_all，P\_all(i,j)表示从第 i 个点到第 j 个点最少需要多少跳。

```
for i = 1:nodes_n
for j = 1:nodes_n
temp_num = last_jump(i,j);
while(temp_num~=i)
P_all(i,j) = P_all(i,j)+1;
temp_num=last_jump(i,temp_num);
end
end
End
```

**第六步：**构造 PDM 算法所需要的各种矩阵

所有锚节点的跳数矩阵 P\_anchors、锚节点的距离矩阵 D\_anchors、转换矩阵 T、非锚节点到锚节点的跳数 P\_Nanchors、未知节点到锚节点的距离 D\_Nanchors

```
P_anchors = P_all(1:anchors_n,1:anchors_n);
D_anchors = pdist2(true_anchors,true_anchors);
D_anchors = D_anchors(1:anchors_n,1:anchors_n);
T =
D_anchors*transpose(P_anchors)*inv(P_anchors*transpose(P_anchors));
P_Nanchors = P_all(1:anchors_n,33:320);
D_Nanchors = T*P_Nanchors;
```

**第七步：**利用最小二乘法估算具体位置

**第八步：**计算误差，输出结果

**重点部分代码说明：**

Floyd 算法求最小跳数矩阵 Hop

```
for k=1:node_n(1)
```

```
for i=1:node_n(1)
```

```

        for j=1:node_n(1)

            if hop(i,k)+hop(k,j)<hop(i,j)

                Hop(i,j)=hop(i,k)+hop(k,j);

            end

        end

    end

end
end

```

求锚节点间距离矩阵 dis

```

for i=1:anchor_n

    for j=1:anchor_n

        dis(i,j)= ...

        sqrt( (xy(2,i)-xy(2,j))^2+(xy(3,i)-xy(3,j))^2 );

    end

end
end

```

求跳数距离转换矩阵 t\_aa

```

dis_aa=dis(1:anchor_n,1:anchor_n);

hop_aa=hop(1:anchor_n,1:anchor_n);

hop_an=hop(1:anchor_n,anchor_n+1:end);

t_aa=dis_aa*transpose(hop_aa)*inv((hop_aa*transpose(hop_aa)));

```

求锚节点到普通节点距离矩阵 dis\_an

```

dis_an=t_aa*hop_an;

```

#### 4.MDS 算法

求距离矩阵 dis

```
for k=1:all_nodes.nodes_n

    for i=1:all_nodes.nodes_n

        for j=1:all_nodes.nodes_n

            if shortest_path(i,k)+shortest_path(k,j)<...
                shortest_path(i,j)%min(h(i,j),h(i,k)+h(k,j))
                shortest_path(i,j)=
                    ...shortest_path(i,k)+shortest_path(k,j);
            end
        end
    end
end
end
```

将距离矩阵用 MDS 降维：

```
J=eye(all_nodes.nodes_n)-ones(all_nodes.nodes_n)/all_nodes.nodes_n;

H=-0.5*J*shortest_path.^2*J;

[V S T]=svd(H);

X=T*sqrt(S);

relative_map=X(:,1:2);

save maps_and_all_nodes.mat relative_map all_nodes;

求解绝对位置

[q r]=fsolve(

    'relative_to_absolute',

    [1,1,0,0,all_nodes.square_L/2,all_nodes.square_L/2],
```

```
optimset('Display','off','TolX',0.001,'TolFun',0.001,'MaxIter',inf,'Max  
FunEvals',inf)
```

```
);
```

```
absolute_map=relative_map*[q(1)*cos(q(3))
```

```
q(2)*sin(q(4));-q(1)*sin(q(3)) q(2)*cos(q(4))]+ repmat([q(5)
```

```
q(6)],all_nodes.nodes_n,1);
```

```
save maps_and_all_nodes.mat absolute_map -APPEND;
```

```
all_nodes.estimated(all_nodes.anchors_n+1:all_nodes.nodes_n,:)=absolu
```

```
te_map(all_nodes.anchors_n+1:all_nodes.nodes_n,:);
```

其中 'relative\_to\_absolute' 函数为

```
function q=relative_to_absolute(init)
```

```
load maps_and_all_nodes.mat;
```

```
Stretch_x=init(1);
```

```
Stretch_y=init(2);
```

```
rotate_angle_x=init(3);
```

```
rotate_angle_y=init(4);
```

```
vx=init(5);
```

```
vy=init(6);
```

```
difference=relative_map(1:all_nodes.anchors_n,:)*[Stretch_x*cos(rota  
te_angle_x)
```

```
Stretch_y*sin(rotate_angle_y);-Stretch_x*sin(rotate_angle_x)
```

```
Stretch_y*cos(rotate_angle_y)]+ repmat([vx
```



```

vy],all_nodes.anchors_n,1)-all_nodes.estimated(1:all_nodes.anchors_n,
:);

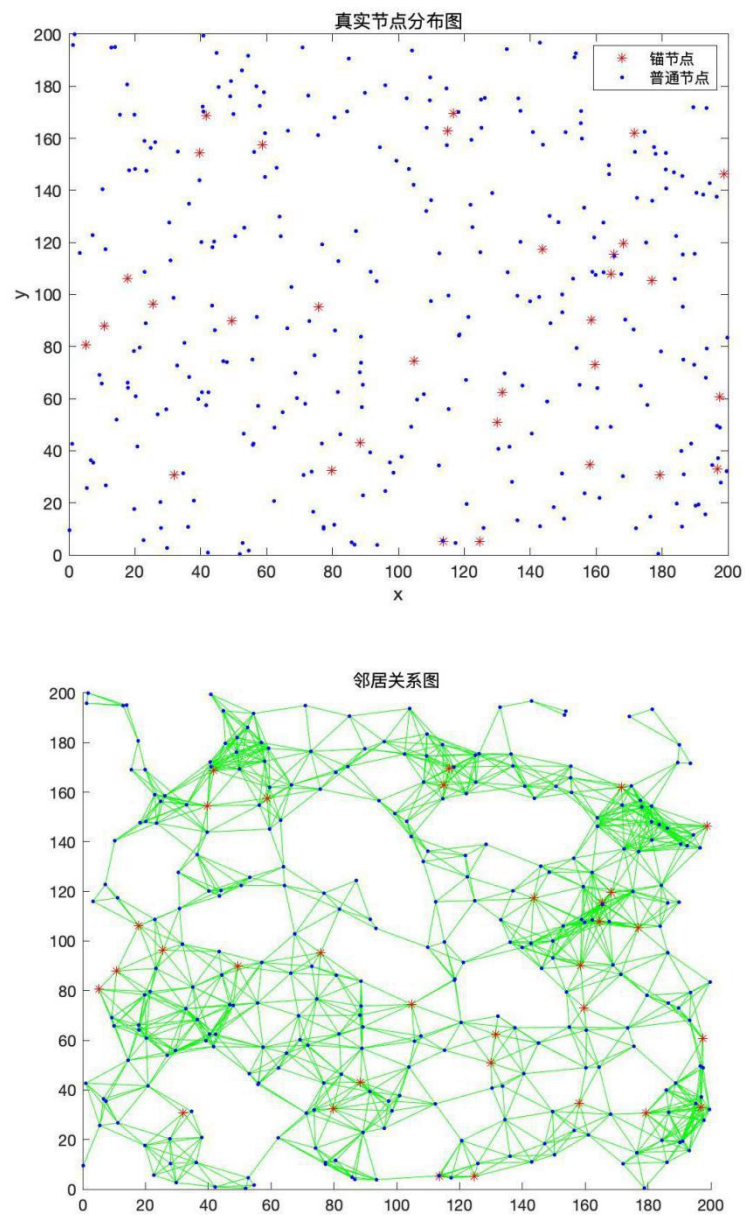
q=transpose(sqrt(sum(transpose(difference.^2))));

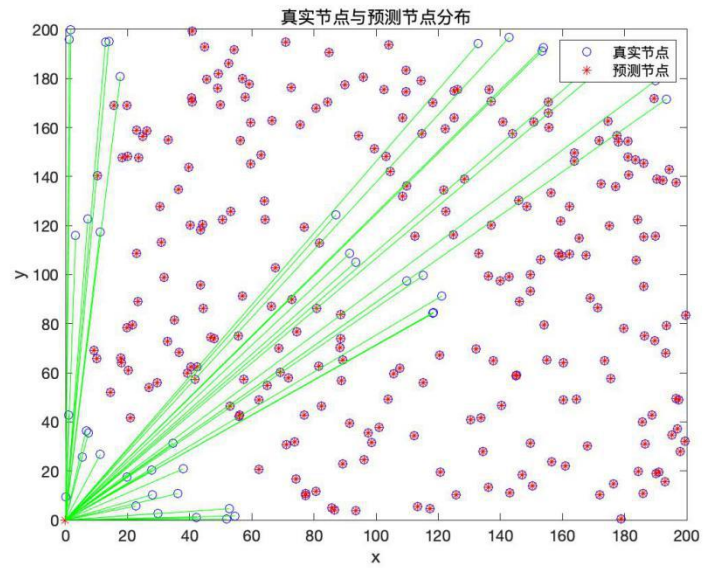
end

```

## 六、实验结果和分析

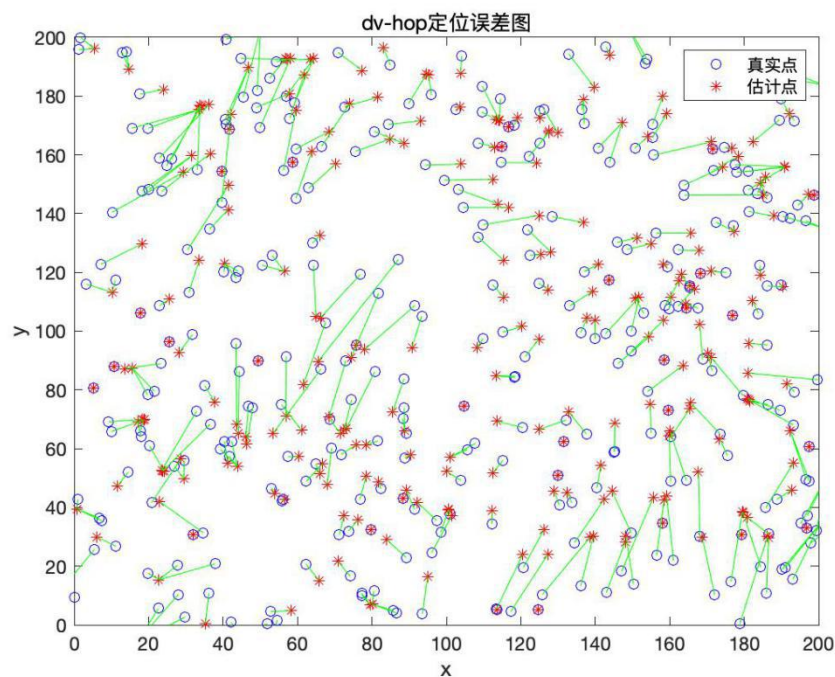
### 1. 迭代多边定位算法





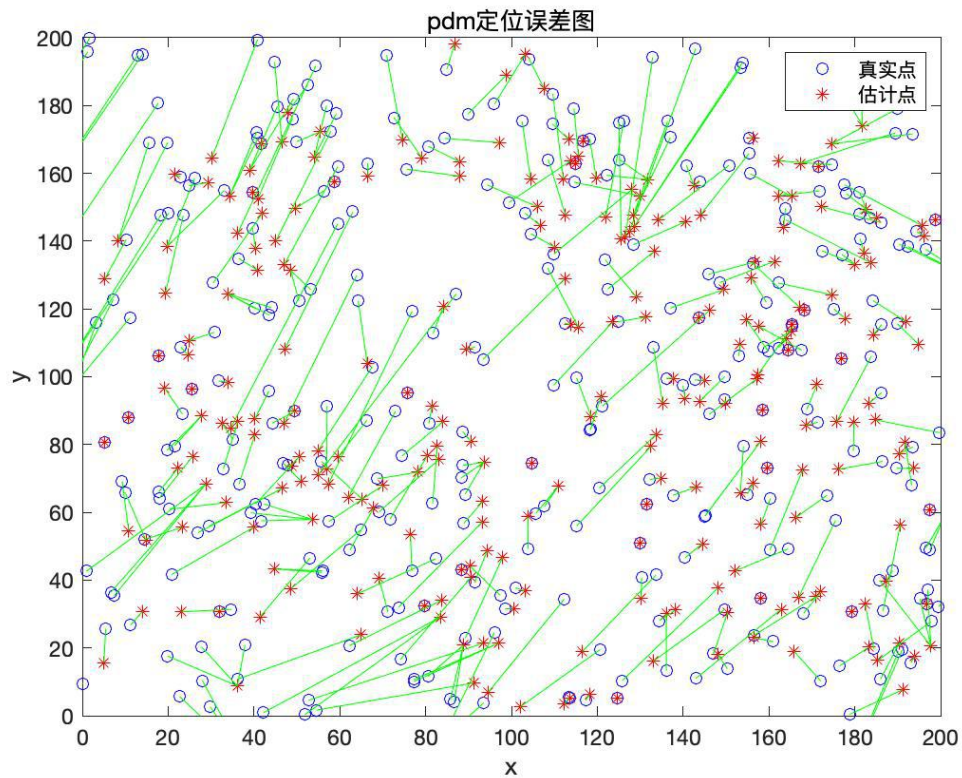
误差分析：会有一些已知距离小于三个的点无法求出位置，而能求出位置的点结果都较为精准。

## 2. DV-HOP 算法



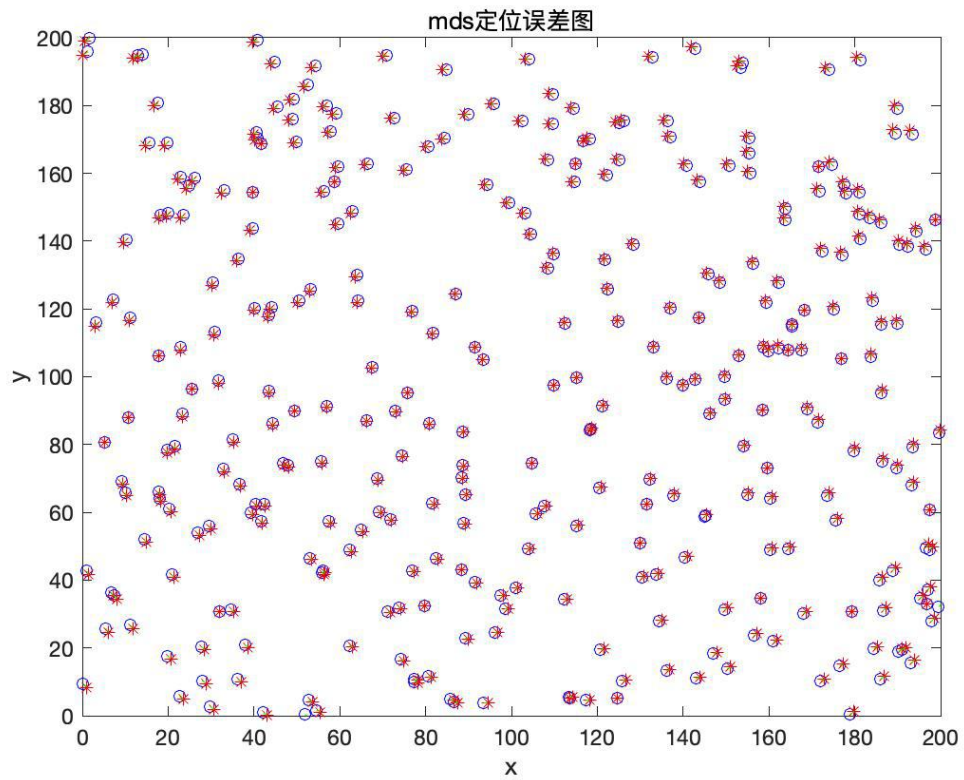
误差分析：由于使用平均每跳距离来估算，所以距离的估算不够准确，引入了误差，常见的平均每跳距离计算方法为取离待求节点最近的锚节点的平均每跳距离计算。

### 3.PDM 算法



误差分析: 由于使用锚节点间的转换矩阵来计算锚节点到普通节点的距离, 因此需要锚节点能刻画整个网络的拓扑结构, 能够反应整个网络跳数于距离的关系。如果锚节点比例较小或者不能刻画网络拓扑结构, 则误差会相当大。

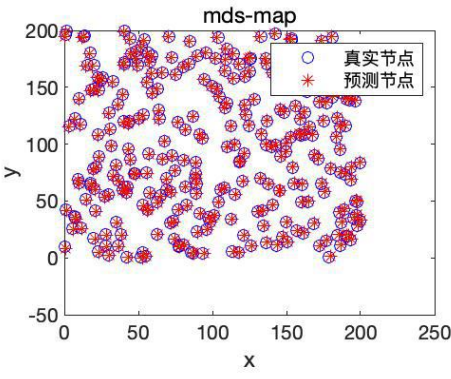
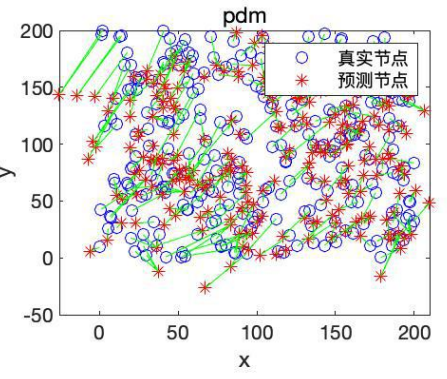
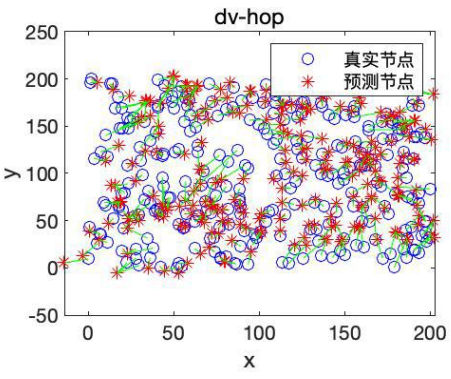
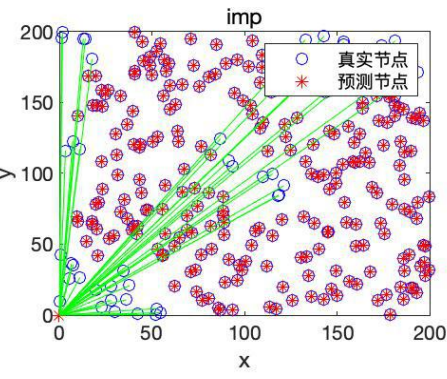
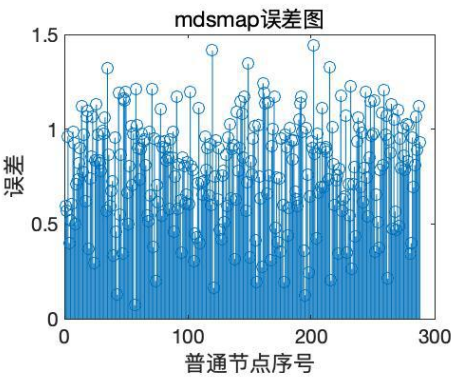
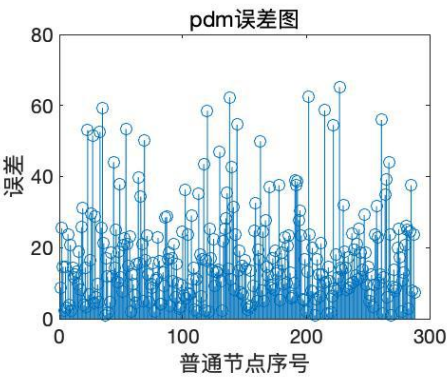
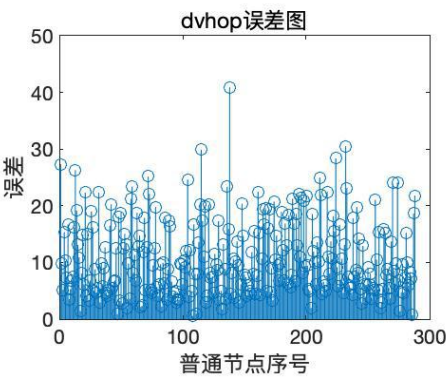
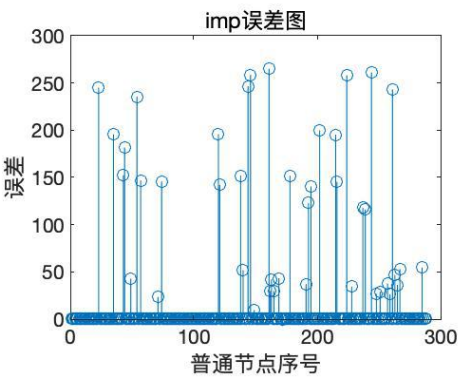
#### 4.MDS 算法



误差分析：MDS 降维会损失一定信息，然而这种损失较小，因此该算法误差也较小。



四种算法对比汇总



## 七、实验总结

在第三次的实验中，我们对于课堂上老师讲授过的几种定位技术进行了实践。通过查阅资料中给定的文献，小组内部讨论思考，分工进行实现。掌握典型的无线传感器网络定位算法基本原理，理解所讲的迭代式多边定位算法、DV-HOP 算法、PDM 定位算法、基于 MDS 的定位算法。

整个过程中，锻炼了我们的文献阅读能力，以及算法实现的能力。并且通过亲身的实践我们对于不同算法的性能有了具体的领会。也提升了与同学协作完成实验的效率。

在本次实验中，我们学习了关于定位技术的相关知识，巩固了课内学习的知识。在实验的过程中，我们曾遇到了不少的困难，也曾在晚上苦苦调试 bug，遇到无数不知道为何出现的 bug，也曾在代码中迷失。但是好在我们都坚持了过来，回看这一路，我们学到了很多。

这次实验中的成功与失败都给了我们丰富的体验，让我们体会到开发的不易。实际应用中的 bug 往往出乎意料，所以我们只有将知识掌握的更加牢固，将能力提升到更高的水平，才能够在实际应用中披荆斩棘，化腐朽为神奇。而成功的体验更是让我们体会到从知识到实践的喜悦，这种将自己所学化为现实的感觉无可替代，给了我们更加充分的自信心。我相信，这次实验对于别人来说是一小步，但对我个人来说是一大步。利用这次实验带给我的体验，我将有更加浓厚的兴趣学习开发定位相关的应用系统展望未来，我们会更加努力学习课程的知识，为未来的职业生涯打下坚实的基础。

如果这次实验我能够完成，那么需要感谢我的老师，张士庚老师，他对我们的教诲如同春风化雨，润物细无声。我们不知不觉就学会了很多关于定位的知识，更了解了许多定位技术实际应用的生动例子。相信经过一学期的学习，我肯定学到了定位的基本要领与精髓，更是能在以后的人生中披荆斩棘，所向披靡。