UNIVERSITATEA POLITEHNICA BUCUREŞTI

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Poziționare în Interior Folosind Amprente Radio

|  |  |
| --- | --- |
| Coordonator ştiinţific:  Conf. Dr. Ing. Dragoș Niculescu | Absolvent:  Adrian Dumitru Nicolau |

BUCUREŞTI

2014

POLITECHNICA UNIVERSITY OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

COMPUTER SCIENCE DEPARTMENT

|  |  |
| --- | --- |
| upb | cs |

DIPLOMA PROJECT

Indoor Positioning System Using Fingerprinting

|  |  |
| --- | --- |
| Thesis supervisor:  Conf. Dr. Ing. Dragoș Niculescu | Adrian Dumitru Nicolau |

BUCHAREST

2014

Notații și abrevieri

**GPS** – **G**lobal **P**ositioning **S**ystem

**SBL** – **S**ervicii **B**azate pe **L**ocație

**RF** – **R**adio **F**requency

**AP** – **A**ccess **P**oint

**FSPL** – **F**ree-**S**pace **P**ath **L**oss

**MAC** – **M**edia **A**ccess **C**ontrol

**NNSS** – **N**earest **N**eighbor in **S**ignal **S**pace

**NN** – **N**earest **N**eighbor

**kNN** – **k**-**N**earest **N**eighbors

**WkNN** – **W**eighted **k**-**N**earest **N**eighbors

**EWkNN** – **E**nhanced **W**eighted **k**-Nearest **N**eighbors

**BSSID** – **B**asic **S**ervice **S**et **Id**entification

**RSSI** – **R**eceived **S**ignal **S**trength **I**ndication

**JSON** – **J**ava**S**cript **O**bject **N**otation

Abstract

Serviciile bazate pe locație sunt în plină creștere o dată cu intrarea în era mobilității. Există două tipuri de localizare: indoor și outdoor. Localizarea outdoor este folosită la scară largă cu ajutorul navigației prin satelit, acronimul GPS fiind corelat cu conceptul de localizare. Pe de altă parte, localizarea indoor nu a beneficiat de o răspândire la fel de mare ca a ramurei gemene. Eforturile făcute în acest domeniu nu s-au concretizat într-un standard „de facto”, dar în ultimii ani progresul în domeniu este vizibil. Proliferarea dizpozitivelor mobile și a rețelelor wireless au crescut interesul către acest domeniu și au facilitat adopția mai multor tehnologii.

Scopul acestui proiect este de a demonstra funcționarea unui astfel de sistem într-un mediu practic și de a face o comparație între mai mulți algoritmi de localizare.

Conținut

[Figuri 6](#_Toc391317515)

[Ecuații 6](#_Toc391317516)

[Tabele 7](#_Toc391317517)

[1. Introducere 7](#_Toc391317518)

[2. Studiul literaturii din domeniu 8](#_Toc391317519)

[2.1 Elemente teoretice generale 8](#_Toc391317520)

[2.1.1 Propagarea semnalului 8](#_Toc391317521)

[2.1.2 Cadre beacon 9](#_Toc391317522)

[2.1.3 Radiotap headers 10](#_Toc391317523)

[2.2 Articole în domeniu 11](#_Toc391317524)

[2.2.1 Amprente radio 11](#_Toc391317525)

[2.2.2 Trilaterație 11](#_Toc391317526)

[2.3 Rezultate 13](#_Toc391317527)

[2.4 Model comercial 13](#_Toc391317528)

[3. Descrierea implementării 14](#_Toc391317529)

[3.1 Android API 15](#_Toc391317530)

[3.2 Încărcarea unei hărți 16](#_Toc391317531)

[3.3 Colectarea datelor (antrenament) 16](#_Toc391317532)

[3.4 Selectarea și parsarea datelor 20](#_Toc391317533)

[3.5 Localizare 21](#_Toc391317534)

[3.5.1 Amprente radio 21](#_Toc391317535)

[3.5.2 Trilaterație 25](#_Toc391317536)

[4. Concluzii 27](#_Toc391317537)

Figuri

[Fig. 1. Componența unui cadru Beacon [4] 9](#_Toc391049159)

[Fig. 2. Cele 3 cazuri ale trilaterației 11](#_Toc391049160)

[Fig. 3. Variabilele sistemului 11](#_Toc391049161)

[Fig. 4. Harta încăperii 14](#_Toc391049162)

[Fig. 5. Puterea semnalului pentru fiecare AP 17](#_Toc391049163)

[Fig. 6. Harta semnalului pentru un singur AP. 18](#_Toc391049164)

[Fig. 7. kNN pentru k = 2 21](#_Toc391049165)

[Fig. 8. kNN pentru k = 3 22](#_Toc391049166)

[Fig. 9. Distanțe vs. RSSI pentru un AP 24](#_Toc391049167)

[Fig. 10. Curba atenuării semnalului pentru un AP 25](#_Toc391049168)

Ecuații

[Ecuația 1. FSPL 8](#_Toc391049194)

[Ecuația 2. FSPL(dB) 8](#_Toc391049195)

[Ecuația 3. FSPL(dB) pentru Metri-MHz 8](#_Toc391049196)

[Ecuația 4. Sistemul pentru trilaterație 11](#_Toc391049197)

[Ecuația 5. Atenuarea semnalului 11](#_Toc391049198)

[Ecuația 6. Distanța euclidiană clasică 20](#_Toc391049199)

[Ecuația 7. Distanța euclidiană pentru n dimensiuni 20](#_Toc391049200)

[Ecuația 8. Formula pentru WkNN 23](#_Toc391049201)

Tabele

[Tabel 1. Eroarea de poziționare a fiecărui algoritm 13](#_Toc391049266)

# Introducere

Ultimii ani au oferit o creștere a numărului de utilizatori mobili în întreaga lume. Implicit, serviciile bazate pe locație (SBL) au cunoscut un real progres întrucât ele pot oferi informații practice utilizatorilor. Să analizăm sistemul global de poziționare prin satelit care este utilizat la scară largă. Această tehnologie, deja matură, stă în spatele unui număr tot mai mare de aplicații care vin să ajute utilizatorul în viața de zi cu zi. De câte ori nu ne-am aflat într-un oraș necunoscut și am căutat poziția noastră pe hartă folosind Google Maps? Sau de câte ori nu am căutat indicații de orientare din punctul unde ne aflam? Aceste servicii sunt posibile cu ajutorul GPS și a aplicațiilor care se folosesc de acest serviciu. GPS-ul reprezintă însă doar jumătate din SBL. Cealaltă jumătate, mult mai puțin răspândită, o reprezintă poziționarea în interior. Poziționarea în interior vine să rezolve problema localizării în cadrul unei clădiri, acolo unde GPS-ul nu mai este de ajutor.

Localizarea în interior se face pe baza unui sistem RF. Proliferarea dizpozitivelor mobile a atras după sine răspândirea rețelelor wireless, astfel că cel mai propice sistem RF la ora actuală îl reprezintă mediul Wi-Fi. Densitatea mare a punctelor de acces din majoritatea clădirilor moderne atrage o acuratețe bună a metodei, iar totodată nu necesită costuri suplimentare pentru implementare. De menționat sunt si eforturile în domeniul Bluetooth Low Energy concretizate prin standardul iBeacon[[1]](#footnote-2) al celor de la Apple, mențiune venită să susțină ipoteza că mai multe medii RF pot oferi servicii de localizare.

Această lucrare își propune să ofere un exemplu concret de localizare în interiorul unei clădiri, folosind mediul Wi-Fi 802.11. Toate experimentele s-au efectuat la etajul 3 al clădirii Freescale Semiconductor România, folosind un dispozitiv mobil Android ca echipament de test.

# 2. Studiul literaturii din domeniu

Domeniul localizării în interior este unul foarte cercetat începând cu anii 1990. La o simplă căutare pe Internet a patentelor în domeniul „indoor positioning”, observăm că unul dintre primele patente publicate se numește „Interior GPS Navigation” (număr de publicație *US 5959575 A*), depus în toamna anului 1997 și publicat doi ani mai târziu. În acesta este descrisă o metodă prin care tehnologia GPS ar putea funcționa pentru medii indoor, cu ajutorul unor dispozitive adiționale. Citând doar primul pas al metodei din secțiunea *SUMMARY OF THE INVENTION*:

„*[...] The method comprises the following steps: (a) disposing a plurality of ground transceivers in proximity of an indoor area to be navigated through [...]*”

ne putem da seama de ce GPS-ul nu reprezintă o opțiune viabilă pentru localizarea indoor. Adăugarea unor dispozitive speciale pentru un singur scop aduce din start un neajuns enorm. În zilele noastre localizarea în interior se poate folosi de infrastructura Wi-Fi tot mai răspândită, fără a trebui investiți bani în aparatură adițională.

În acest capitol voi prezenta elementele teoretice legate de propagarea semnalului și mediul 802.11, continuând cu prezentarea unor articole științifice din domeniu relevante pentru lucrarea de față și finalizând cu o scurtă descriere a tehnologiilor folosite de marile companii pentru localizarea indoor.

## 2.1 Elemente teoretice generale

### 2.1.1 Propagarea semnalului

Mediul indoor are o structură diferită față de cel outdoor. El poate conține elemente (pereți, mobilă) ce pot cauza reflecție, difracție, împrăștiere, toate acestea provocând atenuarea semnalului la receptor. În [1] este expusă următoarea noțiune: caracteristica de propagare a semnalului într-un astfel de mediu trebuie să fie modelată de un set de aproximări, iar studiile efectuate asupra modelărilor dau naștere a două categorii:

* Modele statistice
* Modele specifice bazate pe teoria electromagnetică (ex. Ray tracing)

În prima categorie intră și modelul **Free-space path loss**. FSPL dă ecuația atenuării semnalului radio într-un mediu fără obstacole, obstacole ce ar putea cauza procesele fizice expuse mai sus: reflecție, difracție etc. Ecuația este următoarea:

Ecuația 1. FSPL

unde:

* f – frecvența semnalului, în Hz
* d – distanța față de transmițător, în metri
* c – viteza luminii, în metri/secundă

În acestă ecuație, exponentul are valoarea 2, valoarea specifică FSPL. Alte valori ale exponentului pot ține cont și de caracteristica mediului [2]:

* 1.4 – 1.9: ghid undă (coridoare, tuneluri)
* 3: FSPL și multicăi (camere cu mobilă)
* >3: traversare pereți, podele

Se întâlnește adesea ca puterea semnalului să aibă ca unitate de măsura decibeli. Ecuația devine (înlocuind exponentul cu necunoscuta n):

Ecuația 2. FSPL(dB)

iar pentru d în metri și f în MHz, ultimul termen poate fi calculat:

Ecuația 3. FSPL(dB) pentru Metri-MHz

Ne întrebăm acum dacă pe baza acestei formule putem determina distanța (d) de la un receptor la un transmițător (în cazul aplicației de față de la un dispozitiv Android la un AP). Singura problemă ce intervine aici este valoarea lui n, care nu poate fi prezisă până nu se efectuează o serie de experimente. Lucrarea de față își propune ca după măsurătorile efectuate să încerce să dea un răspuns la această întrebare.

### 2.1.2 Cadre beacon

Lucrarea de față se bazează pe un concept important din specificația IEEE 802.11, și anume transmisia periodică de cadre de tip **beacon**. Un astfel de cadru face parte din familia de cadre de management ale rețelelor 802.11.

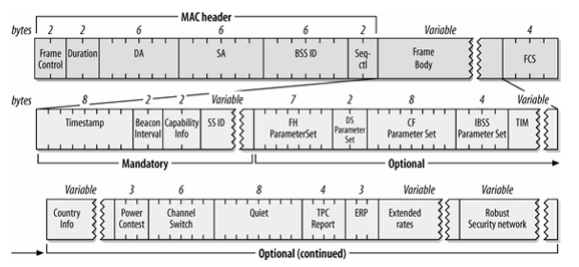


Fig. 1. Componența unui cadru Beacon [3]

Se observă că un cadru beacon este format din: MAC header, Frame Body și FCS. Poziționarea în interior bazată pe amprente radio are nevoie de doar două valori: BSSID – adresa MAC a AP-ului care va furniza practic identitatea sa, RSSI – puterea semnalului recepționat la dispozitivul utilizatorului. Prima valoare se află în MAC header, astfel că prima jumătate este asigurată. A doua valoare, în schimb, nu este inclusă în componența cadrului. Acest lucru se datorează faptului că AP-ul nu are cum să determine puterea cu care un dispozitiv va recepționa beacon-ul transmis. Aici intervin Radiotap headers, prezentate în secțiunea următoare.

Mai prezintă interes și intervalul de timp la care aceste cadre sunt trimise. Frame body conține un câmp pe doi octeți numit Beacon Interval. El reprezintă numărul de unități de timp dintre transmisii succesive, unde o unitate de timp (TU – time unit) are valoarea 1024 µs. Este uzual ca Beacon Interval să fie egal cu 100 TU, ceea ce înseamnă că un beacon va fi transmis la fiecare cca. 0.1s [3]. Întrebarea ce apare aici este câte astfel de cadre va recepționa dispozitivul Android de antrenament. Această întrebare primește un răspuns, cel puțin parțial, în secțiunea *3.1 Android API*.

### 

### 2.1.3 Radiotap headers

Radiotap este standardul de facto pentru recepționarea și injectarea cadrelor 802.11. Mai multe sisteme de operare suportă această tehnologie, ele fiind: FreeBSD, Linux, NetBSD, OpenBSD și Windows. Android fiind un sistem de operare bazat pe kernelul Linux, dispune și el de această tehnologie (sursele Android au fost descărcate de pe site-ul oficial[[2]](#footnote-3)):

adrian@debian:~/source.android.com$ find . | grep radiotap

./external/wpa\_supplicant\_8/src/utils/**radiotap.c**

./external/wpa\_supplicant\_8/src/utils/**radiotap\_iter.h**

./external/wpa\_supplicant\_8/src/utils/**radiotap.h**

Headerele Radiotap sunt practic niște informații suplimentare despre cadrele recepționate. Acestea sunt adăugate de către driverul plăcii de rețea. După cum spuneam în secțiunea anterioară, interesează aflarea puterii semnalului la recepție. În termeni Radiotap[[3]](#footnote-4), aceasta se numește **Antenna Signal**. Specificația spune că acest câmp este pe 8 biți și indică puterea semnalului RF la antenă, în dBm. Rezoluția este de 1mW.

Pe lângă semnalul la antenă, mai există cazul în care poate interesa canalul 802.11 pe care se efectuează operațiile de transmisie/recepție. Câmpul **Channel** din Radiotap furnizează această frecvență, în MHz. Vorbesc despre acest câmp deoarece el ar putea fi de ajutor în calcularea distanței față de AP. FSPL (Free-Space Path Loss) spune că distanța până la AP este o funcție de puterea semnalului și de frecvența purtătoarei. Mai multe despre acest concept au fost prezentate în secțiunea *2.1.1 Propagarea semnalului*.

## 2.2 Articole în domeniu

### 2.2.1 Amprente radio

Există o vastă literatură în domeniul prezentat, dar în general metodele prezentate în articole se împart în două mari categorii: metode empirice și metode probabilistice. Această lucrare abordează prima categorie, lăsând loc cercetării metodelor probabilistice în viitor. Cel mai reprezentativ articol pentru prima categorie, și implicit pentru această lucrare, îl reprezintă [4]. Voi descrie pe scurt abordarea făcută în articol:

* Pentru a reuși localizarea, este nevoie de o încăpere în care să existe AP-uri poziționate astfel încât întreaga suprafață să fie acoperită. Este de preferat să se cunoască dimensiunile încăperii pentru a estima în sistemul metric eroarea de calcul a poziției.
* Procesul începe cu faza de colectare. Faza de colectare presupune deplasarea prin clădire cu un dispozitiv capabil Wi-Fi cu posibilitate de monitorizare a beacon-urilor primite de la AP-uri. Beacon-urile oferă mai multe informații despre emițător, dar relevant pentru lucrare îl reprezintă valoarea puterii semnalului. Baza de date formată la acest pas va conține coordonatele punctelor în care s-a realizat colectarea și puterea semnalului în acel punct. Acest proces mai este numit și *off-line phase*.
* După ce se realizează colectarea (și se prelucrează datele prin medierea valorilor obținute) poate începe faza de localizare (*on-line phase*). Aici se propune algoritmul *nearest neighbor(s) in signal space (NNSS)*. Când un utilizator va cere poziția sa în încăpere, se va compara puterea semnalului în acel loc cu valorile din baza de date și se va alege acel punct care va minimiza distanța în spațiul puterii semnalului. Pentru minimizare se poate folosi, de exemplu, distanța euclidiană.

Algoritmul NN este un algoritm foarte simplu, dar care are și eroarea cea mai mare. El ia în calcul un singur punct, și anume cel mai apropiat de poziția utilizatorului, pentru a determina locația. O îmbunătățire a acestui algoritm îl reprezintă kNN, unde k este un număr fix și reprezintă numărul de puncte luate în considerare pentru furnizarea locației. Algoritmul face o medie a coordonatelor celor mai apropiate puncte și astfel eroarea scade. Pe lângă acestea mai există și alți algoritmi derivați ce sunt prezentați în secțiunea *2.3 Rezultate*.

De menționat este și [5], poate cel mai reprezentativ articol pentru a doua categorie de metode de localizare, cele probabilistice.

### 2.2.2 Trilaterație

Trilaterația este o metodă de determinare a poziției unui obiect în funcție de distanța acestuia față de cel puțin 3 puncte de referință. Fiecare din aceste puncte poate fi considerat centrul unui cerc de rază distanța până la obiectul ce trebuie localizat. Aici se disting două cazuri: cercurile descrise de cele 3 puncte se intersectează total, parțial sau deloc.

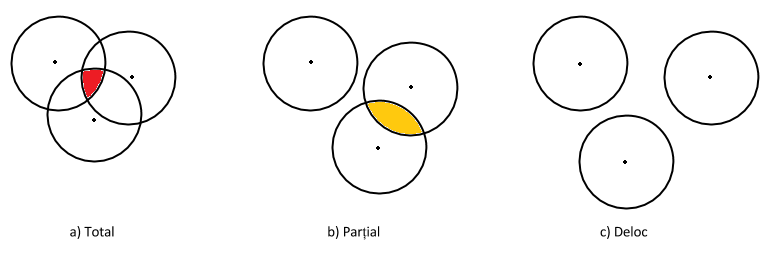


Fig. 2. Cele 3 cazuri ale trilaterației

Deși este lesne de înțeles că primul caz expus este cel mai favorabil, oferind cea mai bună eroare de poziționare, de multe ori în practică scenariul nu se reproduce. Totuși, indiferent de dispunerea AP-urilor și gradul de intersecție a lor, trebuie urmați anumiți pași pentru determinarea poziției care minimizează eroarea. În [6] este prezentată o metodă ce rezolvă problema anterior menționată. Practic, sistemul ce trebuie rezolvat este, în linii mari, următorul:

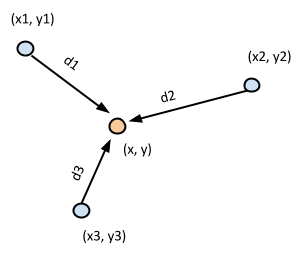


Fig. 3. Variabilele sistemului

Ecuația 4. Sistemul pentru trilaterație

Se observă că avem 3 ecuații cu 11 necunoscute. Trilaterația presupune însă că poziția punctelor de referință este cunoscută, astfel că mai rămân doar 5 necunoscute. Altfel spus, mai trebuie aflate și distanțele d1, d2, d3. După cum am menționat și la finalul secțiunii *2.1.1 Propagarea semnalului*, aflarea acestor distanțe nu se poate realiza fără cunoașterea exponentului de pierdere a semnalului (eng. *Path Loss Exponent*). Există totuși o modalitate de aflare a acestor distanțe pe baza datelor colectate pentru poziționarea folosind amprente radio. În [7] este prezentată următoarea ecuație ce descrie atenuarea semnalului:

Ecuația 5. Atenuarea semnalului

unde:

* PL (eng. *Path Loss*) – reprezintă valoarea semnalului (funcție de distanță)
* d – distanța dintre transmițător și receptor
* d0 – distanță de referință
* n – exponentul de atenuare, prezentat anterior

Aflarea constantelor PL(d0), n și d0 se poate face găsind funcția de acești coeficienți care aproximează cel mai bine punctele colectate. Mai multe în secțiunea de implementare, subcapitolul 3.5.2.

## 2.3 Rezultate

În [8] se propune algoritmul EWkNN și totodată se face și o comparație a algoritmilor de localizare empirici. Îmbunătațirile aduse algoritmului NN pot fi sumarizate astfel:

NN ->kNN -> WkNN -> EWkNN

WkNN, unde W provine de la *weighted*, spune că punctele mai apropiate de locația utilizatorului trebuie să aibă o pondere mai mare. Astfel, media calculată pentru furnizarea locației este de fapt o medie ponderată, și nu una aritmetică. EWkNN (E - *enhanced*) propune schimbarea dinamică a lui k. Cei mai apropiați vecini sunt ordonați descrescător, iar cei ce nu respectă un anumit prag de vecinătate sunt eliminați. k-ul se schimbă astfel la fiecare cerere de locație. Deși algoritmul are cea mai mare complexitate, oferă și cea mai bună acuratețe a poziției după spusele autorilor.

Voi prelua rezultatele comparației între acești algoritmi întrucât sunt relevante pentru lucrarea curentă. Algoritmul NN are eroarea de poziționare cea mai mare și este exprimată în metri. În următorul tabel voi spune că eroarea algoritmului este de 100% pentru că furnizarea unor unități de măsură nu este relevantă în acest moment.

|  |  |
| --- | --- |
| Algoritm | Eroarea de poziționare |
| NN | 100% |
| KNN | 70% |
| WkNN | 50% |
| EWkNN | 37% |

Tabel 1. Eroarea de poziționare a algoritmilor bazați pe NN

Date fiind aceste rezultate, lucrarea de față își propune folosirea incrementală a algoritmilor expuși pentru determinarea locației și totodată pentru validarea rezultatelor de mai sus. Este de sperat ca algoritmul EWkNN să furnizeze într-adevăr cea mai mică eroare, complexitatea puțin mărită prin schimbarea dinamică a valorii k nefiind o problemă (echipamentul de test folosit este un telefon mobil Android ce dispune de 4 nuclee).

## 

## 2.4 Model comercial

În ultimul subcapitol al acestei secțiuni voi vorbi despre un model concret de poziționare în interior. Google Inc. abordează acest segment de piață încă din anul 2011, atunci când utilizatorii Android au putut fi, pentru prima dată, ghidați în interiorul unei clădiri[[4]](#footnote-5). Clădirea cu pricina era, desigur, un mall (mai exact Mall of America, Bloomington, Minnesota). Localizarea s-a realizat tot cu ajutorul amprentelor radio, citez:

„[...] *the location-finding abilities of Google Maps depend heavily on broadcasts from the mall’s many Wi-Fi access points*.” David Schneider, IEEE Spectrum Senior Editor

De la acest prim pas, Google a continuat să investească în acest segment de piață. Un an mai târziu, pe 17 octombrie 2012, Google a lansat două noi API-uri pentru a veni în ajutorul dezvoltatorilor[[5]](#footnote-6):

* Google Maps Tracks API – oferă suport pentru GPS
* Google Maps Geolocation API – citez: permite unei aplicații sau dispozitiv să-și determine locația **fără ajutorul GPS-ului**, uitându-se în schimb la **AP-urile 802.11** sau celulele GSM din proximitate

Introducerea acestui de-al doilea API a facilitat dezvoltarea aplicațiilor în acest domeniu, iar rezultatele sunt vizibile. De multe ori, folosind Google Maps, reușim să fim poziționați decent în interiorul unei clădiri de importanță mărită, precum un aeroport de pildă. Utilizarea acestei tehnologii la scară largă presupune de asemenea colectarea beacon-urilor la scară largă. Aici se disting două categorii:

* colectarea realizată de companie: în [9] se arată că, prin intermediul mașinilor pentru colectarea de imagini utilizate în proiectul StreetView se culeg de asemenea și informații privind mediul Wi-Fi. Echipamentul este capabil să proceseze toate cadrele de broadcast (printre care și cele beacon) primite pentru a crea astfel o hartă a semnalului. Fără îndoială, acest lucru a ridicat întrebări referitoare la intimitate, astfel că Google a propus următoarea soluție[[6]](#footnote-7): adăugarea sufixului „\_nomap” la SSID-ul unei rețele o va face „invizibilă” la colectare.
* colectarea realizată de utilizatorii de rând: **Google Maps Floor Plan Marker** este un produs Google ce permite deținătorilor de spații comerciale, în special restaurante, mall-uri, etc., să încarce harta încăperii deja antrenată pentru ca serviciul de Geolocation să funcționeze în cadrul ei. Astfel, utilizatorii sunt rugați să se plimbe prin clădire cu un dispozitiv capabil 802.11 pentru a recolta date relevante din cadrele de broadcast, pentru ca mai târziu acestea să fie încărcate în cloud.

De remarcat este și efortul Apple Inc. făcut în acest domeniu, foarte similar cu cele prezentate până acum (de exemplu, API-ul despre care vorbeam mai sus se numește aici **CoreLocation API**). Ceea ce se distinge însă este și investiția în domeniul hardware [10], acest coprocesor fiind menit, pe lângă altele, să ofere servicii de localizare cu un consum redus de putere.

# 3. Descrierea implementării

După cum menționam în introducere, încăperea în care se va încerca localizarea o reprezintă etajul 3 al clădirii Freescale Semiconductor România. Încăperea are o lungime de cca. 30 metri si o lățime de cca. 15 metri. Spațiul este unul deschis, fără pereți care ar putea atenua semnalul într-un mod ce nu ar permite localizarea. Pe întreg etajul există semnal Wi-Fi de la cel puțin două AP-uri.

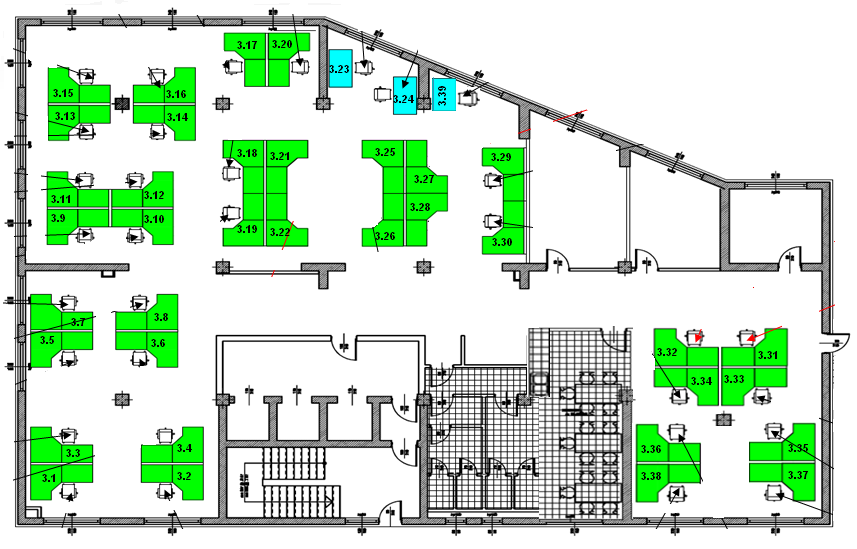


Fig. 4. Harta încăperii

Pentru colectarea datelor despre puterea semnalului am implementat o aplicație Android care oferă următoarele funcționalități:

* Posibilitate de a vedea toate AP-urile din proximitate – utilă pentru a determina dacă în încăpere sunt suficiente AP-uri pentru a încerca localizarea
* Capabilitatea de încărcare a unei hărți proprii (fișierul trebuie să fie în format BMP)
* Colectarea de date din încăpere (folosind harta de mai sus) într-un fișier JSON
* Posibilitatea parsării datelor colectate dintr-un fișier JSON
* Posibilitatea de a selecta metoda (algoritmul) cu care se va încerca poziționarea
* Capabilitate de a afișa poziția curentă, cu istoric (punctele în care utilizatorul a fost identificat nu dispar de pe hartă)

## 3.1 Android API

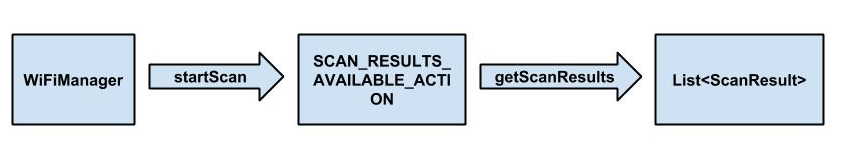
Preluarea datelor despre mediul Wi-Fi pe un dizpozitiv Android se face cu ușurință prin intermediul API-ului expus. Sistemul expune un serviciu numit **WiFiManager[[7]](#footnote-8)** ce se ocupă de toate aspectele conectivității 802.11. Obținerea unei instanțe a acestui serviciu se efectuează prin apelul **Context.getSystemService(Context.WIFI\_SERVICE)**. În cele ce urmează, toate câmpurile și metodele prezentate sunt relative la acest serviciu (clasă Java). De menționat este si faptul că acest modul este prezent în Android încă de la prima versiune, API level 1. Astfel, aplicația poate funcționa pe întreaga gamă de dispozitive Android.

Deși acest serviciu permite utilizatorului accesarea unui număr mare de funcționalități, relevant pentru lucrarea de față o reprezintă doar funcționalitatea de obținere a informațiilor obținute din scanarea AP-urilor. Aici, metoda de interes furnizată de Android se numește **startScan()**.

startScan() face o cerere pentru scanarea mediului. Rezultatele scanării nu sunt returnate imediat, ci asincron. Evenimentul de finalizare a scanării este anunțat prin setarea câmpului **SCAN\_RESULTS\_AVAILABLE\_ACTION**, după care se poate apela metoda **getScanResults()** ce returnează o listă de rezultate. Rezultatele sunt niște obiecte de tip **ScanResult** (clasă Java, de asemenea prezent în API level 1) ce posedă următoarele câmpuri:

* public String **BSSID –** adresa MAC a AP-ului (în modul infrastructură)
* public String **SSID –** numele rețelei
* public int **frequency –** frecvența canalului, în MHz
* public int **level –** puterea semnalului, în dBm

Pe lângă acestea, mai există două alte câmpuri, **capabilities** și **timestamp**, care însă nu prezintă interes acum. Poziționarea bazată pe amprente radio se va face pe baza tuplurilor (BSSID, level). Următoarea diagramă face un rezumat a celor prezentate în acest subcapitol.



**Notă**: Experimental, un apel startScan() durează cca. 500ms pentru finalizare. În acest caz, se poate considera că o scanare ia în considerare între 1-5 cadre beacon pentru raportarea nivelului de semnal, având în vedere cele prezentate în secțiunea *2.1.2 Cadre beacon*.

## 3.2 Încărcarea unei hărți

Aplicația vine preîncărcată cu harta etajului Freescale folosită în perioada de dezvoltare. Un utilizator va dori însă să încarce propria hartă pentru a realiza poziționarea în diferite încăperi, după bunul plac. La simpla apăsare a unui buton este afișat un **DialogFragment**[[8]](#footnote-9)ce ia forma unui selector de fișiere. Practic, acesta permite utilizatorului să selecteze o imagine care va putea fi folosită apoi în faza de colectare. Aplicația listează toate fișierele cu extensia .bmp din rădăcina dizpozitivului de stocare extern. În cod, aceasta se traduce prin **Environment.*getExternalStorageDirectory*** (ex.: */storage/emulated/0* pentru LG Nexus 4).

Menționam mai devreme că imaginea trebuie să fie în format BMP. Aceasta se datorează faptului că afișarea efectivă a imaginii se realizează prin intermediul clasei **android.graphics.Bitmap**[[9]](#footnote-10)ce poate gestiona doar fișiere de acest tip. Indiferent de harta pentru poziționare, fie că este cea preinstalată, fie că este una aleasă la runtime, intern ea se va transforma într-un obiect de tip Bitmap prin apelarea funcției createScaledBitmap ce are următorul antet:

public static Bitmap createScaledBitmap(Bitmap src, int dstWidth, int dstHeigth, boolean filter)

Apelată cu parametri potriviți, ea realizează scalarea imaginii la dimensiunea ecranului pentru o mai bună experiență utilizator.

Peste imagine este desenată apoi o grilă (eng. *grid*). Aceasta vine în ajutorul fazei de colectare, oferind o acuratețe mai bună la selectarea punctelor de antrenament.

## 3.3 Colectarea datelor (antrenament)

Funcționalitatea de colectare a datelor este poate cea mai importantă a acestei aplicații Android. Colectarea datelor presupune ca dezvoltatorul să își aleagă o serie de puncte de pe hartă, preferabil echidistante, iar apoi să se plimbe cu echipamentul ce are aplicația instalată (fie telefon, fie tabletă) prin aceste puncte. La fiecare popas, este necesar un singur click pe hartă pentru ca puterea semnalului în acel punct să fie salvată. Mai exact se salvează 5 mostre ale puterii semnalului pentru ca eventualele iregularități: oameni deplasându-se în proximitatea aparatului etc. să fie nivelate. Astfel, la finalul fazei de colectare, se va crea un fișier JSON (am ales acest tip de fișier deoarece este relativ simplu de parsat în orice limbaj de programare) cu tupluri (x, y, Samples[]) unde fiecare obiect Sample va conține tupluri de forma (BSSID, RSSI). Un exemplu de token JSON ce conține date despre un singur punct:

|  |  |
| --- | --- |
| {   "point1": {    "x": 518,    "y": 577,    "sample1": [     {      "bssid": "00:1f:1f:2c:35:e8",      "rssi": -84     },     {      "bssid": "00:17:0f:da:ba:e2",      "rssi": -71     },     {      "bssid": "00:17:0f:da:ba:43",      "rssi": -52     }, [...]  [...]  } | Se observă ca fiecare punct este numerotat, iar fiecare Sample în interiorul unui punct de asemenea. Poziția (x, y) reprezintă coordonatele pixelului selectat de către utilizator. Chiar și pentru scopul final al aplicației, acela de a estima poziția pe hartă, nu trebuie făcută conversia la sistemul metric (va trebui făcută în schimb pentru raportarea erorilor).  Mai departe sunt salvate tuplurile de care vorbeam mai sus. Un aspect important îl reprezintă filtrarea ulterioară a anumitor BSSID-uri. Se observă în exemplu că două BSSID-uri au același prefix 00:17:0f:da:ba: . Acest lucru poate însemna că AP-urile ce respectă această proprietate se află sub administrare comună, de unde putem deduce că ele nu își vor schimba poziția în timp. |

Într-adevăr, după cum ulterior am aflat, acest prefix este specific Freescale, AP-urile făcând parte din aceeași familie de produse. Cel de-al treilea AP din exemplu poate fi considerat unul temporar, deoarece se afla pe biroul unui angajat. Acest aspect este important deoarece schimbările poziției AP-urilor pot cauza erori majore de poziționare. Aplicația de față salvează toate datele capturate în fișierul JSON, dar ia în calcul pentru poziționare doar pe cele cu prefixul specificat. Totuși, când se încearcă poziționarea în alte încăperi, această restricție dispare.

Ceea ce nu a fost menționat în secțiunea anterioară este faptul că harta oferă funcționalități de **drag** și **zoom**. Acest lucru este îndeosebi folositor pentru dispozitive Android cu ecran mai mic, deoarece oferă o acuratețe mult mai bună preluării poziției selectate (tuplul (x, y)).

O dată creat acest fișier, în perioada de dezvoltare se pot analiza diverse caracteristici ale mediului. Mai multe despre interpretarea lor în secțiunea *4.1 Analiza preliminară*.

## 

## 3.4 Selectarea și parsarea datelor

Similar cu funcționalitatea de alegere a unei hărți, utilizatorul are posibilitatea să aleagă orice fișier de tip JSON pe baza căruia dorește să se poziționeze în încăpere. Deși acest aspect are mai degrabă un rol important în partea de dezvoltare, este prezent și în aplicația finală din varii motive. Este oferit astfel un procedeu de analiză în timp a rezultatelor obținute. Să presupunem că în ziua Z cu baza de date B1 se obține eroarea E1, iar în ziua Z + 30, după o nouă antrenare în urma căreia rezultă baza de date B2, se obține eroarea E2. În condițiile în care aceleași AP-uri sunt vizibile în încăpere și comparând E1 cu E2, se pot trage anumite concluzii de genul:

* anumite AP-uri au fost relocate
* încăperea a fost modificată (posibil mobilier dispus altfel)
* există surse de interferență

După alegerea fișierului de antrenament, este necesară parsarea acestuia și stocarea datelor în obiecte Java. Parsarea se realizează cu ajutorul unui obiect JsonReader[[10]](#footnote-11) ce este capabil să consume unul câte unul token-uri din fișierul JSON selectat.

Pentru stocarea datelor, am implementat două clase container, prima numită *PointData* ce deține următoarele câmpuri relevante:

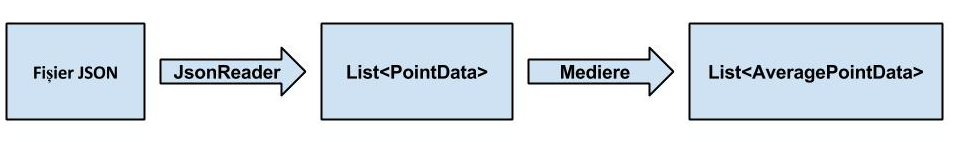
* int x – abscisa punctului de coordonate
* int y – ordonata punctului de coordonate
* List<Map<String, Integer>> samples – lista ce conține toate mostrele recoltate într-un anumit punct (x, y) sub formă de dicționar cu tupluri (BSSID, RSSI); după cum am mai menționat, în fiecare punct se fac mai multe citiri pentru ca eventualele iregularități să fie nivelate ulterior

și a doua numită *AveragePointData* pentru soluționarea problemei mai sus expuse, cea de mediere a datelor, diferența între ele fiind că

List<Map<String, Integer>> **samples** --> Map<String, Integer> **averageSample**

De menționat este că procesul de mediere ține cont și de numărul de prezențe pe care un anumit AP îl înregistrează în mostrele colectate. De exemplu, dacă tuplul (BSSID, RSSI)(AP1) apare în 4/5 mostre într-un anumit punct, media va fi una aritmetică cu numitorul 4. Am ales această abordare deoarece uneori, deși foarte rar, scanarea AP-urilor în Android mai poate pierde beacon-uri, iar acest neajuns nu ar trebui să influențeze valoarea finală.

Cele prezentate mai sus pot fi sumarizate în următoarea diagramă:

****

Mai departe, procesul de localizarea va folosi doar datele obținute după mediere, sub formă de listă de obiecte *AveragePointData*.

## 

## 3.5 Localizare

### 3.5.1 Amprente radio

În secțiunea *2.3 Rezultate* prezentam mai multe tipuri de algoritmi de poziționare bazați pe amprente radio, toți fiind derivați din algoritmul Nearest Neighbour. Aceștia erau NN, KNN, WKNN și EWKNN. Utilizatorul poate alege din aplicație algoritmul pe care dorește să îl folosească; de altfel, după selectarea unui algoritm se poate alege cu ușurință un altul, pentru vizualizarea în timp real a diferențelor între pozițiile obținute.

În secțiunea anterioară spuneam că pentru localizare se va folosi o listă de obiecte *AveragePointData* drept date offline. Datele online reprezintă o singură mostră preluată ce se traduce într-un tuplu (x, y, Sample), unde Sample este un nume dat obiectului Java HashMap<String, Integer>. Datele online sunt preluate similar procedeului din secțiunea *3.3 Colectarea datelor*, dar în loc să se dea click pe hartă, totul se produce la apăsarea butonului *Where Am I?* (de asemenea, o singură mostră este preluată pentru că acum nu se mai dorește nivelarea).

Următorul pas îl constituie compararea acestor date online cu fiecare dintre datele offline. Astfel, se dorește stabilirea unei ierarhii între punctele offline, în funcție de „apropierea” lor față de acest punct online. După cum se menționează în [1], mai multe metrici pot fi folosite, printre care:

* distanța euclidiană
* distanța Manhattan
* metrică proprie bazată pe anumite ponderi alocate AP-urilor

Aplicația implementează doar prima variantă, distanța euclidiană. Aceeași metrică este folosită și în [1].

#### 3.5.1.1 Distanța euclidiană în spațiul semnal

În matematică, distanța euclidiană reprezintă distanța obișnuită dintre două puncte în plan, calculată pe baza teoremei lui Pitagora. Astfel, dacă avem două puncte x(x1, x2) și y(y1, y2), distanța euclidiană este dată de formula:

Ecuația 6. Distanța euclidiană clasică

Acesta reprezintă un exemplu de folosire a distanței euclidiene într-un spațiu bidimensional. Totuși, formula se poate extinde cu ușurință la un spațiu n-dimensional:

Ecuația 7. Distanța euclidiană pentru n dimensiuni

În cazul aplicației de față, distanța ce trebuie calculată este într-adevăr una în spațiu n-dimensional, unde n reprezintă numărul de AP-uri vizibile (mai exact semnalul preluat de la fiecare dintre acestea) prezente în datele online. Numim acest spațiu spațiul semnal, iar distanța ce trebuie calculată se bazează pe următorul algoritm:

distance = 0

for Entry<String, Integer> in onlineData:

for Entry<String, Double> in offlineData:

if key1 equals key2:

distance += (value1 – value2)^2

return sqrt(distance)

Se observă aici că prima buclă iterează peste obiecte de tip <String, Integer>, iar ce-a de-a doua după obiecte de tip <String, Double>. Diferența provine din faptul că API-ul Android furnizează valoarea semnalului recepționat ca Integer (ex. -42 dBm) – onlineData, pe când datele din offlineData au fost mediate și pot deveni raționale.

Există deci o soluție pentru sortarea datelor offline în funcție de apropierea lor față de datele online. Să numim această colecție sortată *sortedOfflineData*. Următoarele secțiuni prezintă algoritmii folosiți în aplicație, foarte ușor de înțeles având la dispoziție această structură de date.

#### 3.5.1.2 NN

Algoritmul Nearest Neighbour, după cum îi spune și numele, va considera că poziția în care se află utilizatorul este cea care minimizează distanța față de datele online. Acestea fiind spuse, poziția întoarsă va fi:

first element of **sortedOfflineData**

#### 3.5.1.3 kNN

Algoritmul k Nearest Neighbours va calcula poziția curentă pe baza celor mai apropiate k puncte. Poziția curentă va fi media aritmetică a acestor puncte. Întrebarea ce apare aici este „Cât trebuie să fie k?”. Analizăm diversele valori pe care k le poate lua:

* k = 1: algoritmul devine NN
* k = 2: kNN va întoarce poziția curentă ca fiind jumătatea segmentului determinat de cele mai apropiate două puncte. Această estimare nu este foarte bună, întrucât teoretic poziția curentă poate fi de fapt orice punct de pe dreapta egal depărtată de aceste cele mai apropiate două puncte.

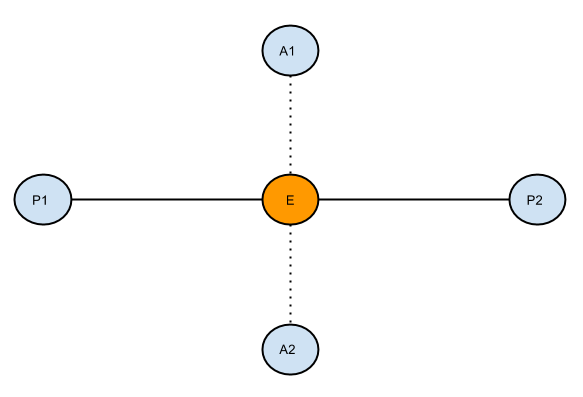


Fig. 5. kNN pentru k = 2

În figura de mai sus, avem:

* P1, P2 – cele mai apropiate 2 puncte offline
* E – poziția estimată
* A1, A2 – candidați pentru poziția actuală

Se observă că pentru k = 2, poziția estimată va fi E, iar eroarea poate fi foarte mare dacă A1 sau A2 sunt de fapt pozițiile curente.

* k = 3: aici lucrurile stau mult mai bine, deoarece poziția estimată va fi centrul unui triunghi, iar poziția curentă nu poate fi în afara acestuia:

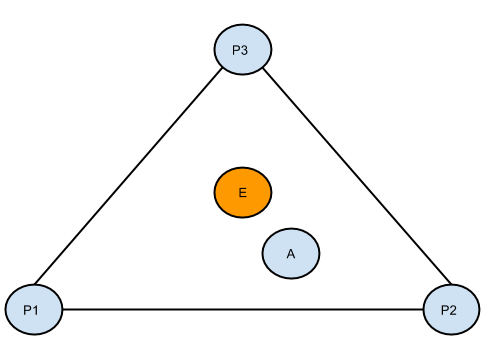


Fig. 6. kNN pentru k = 3

Se observă că adăugarea unui punct atrage după sine o mult mai bună valoare a erorii de poziționare.

* k > 3: eroarea de poziționare poate scădea în continuare, dar destul de lent. După un anumit prag, totuși, acuratețea scade. Aceasta se datorează faptului că puncte din ce în ce mai îndepărtate sunt incluse în soluție [1].

Acestea fiind spuse, aplicația de față folosește valoarea 3 pentru k. De altfel, în mai multe lucrări de specialitate, printre care și [1], aceeași valoare este folosită.

Tradus în pseudocod, algoritmul arată astfel:

kNN(k):

sumX = 0

sumY = 0

for index in range k:

offlinePoint = sortedOfflineData.get(index)

if offlinePoint is null:

return null

sumX += offlinePoint.x

sumY += offlinePoint.y

return new Point(sumX / k, sumY / k)

Există posibilitatea ca datele online să nu conțină k BSSID-uri comune cu datele offline. În acest caz se va returna *null*, iar utilizatorul va fi întâmpinat cu un mesaj cu textul „Not Enough Data..”. Aceasta însemnă că datele online conțin prea puține informații pentru ca localizarea să se poate realiza cu algoritmul selectat.

#### 3.5.1.4 WkNN

Algoritmul Weighted k Nearest Neighbours reprezintă, după cum îi spune și numele, o îmbunățire a algoritmului precedent. Dacă la kNN poziția era estimată ca fiind media aritmetică a celor mai apropiate k puncte, aici media folosită va fi una ponderată. Ponderile vor fi chiar inversul distanțelor calculate în secțiunea *3.5.1.1.* Formula arată astfel, păstrând notația Pi pentru punctele de antrenament, și adăugând Di ca fiind distanța față de punctul Pi:

Ecuația 8. Formula pentru WkNN

În pseudocod avem:

WkNN(k):

sumX = 0

sumY = 0

denominator = 0

for index in range k:

offlinePoint = sortedOfflineData.get(index)

if offlinePoint is null:

return null

sumX += offlinePoint.x / offlinePoint.distance

sumY += offlinePoint.y / offlinePoint.distance

denominator += 1 / offlinePoint.distance

return new Point(sumX / denominator, sumY / denominator)

Acest algoritm propune o îmbunătățire a erorii de poziționare cu până la 50% ([3]) față de precedentul.

#### 3.5.1.4 EWkNN

Ultima îmbunătățire adusă o reprezintă algoritmul Enhanced WkNN. Singura modificare prezentă aici o reprezintă ajustarea dinamică a valorii k pe baza unei valori de prag (eng. threshold). Această ajustare garantează faptul că întotdeauna se va lua cea mai bună decizie în privința valorii k. Astfel, cele mai apropiate puncte (apropierea fiind stabilită de o valoare de prag a distanței, și nu de o valoare k prestabilită) vor avea aportul lor la calcularea poziției curente. În pseudocod:

EWkNN(threshold):

counter = 0

for offlinePoint in sortedOfflineData:

if offlinePoint.distance <= threshold:

counter++

return WkNN(counter)

### 3.5.2 Trilaterație

Pentru această parte am abandonat dezvoltarea pe Android și am lucrat cu Python cu ajutorul modulelor **NumPy**[[11]](#footnote-12), **SciPy**[[12]](#footnote-13) și **matplotlib**[[13]](#footnote-14). După cum spuneam și în secțiunea 2.2.2, poziționarea prin trilaterație poate reuși cu ajutorul datelor colectate în perioada de antrenare.

Un prim pas îl reprezintă identificarea poziției AP-urilor în încăpere. În cazul etajul 3 de la Freescale, AP-ul cu MAC-ul **00:17:0f:da:ba:42** se află, mai mult sau mai puțin, la poziția (612, 1063) în pixeli. El este vizibil cu ochiul liber, astfel că poziția sa a fost determinată apăsând pe harta din aplicație în punctul respectiv. Următorul pas îl constituie calcularea distanței față de el a tuturor punctelor colectate în faza offline. Acestea se pot calcula cu ușurință cu ajutorul distanței euclidiene pentru două puncte. S-a putut genera astfel următorul grafic:

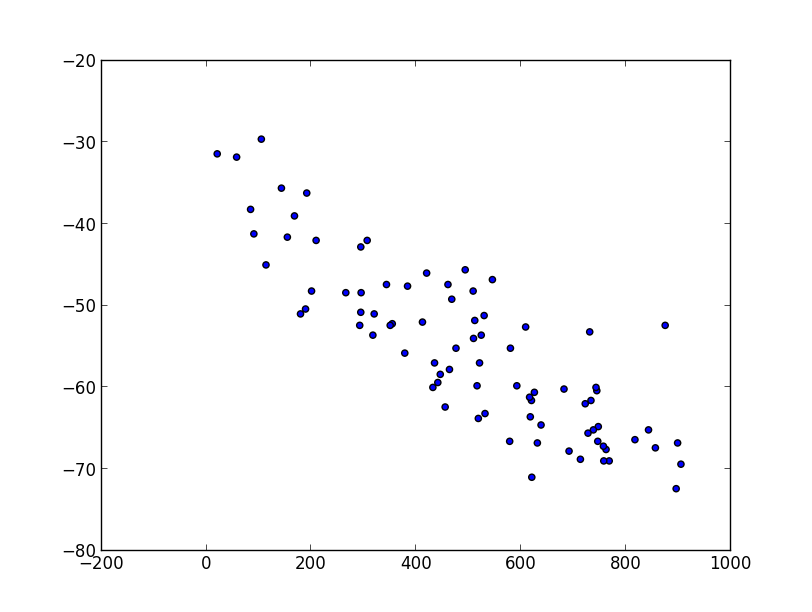


Fig. 7. Distanțe vs. RSSI pentru un AP

Axa ordonatei este dată de puterea semnalului, în dBm, iar axa abscisei conține distanțele față de AP-ul în cauză, în pixeli. Se observă că nu există distanță negativă, fapt de altfel imposibil. Următorul pas îl constituie găsirea funcției care aproximează cel mai bine aceste puncte.

Modulul SciPy anterior menționat expune funcția:

scipy.optimize.curve\_fit(f, xdata, ydata, p0=None, sigma=None, absolute\_sigma=False, \*\*kw)[[14]](#footnote-15)

ce folosește metoda celor mai mici pătrate pentru a face fit unei funcții f pe un set de date, unde:

* f – funcția model f(x, \*popt); aceasta ia variabila independentă x ca prim parametru, iar apoi un număr variabil de coeficienți. Metoda curve\_fit va găsi valorile pentru acești coeficienți astfel încât funcția să aproximeze optim punctele de intrare
* xdata – abscisele punctelor de intrare
* ydata – ordonatele punctelor de intrare
* p0 – listă cu valorile inițiale ale coeficienților; dacă nu este furnizată, toți coeficienții sunt considerați 1

Funcția întoarce o listă cu valorile optime pentru coeficienți astfel încât suma pătratelor erorilor generate de f(xdata, \*popt) – ydata să fie minimizată. După cum spuneam în secțiunea TODO formula TODO, funcția f este următoarea (scrisă în Python):

**>>> def** func(x, P0, n, d0):

**...**  **return** P0 – 10 \* n \* numpy.log10(x / d0)

[...]

**>>>** popt, \_ = curve\_fit(func, distance\_data, rssi\_data)

ceea ce pentru AP-ul **00:17:0f:da:ba:42** întoarce:

**>>>** printpopt

**[12.12662913, 2.8279147, 1.74990549]**

Se remarcă aici un lucru important, găsirea exponentului de atenuare n la care promiteam că vom da un răspuns încă din secțiunea *2.1.1 Propagarea semnalului*. Pentru datele respective de intrare și pentru acest AP, n are valoare 2.828. Un coeficient apropiat de valoarea 3 înseamnă că mediul conține obiecte ce cauzează fenomenul de multipath, cel mai adesea mobilă. Cu acești coeficienți găsiți graficul funcției arată astfel:

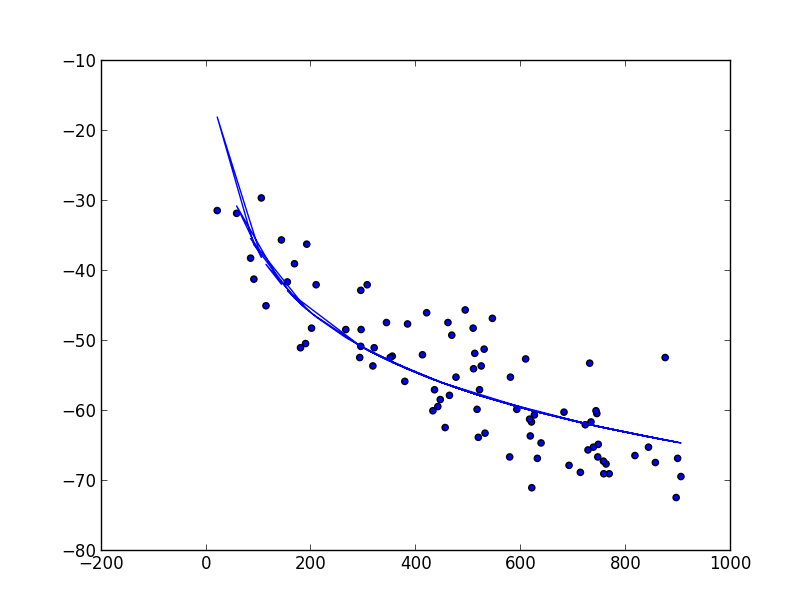


Fig. 8. Curba atenuării semnalului pentru un AP

# Se dorește acum ca pentru o valoare a puterii semnalului arbitrară, colectată online, să se identifice distanța teoretică față de AP. Pentru aceasta este necesară aplicarea funcției inverse pe această valoare:

Ecuația 9. Funcția inversă de atenuare.

unde P0, n și d0 au fost determinate mai devreme (**popt**).

Distanțele față de AP-uri pot fi considerate razele unor cercuri pe care, teoretic, se află în spațiu dispozitivul de pe care se încearcă localizarea. Centrele acestor cercuri, adică pozițiile efective ale AP-urilor pe etaj, se pot afla în două moduri:

* lansând aplicația în modul debug și apăsând pe hartă: se vor afișa informații despre coordonata punctului, în pixeli. Aceasta presupune că utilizatorul deține informația privind amplasarea AP-ului în încăpere.
* generând harta semnalului pentru un anumit AP pe baza informațiilor colectate în perioada de antrenare și identificând punctul unde semnalul este cel mai puternic. Se poate apoi aplica prima metodă.

Având informațiile despre poziția AP-urilor în încăpere și distanțele teoretice ale dispozitivului mobil față de acestea se pot trasa grafic cercuri pentru o mai bună înțelegere a problemei expuse.

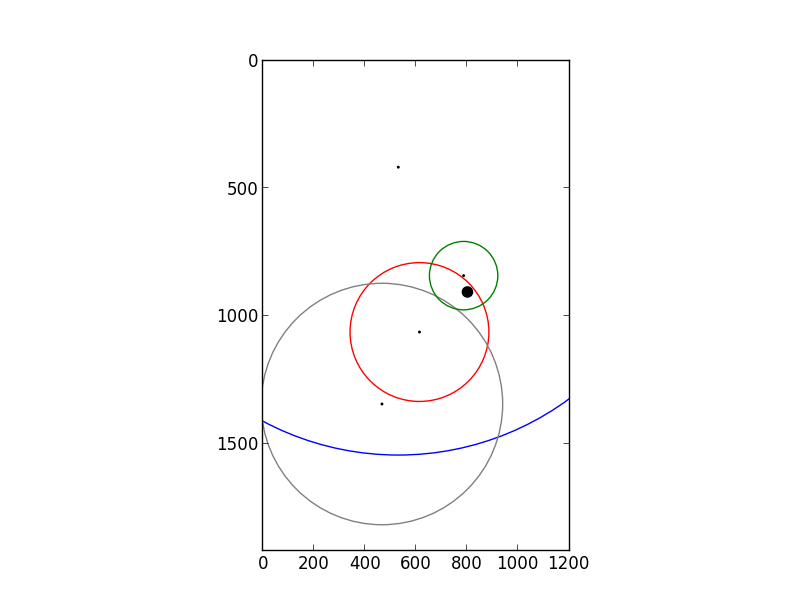


Fig. 9. Interpretarea grafică a trilaterației.

# În figura de mai sus, punctul negru îngroșat îl reprezintă poziția efectivă a dispozitivului ce se dorește localizat. Restul de puncte negre reprezintă poziția efectivă a AP-urilor în încăpere. Deși numărul AP-urilor este 4, am păstrat termenul *trilaterație* în favoarea termenului *multilaterație* pentru că este mai familiar. Analizând figura, se pare că acesta este un caz fericit. Cele 4 cercuri se intersectează aproape perfect într-o zonă foarte aproape de poziția efectivă a dispozitivului mobil, iar metoda de poziționare are astfel o eroare foarte mică.

Ceea ce nu place în figură este cercul descris de cel mai nordic AP (curba albastră). Acesta se îndepărtează foarte mult de poziția fizică, astfel că nu aduce nicio contribuție algoritmului. Aceste probleme sunt, totuși, inerente, deoarece curba atenuării semnalului este doar un fit logaritmic pe un set de date empiric. La valori mici ale semnalului (sub -60 dBm), cum este cazul pentru acest AP, panta curbei se micșorează, iar distanțele raportate pot diferi foarte mult de realitate. În acest caz, există două abordări:

* dacă numărul AP-urilor este 3, nu se poate face nimic pentru că nu există suficiente date pentru rezolvarea matematică a sistemului de ecuații
* dacă există mai mult de 3 AP-uri (fie **n** numărul lor) obținerea unei poziții poate fi urmată de o recalculare a distanțelor dintre aceasta și locul AP-urilor. Se pot apoi compara aceste noi distanțe obținute cu cele vechi, iar AP-ul care aduce cea mai mare eroare poate fi eliminat. Această procedură poate fi repetată de maxim **n – 3** ori, până când nu se mai obține nicio îmbunătățire (se poate ca încă de la primul pas să nu se îmbunătățească nimic). Un astfel de proces este exemplificat în secțiunea 4.2 Trilaterație unde sunt prezentate rezultatele obținute

//TODO Outlier speak

# 4. Rezultate

## 4.1 Analiza preliminară

Pe baza fișierelor JSON salvate în perioada de dezvoltare a aplicației am dorit o interpretare grafică a datelor colectate pas cu pas. În prima etapă am dorit să văd cum arată curba semnalului pentru toate AP-urile din încăpere, parcurgând etajul doar pe o axă. Fig. 10 arată graficul obținut, limbajul de programare folosit fiind Python cu modulul **matplotlib**.

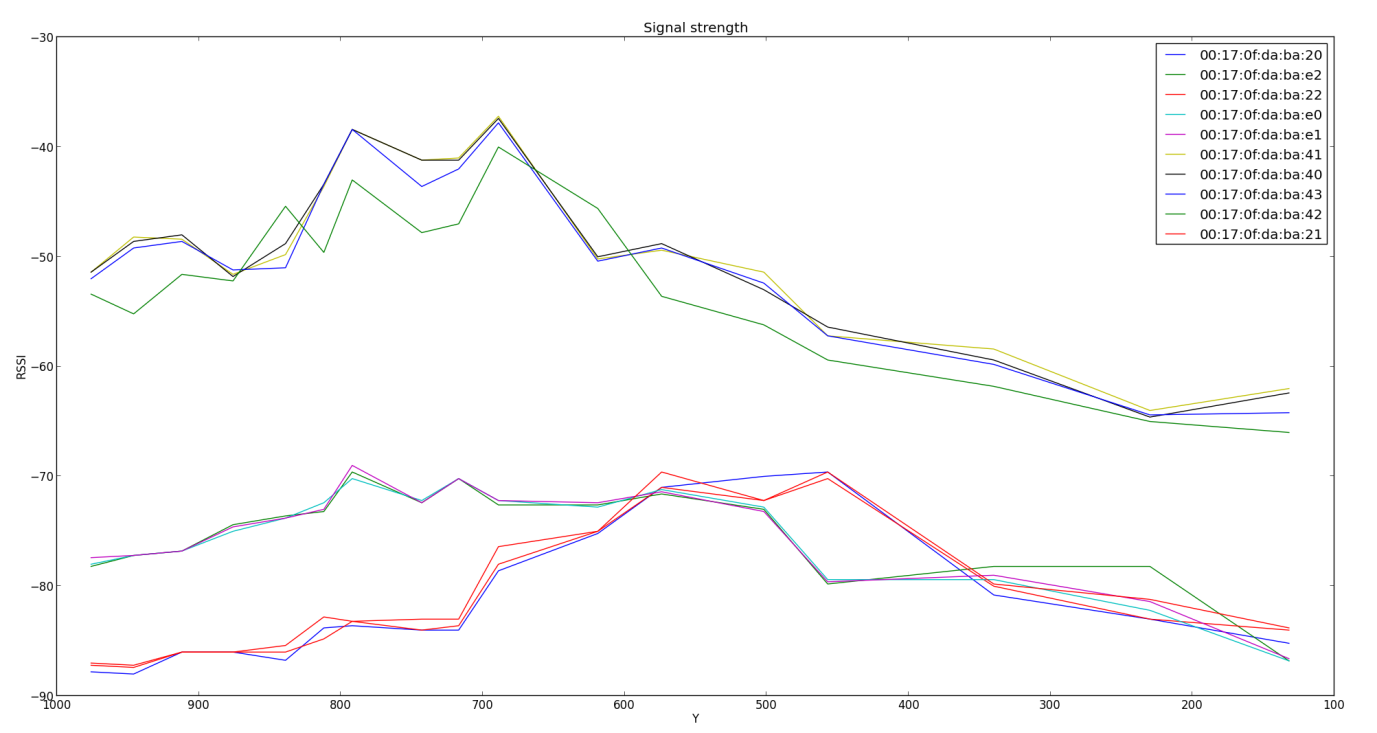


Fig. 10. Puterea semnalului pentru fiecare AP

în funcție de poziție, mergând pe cea mai lungă linie dreaptă a încăperii.

Se poate observa că, deși sunt afișate curbele pentru 10 adrese MAC, se disting 3 grupuri. Acest lucru nu este cel sperat, deoarece înseamnă că un singur AP deține mai multe adrese MAC (fiecare AP face broadcast la mai multe SSID-uri). Un alt neajuns îl constituie apropierea între curbele celor două grupuri din partea de jos a graficului. Este posibil ca această apropiere să introducă erori de poziționare.

În momentul acesta am decis să testez trivial aplicația cu doar câteva puncte de antrenament, iar rezultatele obținute au fost foarte proaste, în marea majoritate a cazurilor erorile depășind 10 metri. Un prim pas spre determinarea cauzei a fost identificarea pozițiilor fizice ale AP-urile în încăpere. După cum se poate observa în figura de mai sus, două dintre cele trei AP-uri au un semnal constant mai slab decât cel de-al treilea, acestuia din urmă fiindu-i cunoscută poziția întrucât se afla la vedere pe etaj. Ulterior am reușit să aflu că semnalul slab era cauzat de diferența de nivel întrucât cele două AP-uri se aflau pe cele două etaje adiacente, în aceeași poziție relativă la planul bidimensional ca cea a AP-ului de pe etajul unde îmi desfășuram activitatea.

Acestea fiind descoperite, am ales să nu folosesc semnalele colectate de la aceste AP-uri de pe alte nivele. Ceea ce mi-a întărit decizia a fost și generarea hărților de semnal pentru ele la distanță de o zi. Se observă în figura 11 că propagarea semnalului pentru un AP printr-o podea nu este deloc deterministică, de aceea aceste AP-uri nu trebuie luate în considerare pentru estimarea poziției.

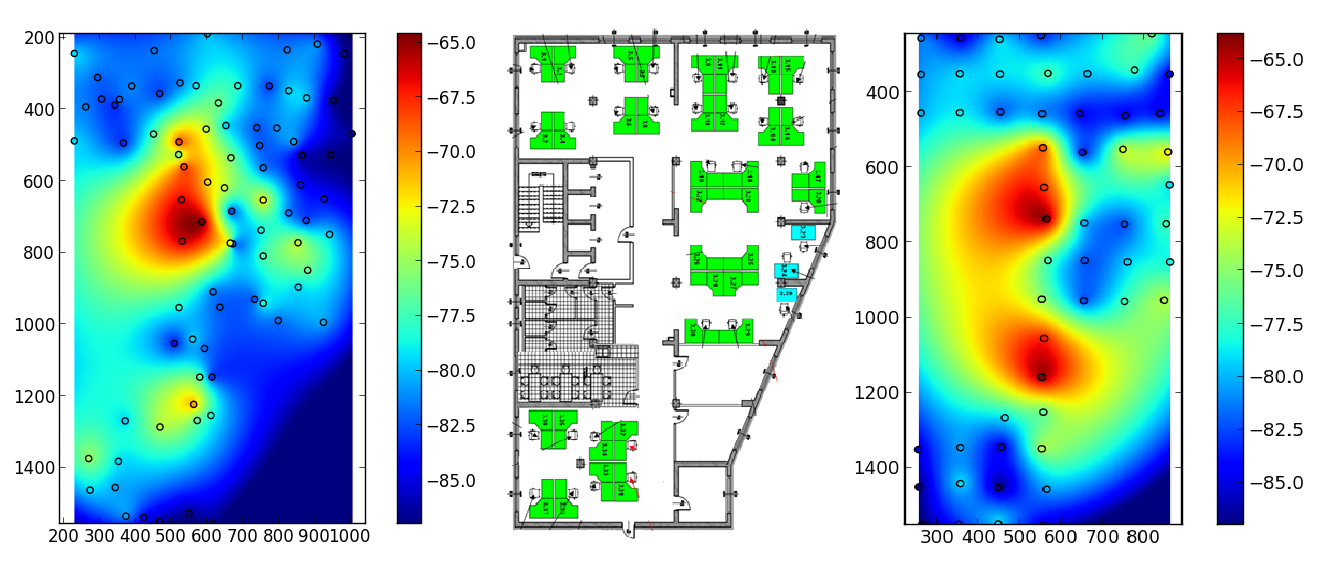


Fig. 11. Diferența între harțile de semnal pentru un singur AP la distanță de o zi

Aceste experimente au redus rând pe rând numărul AP-urilor de la 10 – câte BSSID-uri erau văzute pe aer – la 3 – un AP are mai multe BSSID-uri – și mai departe la unul singur întrucât AP-uri de pe etaje diferite introduc erori foarte mari de poziționare. În cazul acesta am decis să amplasez în încăpere încă 3 AP-uri pentru un total de 4. În cele ce urmează prezint harta semnalului pentru fiecare dintre acestea, precum și poziția fizică unde ele sunt amplasate (pozițiile fizice ale AP-urilor sunt marcate prin cercuri albastre pe harta încăperii).

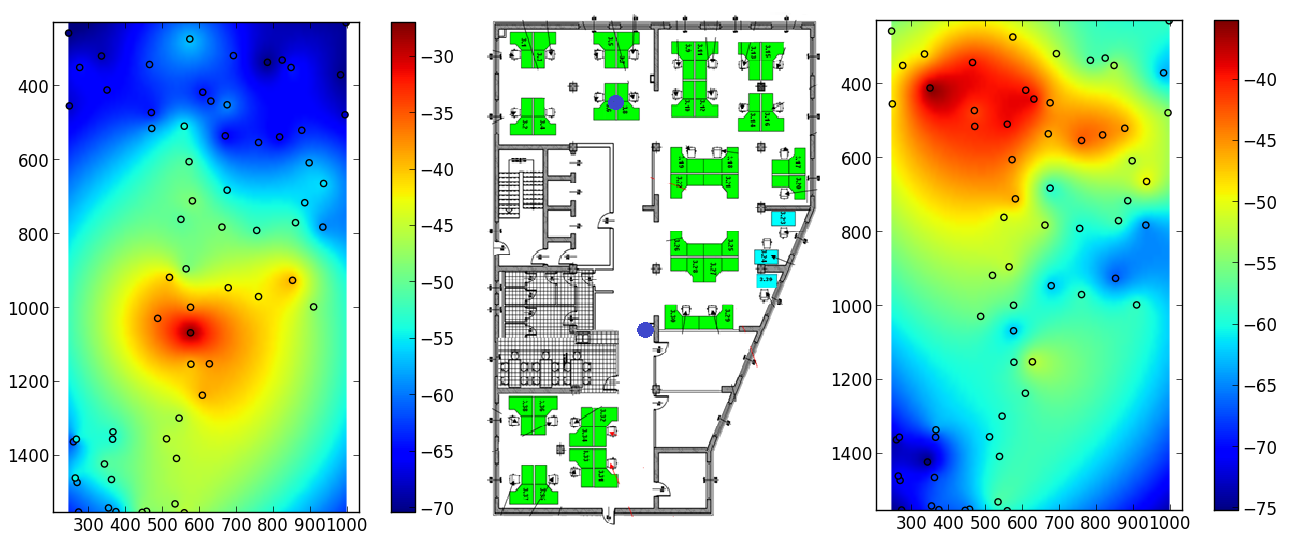


Fig. 12. Harta semnalului pentru AP-urile 1 și 2, raportată la dimensiunile încăperii.

**Notă**: Poziția fizică a unui AP, în caz că nu este cunoscută, poate fi estimată cu o precizie bună analizând harta semnalului generat de el și interpolând printre valori (după cum se poate observa mai sus, semnalul cel mai puternic marcat prin zonele roșii este identificat în proximitatea AP-urilor).

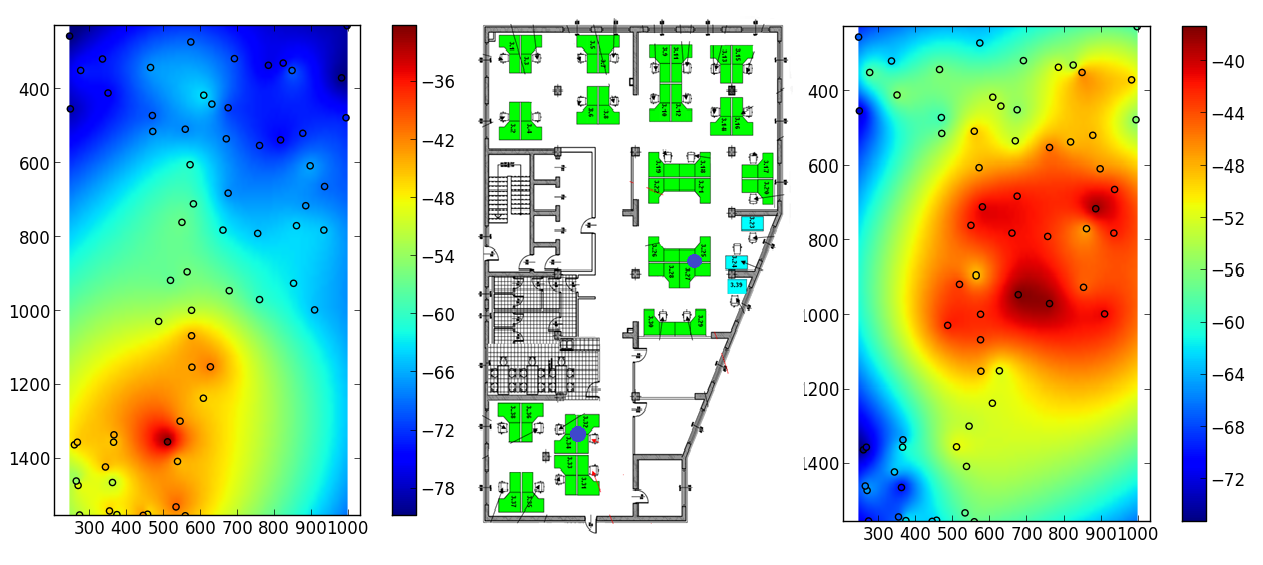


Fig. 13. Harta semnalului pentru AP-urile 3 și 4, raportată la dimensiunile încăperii

Hărțile de semnal au fost generate cu ajutorul funcției scipy.interpolate.Rbf(*\*args*) ce este capabilă să efectueze interpolare pe un set n-dimensional de puncte dispersate. În cazul de față, \*args a fost compus din:

* numpy.array([point.x for point in offlineData])
* numpy.array([point.y for point in offlineData])
* numpy.array([point.bssid[1..4].rssi for point in offlineData])
* function=’linear’ – interpolare liniară

## 4.2 Amprente radio

# Pentru determinarea erorilor de poziționare am ales un traseu prin încăpere pe care să-l parcurg pe rând cu cei 4 algoritmi folosiți: NN, kNN, WkNN, EWkNN. Pe acest traseu, mi-am ales 28 de puncte unde doream să mi se furnizeze locația. Pozițiile estimate de algoritm sunt reprezentate de cercurile albastre (acestea sunt afișate din aplicație când se execută click). Fiecare poziție estimată este legată de poziția actuală printr-o linie neagră.



Fig. 14. NN Fig. 15. kNN

# Pentru raportarea erorii algoritmilor am decis să calculez eroarea totală și să fac media aritmetică pe cele 28 de puncte. Am obținut următoarele:

* NN
  + 54,6 m eroare totală
  + 1,95 m eroare medie
* kNN
  + 48,3 m eroare totală
  + 1,73 m eroare medie

# joined_wknn_ewknn.png

Fig. 16. WkNN Fig. 17 EWkNN

# WkNN

* + 47,95 eroare totală
  + 1,71 m eroare medie
* EWkNN
  + 52,1 m eroare totală
  + 1,86 m eroare medie

În urma acestor rezultate, se pot trage următoarele concluzii (pentru mediul în care s-a încercat localizarea):

* NN este algoritmul ce oferă cea mai mare eroare cumulată de poziționare, fapt de altfel normal având în vedere simplitatea lui.
* kNN și WkNN se comportă similar, diferența rezultatelor celor doi algoritmi fiind foarte mică. Ambii algoritmi au fost rulați pentru k=3.
* EWkNN, în mod ciudat, se comportă mai rău decât anteriorii doi algoritmi, ducând mai degrabă la performanțe similare celui mai trivial dintre aceștia, NN. Aici problema poate apărea din setarea greșită a acelui prag de eroare despre care vorbeam în 3.5.1.4. De-a lungul implementării, experimentând cu mediul și cu diverse valori pentru acest prag, am ajuns la concluzia empirică că el ar trebui setat la valoarea , unde n reprezintă numărul de AP-uri găsite online. Practic, am dorit ca diferența medie între datele online și cele offline să fie de 20dBm / AP. S-a dovedit însă că, pentru datele din baza de date și pentru mediul în pricină, algoritmul găsește prea puține puncte pentru care să realizeze media coordonatelor, astfel că performanțele tind spre cele ale NN.

## 4.3 Trilaterație

Rezultatele acestei metode au fost determinate pe calculatorul personal cu ajutorul unor scripturi Python. Am ales să aleg un număr de 10 puncte de pe etaj pentru a raporta erorile măsurate. Pentru aceasta, pe lângă datele colectate în faza de antrenare, am mai colectat valorile semnalului în aceste 10 puncte pentru a putea încerca localizarea și din fața calculatorului, fără să mai am nevoie de dispozitivul mobil Android. În cele ce urmează voi prezenta pașii urmați pentru calcularea poziției și a erorii aferente pentru un singur punct (pentru o înțelegere mai bună a problemei, voi face referire la Fig. 11, acolo unde sunt prezentate grafic datele de intrare pentru acest punct), iar apoi voi raporta erorile pentru toate cele 10 puncte, precum și pe cea globală. Pentru restul de 9 puncte, figurile aferente se află în anexa 2 //TODO, aici fiind prezentate doar rezultatele.

După cum se observă în Fig. 11, pentru furnizarea poziției avem date de la 4 AP-uri. În pixeli, acestea au următoarele valori:

|  |  |  |
| --- | --- | --- |
| **x AP** | **y AP** | **Distanța față de AP** |
| 612 | 1063 | 272 |
| 529 | 417 | 1127 |
| 465 | 1345 | 473 |
| 785 | 842 | 134 |

Tabel 2. Date de intrare pentru trilaterație

Pe lângă datele de mai sus, mai este cunoscută și poziția efectivă a dispozitivului de la momentul colectării datelor **E -** **(800, 906).** Rezolvând sistemul de ecuații TODO prima soluție obținută este **S1 - (913, 1196).** Având 4 AP-uri, se poate ajunge la o soluție mai apropiată de realitate eliminând AP-ul care introduce cea mai mare eroare. Astfel, se pot recalcula distanțele:

Cea mai mare eroare o oferă AP-ul 2, astfel încât următoarea rulare a algoritmului se va face pe un set de date redus la doar 3 AP-uri. Soluția obținută de data aceasta este **S2 - (841, 991)**,într-adevăr mai bună decât cea inițială. Mai departe nu se poate optimiza întrucât cu mai puțin de 3 AP-uri nu se poate rezolva sistemul de ecuații. Avem așadar, pentru acest punct, eroarea de poziționare este de (ținând cont de transformarea 43 pixeli ecran = 1 metru real):

Pentru restul de 9 puncte analizate pozițiile AP-urilor rămân neschimbate. În tabelul următor voi prezenta datele obținute pentru aceste puncte, cu mențiunea că imaginile aferente nr. crt. se găsesc în anexa //TODO.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. crt. | Poziția efectivă | Poziția estimată S1 | Poziția estimată S2 | Eroarea |
| 1 | (501, 453) | (620, 409) | **(454, 518)** | 80.2px, 1.85m |
| 2 | (583, 457) | **(509, 616)** | --- | 175.4px, 4m |
| 3 | (680, 437) | **(678, 417)** | --- | 20.1px, 0.5m |
| 4 | (774, 446) | (771, 387) | **(746, 480)** | 44px, 1m |
| 5 | (853, 480) | **(819, 414)** | --- | 74.2, 1.7m |
| 6 | (677, 538) | (476, 623) | **(779, 364)** | 201.7px, 4.65m |
| 7 | (827, 677) | (653, 882) | **(890, 758)** | 102.6px, 2.4m |
| 8 | (687, 671) | (1065, 978) | **(951, 829)** | 307.7px, 7.1m |
| 9 | (664, 893) | **(481, 999)** | --- | 211.5px, 4.9m |

Tabel 3. Rezultatele trilaterației

Totalul erorilor este **28.1m**, ceea ce înseamnă o eroare medie de 3.12 m / punct solicitat. De remarcat este faptul că această metodă este foarte imprevizibilă, erorile de poziționare variind foarte mult de la punct la punct (între minimul de 0.5m și maximul de 7.1m). De asemenea, pentru 60% dintre puncte rezultatul s-a putut îmbunătăți prin tehnica de eliminare a unui AP.

# 5. Concluzii

## 5.1 Comparație

Următorul tabel sumarizează diferențele și similaritățile dintre cele două metode prezentate în lucrarea de față, amprente radio și trilaterație.

|  |  |  |
| --- | --- | --- |
| **Caracteristici** | **Amprente radio** | **Trilaterație** |
| **Harta încăperii** | Necesară | Necesară |
| **Număr AP-uri** | Nu există o limită inferioară | Minim 3 |
| **Antrenare (offline)** | Necesară | Nu este necesară, atât timp cât se poate aplica FSPL (camere libere) sau se cunoaște exponentul de atenuare al mediului |
| **Cunoașterea poziției AP-urilor** | Nu este necesară | Necesară |
| **Dependențe** | Metoda nu depinde de aplicarea altor metode mai întâi | Poate fi necesară antrenarea offline pentru determinarea funcției de atenuare a semnalului |
| **Algoritmi folosiți** | Derivați din Nearest Neighbour | Matematici pentru rezolvarea sistemelor de ecuații ale cercurilor descrise |
| **Complexitate** | Mare, dezvoltarea de software pentru antrenare | Medie, dezvoltarea algoritmilor |

Tabel 4. Comparație între amprente radio și trilaterație

Lucrarea de față își dorește să prezinte un model concret de localizare în interior folosind amprente radio (metodă empirică), dar totodată lasă loc abordării altor metode pe baza datelor colectate de-a lungul implementării. Aplicația dezvoltată în Android poate fi folosită pentru diverse încăperi, singurul aspect ce trebuie modificat fiind harta încăperii (un nou utilizator va putea fi capabil să încarce propria hartă după bunul plac).

bibliografie

[1] Ozkan Katircioglu, Hasan Isel, Osman Ceylan, Firat Taraktas, H. Bulent Yagci, Comparing Ray Tracing, Free Space Path Loss and Logarithmic Distance Path Loss Models in Success of Indoor Localization with RSSI, 19th Telecommunications forum TELFOR 2011, Serbia, Belgrade, November 22-24, 2011

[2] Guoqiang Mao, Brian D.O. Anderson, Barıs Fidan, Path loss exponent estimation for wireless sensor

network localization, 29 November 2006

[3] Matthew Gast, 802.11 Wireless Networks: The Definitive Guide, Second Edition, 2005

[4] Paramvir Bahl and Venkata N. Padmanabhan, RADAR: An In-Building RF-based User Location and Tracking System, Microsoft Research

[5] Moustafa Youssef and Ashok Agrawala, The Horus WLAN Location Determination System, Department of Computer Science, University of Maryland

[6] Aswin N Raghavan, Harini Ananthapadmanaban, Manimaran S Sivamurugan, Balaraman Ravindran, Accurate Mobile Robot Localization in indoor environments using Bluetooth

[7] Scott Y. Seidel, Student Member, IEEE, and Theodore S. Rappaport, Senior Member, IEEE, 914 MHz Path Loss Prediction Models for Indoor Wireless Communications in Multifloored Buildings

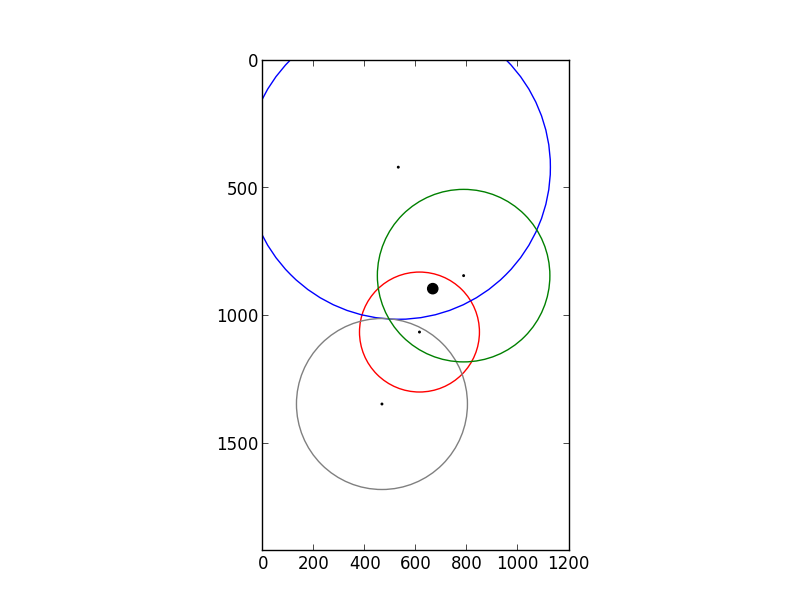
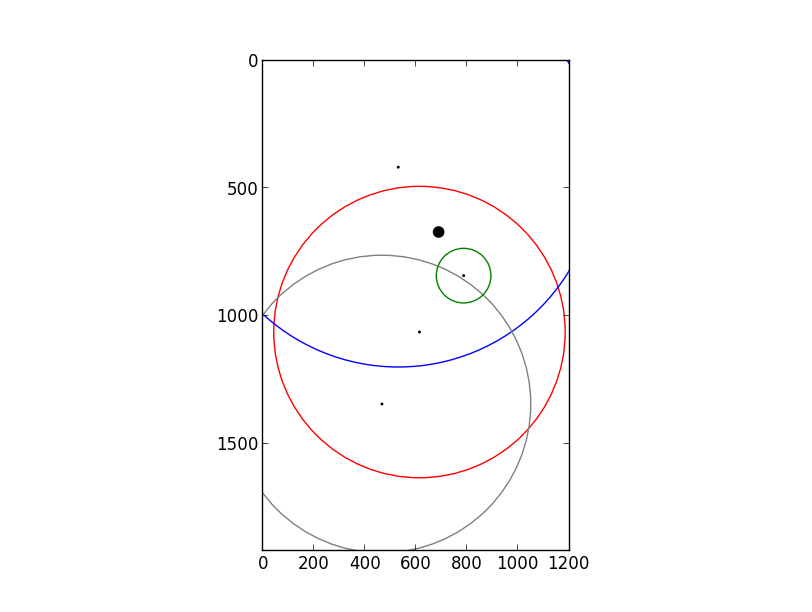
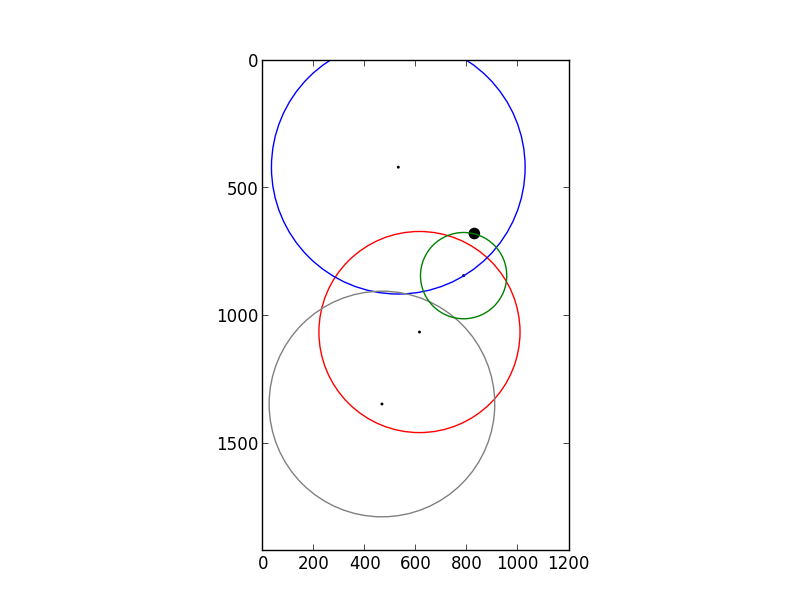
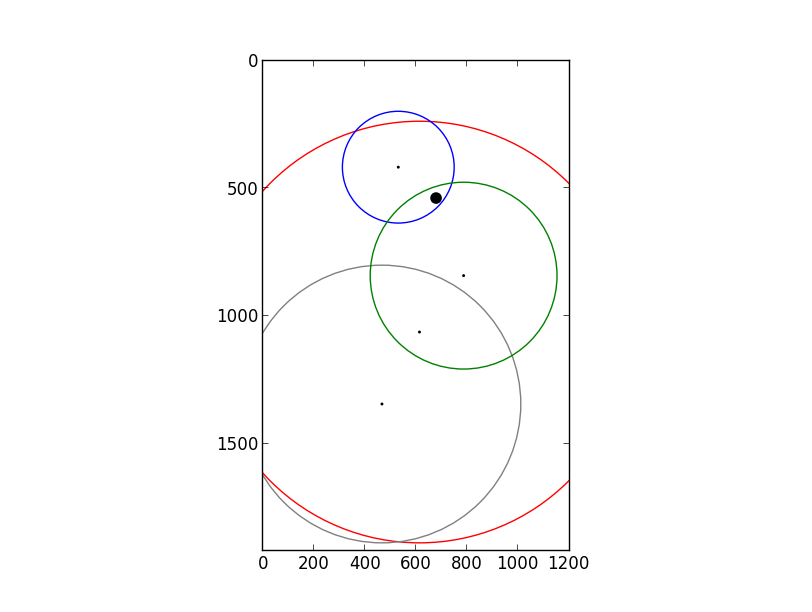
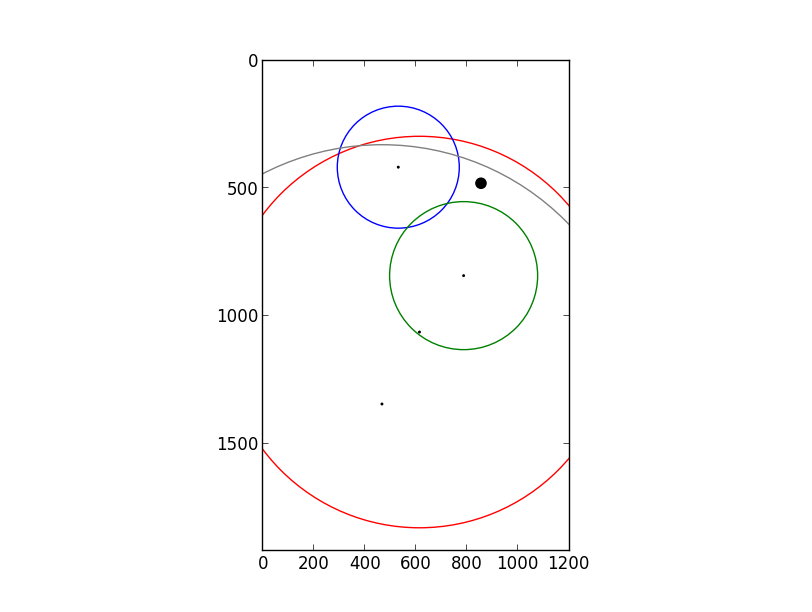
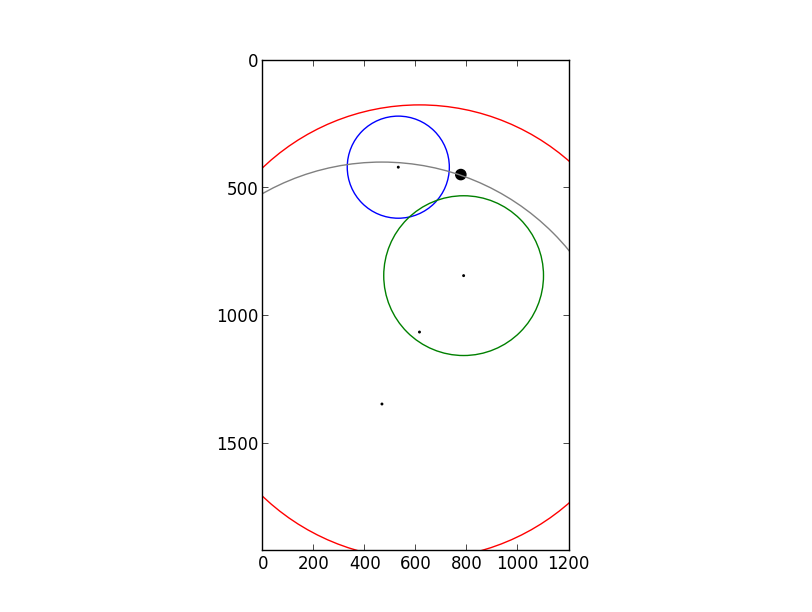
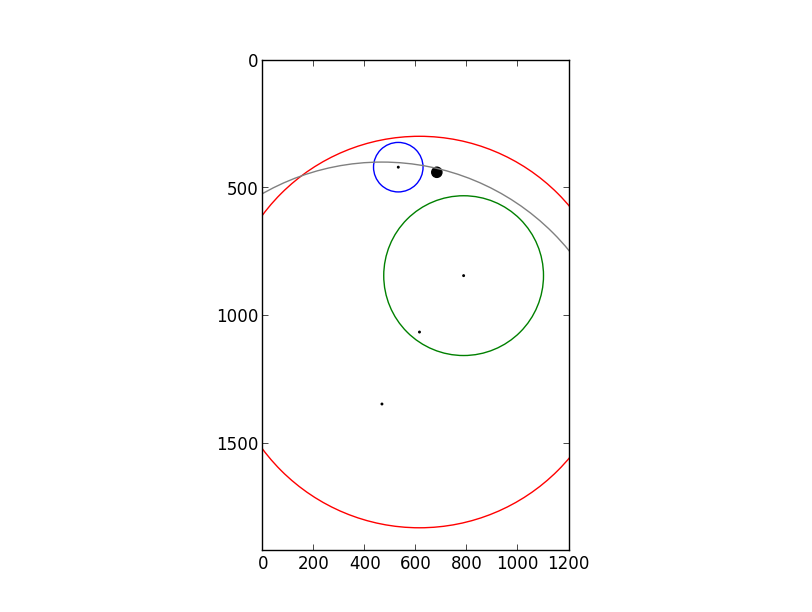
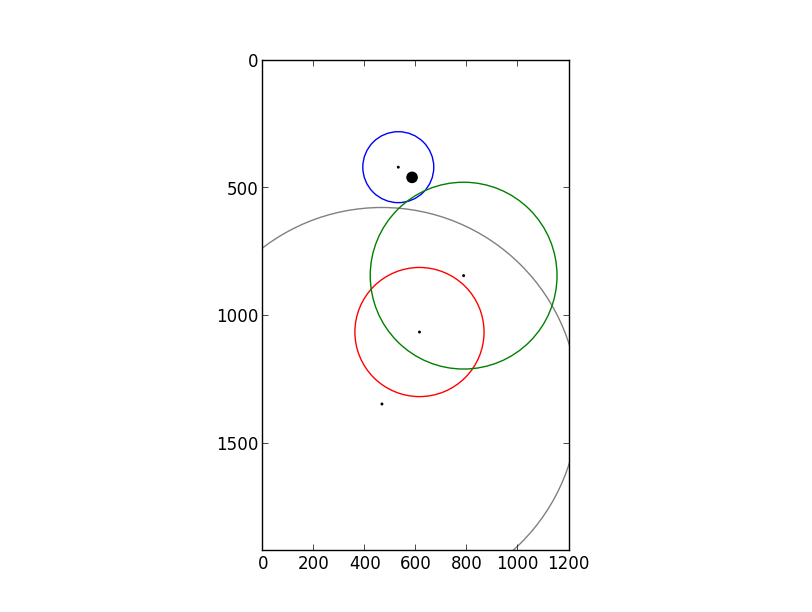
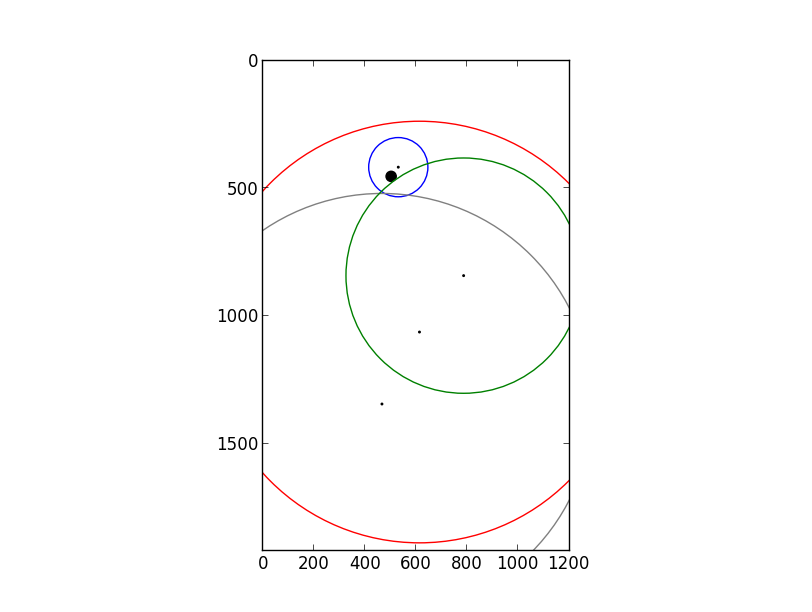
[8] Beomju Shin, lung Ho Lee, Taikjin Lee, Hyung Seok Kim, Enhanced Weighted K-Nearest Neighbor Algorithm for Indoor Wi-Fi Positioning Systems

[9] Copy of Google’s submission today to several national data protection authorities on vehicle-based collection of wifi data for use in Google location based services, Raphael Leiteritz, Product Manager, Google Inc., 27th April, 2010

[10] „Apple M7” Wikipedia. Wikimedia Foundation, December 18, 2013

**ANEXA A**

Trilaterația în imagini (în ordinea nr. crt. din tabelul 3)



**ANEXA B**

Cod și screenshoturi pentru aplicația Android

**MainActivity.java**

package ro.pub.cs.radar.gui;

imports [...]

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Spinner spinner1 = (Spinner) findViewById(R.id.spinner1);

// Create an ArrayAdapter using the string array and a default spinner

// layout

ArrayAdapter<CharSequence> adapter1 = ArrayAdapter.createFromResource(this,

R.array.algorithms, android.R.layout.simple\_spinner\_item);

// Specify the layout to use when the list of choices appears

adapter1.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);

// Apply the adapter to the spinner

spinner1.setAdapter(adapter1);

spinner1.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {

WhereAmIActivity.algorithm = parent.getItemAtPosition(pos).toString();

}

@Override

public void onNothingSelected(AdapterView<?> arg0) {

}

});

Spinner spinner2 = (Spinner) findViewById(R.id.spinner2);

ArrayAdapter<CharSequence> adapter2 = ArrayAdapter.createFromResource(this,

R.array.methods, android.R.layout.simple\_spinner\_item);

adapter2.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);

spinner2.setAdapter(adapter2);

Button APs = (Button) findViewById(R.id.button1);

APs.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(getApplicationContext(), ScanActivity.class);

startActivity(intent);

}

});

Button chooseMap = (Button) findViewById(R.id.button2);

chooseMap.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

FileChooserDialogFragment df = new FileChooserDialogFragment(Constants.BMP\_EXT);

df.show(getFragmentManager(), "dialog");

}

});

Button map = (Button) findViewById(R.id.button3);

map.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(getApplicationContext(), MapActivity.class);

startActivity(intent);

}

});

Button chooseDb = (Button) findViewById(R.id.button4);

chooseDb.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

FileChooserDialogFragment df = new FileChooserDialogFragment(Constants.JSON\_EXT);

df.show(getFragmentManager(), "dialog");

}

});

Button find = (Button) findViewById(R.id.button5);

find.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

if (Parser.points == null) {

Toast.makeText(getApplicationContext(), "Please select a database first!",

Toast.LENGTH\_SHORT).show();

} else {

Intent intent = new Intent(getApplicationContext(), WhereAmIActivity.class);

startActivity(intent);

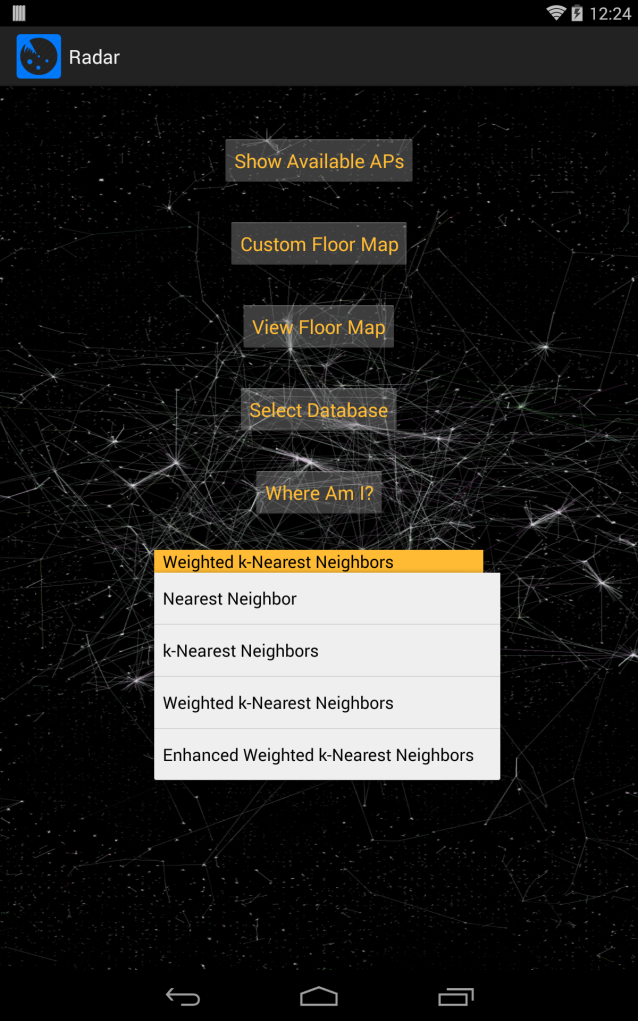
}

}

});

}

}



**ScanActivity.java**

package ro.pub.cs.radar.gui;

imports [...]

public class ScanActivity extends Activity {

private ListView lv;

private ArrayAdapter<String> adapter;

private WifiManager mainWifi;

private WifiReceiver receiverWifi;

private List<ScanResult> wifiList;

private List<String> SSIDs = new ArrayList<String>();

private List<String> attributes = new ArrayList<String>();

private ArrayList<String> history = new ArrayList<String>();

private Handler mHandler;

private int mInterval = 0;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_scan);

lv = (ListView) findViewById(R.id.listView);

adapter = new ArrayAdapter<String>(this, android.R.layout.simple\_list\_item\_1,

android.R.id.text1, SSIDs);

lv.setAdapter(adapter);

lv.setOnItemClickListener(new OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

AlertDialog.Builder adb = new AlertDialog.Builder(ScanActivity.this);

adb.setTitle("Network Info");

adb.setMessage(attributes.get(position));

adb.setPositiveButton("Ok", null);

adb.show();

}

});

mainWifi = (WifiManager) getSystemService(Context.WIFI\_SERVICE);

receiverWifi = new WifiReceiver();

registerReceiver(receiverWifi, new IntentFilter(WifiManager.SCAN\_RESULTS\_AVAILABLE\_ACTION));

mainWifi.startScan();

// refresh information

mHandler = new Handler();

startRepeatingTask();

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

menu.add(0, 0, 0, "Refresh");

return super.onCreateOptionsMenu(menu);

}

@Override

public boolean onMenuItemSelected(int featureId, MenuItem item) {

mainWifi.startScan();

return super.onMenuItemSelected(featureId, item);

}

@Override

protected void onPause() {

unregisterReceiver(receiverWifi);

super.onPause();

}

@Override

protected void onResume() {

registerReceiver(receiverWifi, new IntentFilter(WifiManager.SCAN\_RESULTS\_AVAILABLE\_ACTION));

super.onResume();

}

Runnable mStatusChecker = new Runnable() {

@Override

public void run() {

mainWifi.startScan();

mHandler.postDelayed(mStatusChecker, mInterval);

}

};

void startRepeatingTask() {

mStatusChecker.run();

}

class WifiReceiver extends BroadcastReceiver {

@SuppressLint("SimpleDateFormat")

public void onReceive(Context c, Intent intent) {

// Dismiss all previous data

SSIDs.clear();

attributes.clear();

Calendar cal = Calendar.getInstance();

cal.getTime();

SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss:SSS");

wifiList = mainWifi.getScanResults();

for (int i = 0; i < wifiList.size(); i++) {

String time = sdf.format(cal.getTime());

history.add(time + " " + wifiList.get(i).SSID + " " + wifiList.get(i).BSSID + " "

+ wifiList.get(i).level);

SSIDs.add(wifiList.get(i).SSID);

attributes.add(wifiList.get(i).toString().replace(", ", "\n"));

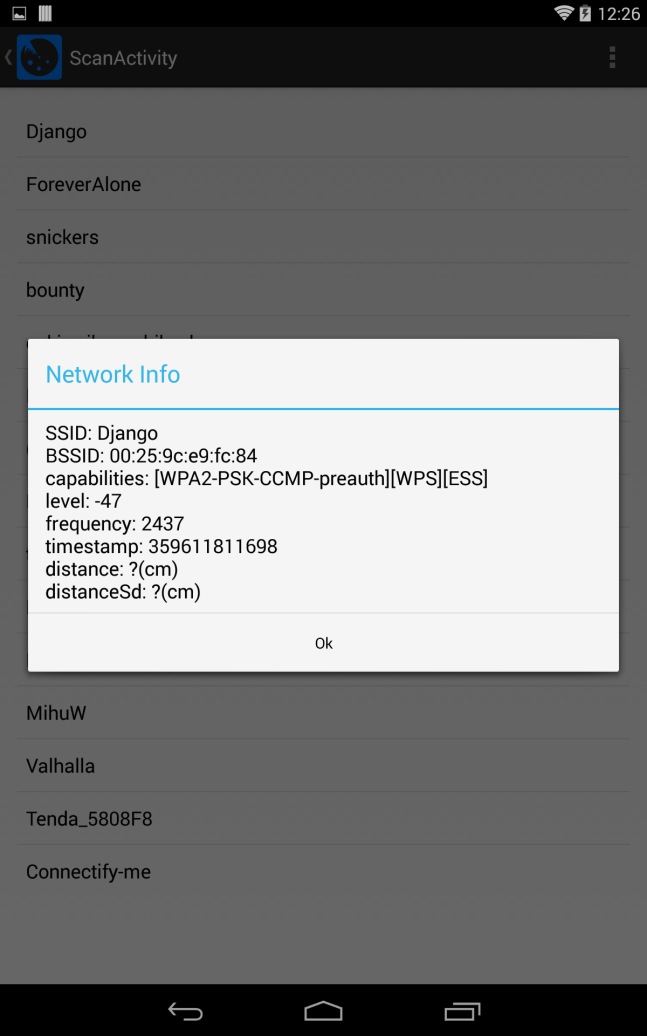
}

adapter.notifyDataSetChanged();

}

}

}



**FileChooserDialogFragment.java**

package ro.pub.cs.radar.gui;

imports [...]

public class FileChooserDialogFragment extends DialogFragment {

private String[] mFileList;

private File mPath = new File(Environment.getExternalStorageDirectory().toString());

private String mChosenFile;

private String fileType;

public FileChooserDialogFragment(String fileType) {

this.fileType = fileType;

}

private void loadFileList() {

if (mPath.exists()) {

FilenameFilter filter = new FilenameFilter() {

@Override

public boolean accept(File dir, String filename) {

return filename.contains(fileType);

}

};

mFileList = mPath.list(filter);

} else {

mFileList = new String[0];

}

}

@Override

public Dialog onCreateDialog(Bundle savedInstanceState) {

loadFileList();

AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

builder.setTitle("Choose " + fileType.toUpperCase(Locale.getDefault()) + " file");

if (mFileList == null) {

return builder.create();

}

builder.setItems(mFileList, new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

mChosenFile = mFileList[which];

if (fileType.equals(Constants.JSON\_EXT)) {

Parser parser = new Parser(mChosenFile);

try {

parser.execute();

} catch (IOException e) {

e.printStackTrace();

}

} else if (fileType.equals(Constants.BMP\_EXT)) {

MapView.customMap = mPath + "/" + mChosenFile;

Constants.FSL = "";

}

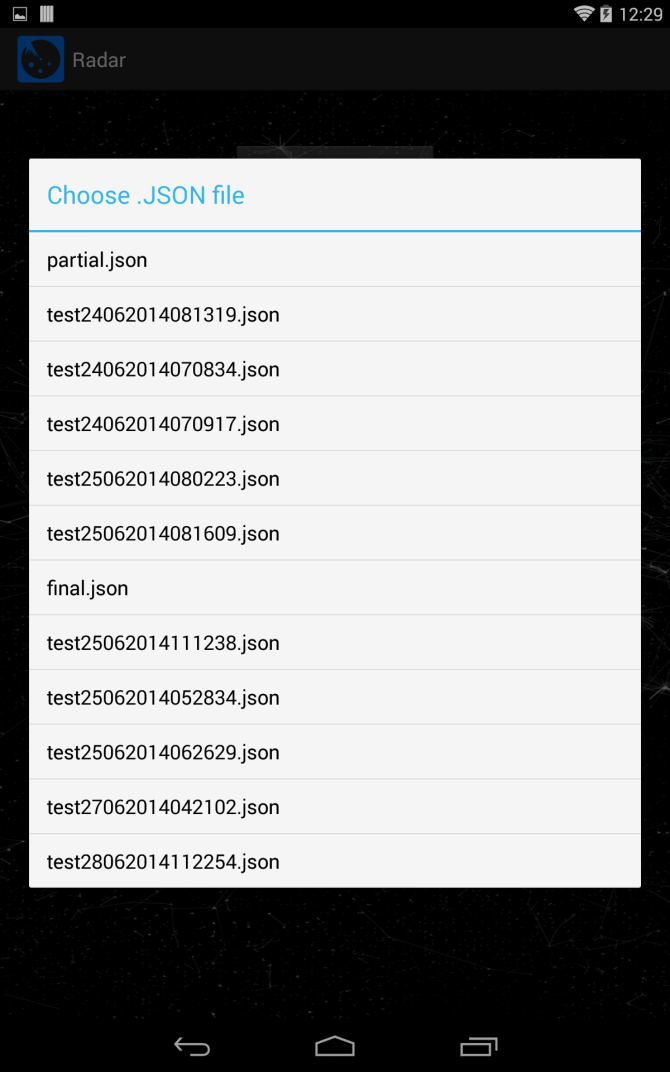
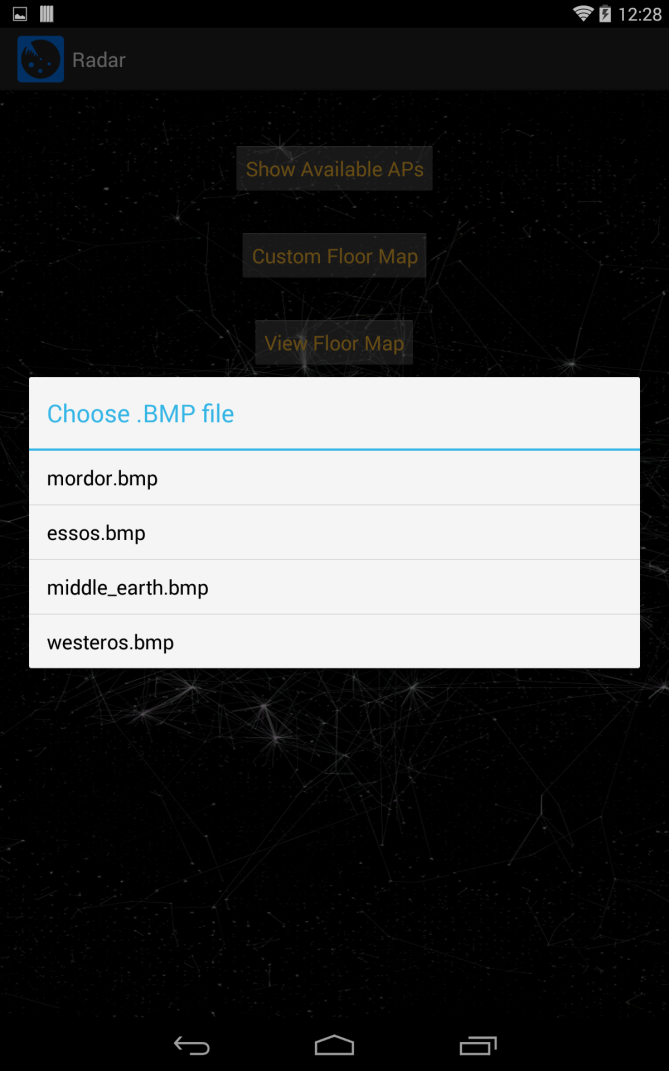
}

});

return builder.show();

}

}



**Algorithms.java**

package ro.pub.cs.radar.positioning;

imports [...]

public class Algorithms {

private HashMap<String, Integer> onlineData;

private ArrayList<AveragePointData> offlineData;

private TreeMap<Double, PointF> sortedPointsByDistance;

public Algorithms(HashMap<String, Integer> onlineData, ArrayList<AveragePointData> offlineData) {

this.onlineData = onlineData;

this.offlineData = offlineData;

this.sortedPointsByDistance = new TreeMap<Double, PointF>();

this.sortPointsByDistance();

}

/\*\*\*

\* Sort the offline data with regard to the measured RSSI.

\*/

private void sortPointsByDistance() {

double distance;

for (AveragePointData apd : offlineData) {

distance = Common.euclideanDistance(apd.getData(), onlineData);

if (distance != 0) {

sortedPointsByDistance.put(distance, new PointF(apd.getX(), apd.getY()));

}

}

}

/\*\*\*

\* Nearest Neighbor algorithm. Return the point that minimizes the distance.

\*/

public PointF NN() {

try {

return sortedPointsByDistance.get(sortedPointsByDistance.firstKey());

} catch (NoSuchElementException nsee) {

return null;

}

}

/\*\*\*

\* k-Nearest Neighbors algorithm. Return the averaged coordinates from the

\* closest k points.

\*/

public PointF KNN() {

int k = Constants.k;

float sumX = 0, sumY = 0;

for (int i = 0; i < k; i++) {

Map.Entry<Double, PointF> e = sortedPointsByDistance.pollFirstEntry();

if (e == null) {

return null;

}

sumX += e.getValue().x;

sumY += e.getValue().y;

}

return new PointF(sumX / k, sumY / k);

}

/\*\*\*

\* Weighted k-Nearest Neighbors algorithm. Return the averaged coordinates

\* from the closest k points, but weigh them first according to the sorting.

\*/

public PointF WKNN() {

int k = Constants.k;

float sumX = 0, sumY = 0;

float denominator = 0;

for (int i = 0; i < k; i++) {

Map.Entry<Double, PointF> e = sortedPointsByDistance.pollFirstEntry();

if (e == null) {

return null;

}

sumX += e.getValue().x / e.getKey();

sumY += e.getValue().y / e.getKey();

denominator += 1.0 / e.getKey();

}

return new PointF(sumX / denominator, sumY / denominator);

}

/\*\*\*

\* Enhanced Weighted k-Nearest Neighbors algorithm. Similar to WKNN, but k

\* changes based on a threshold value for distance. The threshold is 20 \*

\* sqrt(number of offline points). This means that, in average, any offline

\* measured data that differs with 20 dBm from the online should count for

\* estimating position.

\*/

public PointF EWKNN() {

float sumX = 0, sumY = 0;

float denominator = 0;

Map.Entry<Double, PointF> e = null;

// return at least the closest point

do {

e = sortedPointsByDistance.pollFirstEntry();

if (e == null) {

break;

}

sumX += e.getValue().x / e.getKey();

sumY += e.getValue().y / e.getKey();

denominator += 1.0 / e.getKey();

} while (e.getKey() <= Constants.threshold\_factor

\* Math.sqrt(sortedPointsByDistance.size()));

if (denominator != 0) {

return new PointF(sumX / denominator, sumY / denominator);

} else {

return null;

}

}

}

**WhereAmIActivity.java**

package ro.pub.cs.radar.gui;

imports [...]

public class WhereAmIActivity extends Activity {

public static String algorithm = "Nearest Neighbor";

private HashMap<String, Integer> onlineData;

private ArrayList<AveragePointData> offlineData;

private MapView mapView;

private int mInterval = 5000; // update position every 5 seconds

private Handler mHandler;

private float lastX, lastY;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_map);

FrameLayout fl = (FrameLayout) findViewById(R.id.mapLayout);

this.mapView = new MapView(this, this);

fl.addView(this.mapView);

this.mHandler = new Handler();

this.startRepeatingTask();

}

@Override

protected void onDestroy() {

super.onDestroy();

stopRepeatingTask();

}

// Get data from current location

private void getCurrentData() {

Collector collector = new Collector(this);

collector.start();

try {

collector.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

private Runnable picasso = new Runnable() {

@Override

public void run() {

getCurrentData();

mHandler.postDelayed(picasso, mInterval);

}

};

private void startRepeatingTask() {

picasso.run();

}

private void stopRepeatingTask() {

mHandler.removeCallbacks(picasso);

}

private void drawPosition() {

Algorithms a = new Algorithms(onlineData, offlineData);

PointF position = null;

if (algorithm.startsWith("N")) {

position = a.NN();

} else if (algorithm.startsWith("k")) {

position = a.KNN();

} else if (algorithm.startsWith("W")) {

position = a.WKNN();

} else if (algorithm.startsWith("E")) {

position = a.EWKNN();

}

if (position != null) {

Canvas canvas = new Canvas(this.mapView.getBitmap());

Paint paint = new Paint();

paint.setColor(Color.BLUE);

canvas.drawCircle(position.x, position.y, 13, paint);

lastX = position.x;

lastY = position.y;

mapView.invalidate();

} else {

Toast.makeText(getApplicationContext(), "Not enough data..", Toast.LENGTH\_SHORT).show();

}

}

public void setOnlineData(HashMap<String, Integer> onlineData) {

this.onlineData = onlineData;

Log.v("POS", this.onlineData.toString());

setOfflineData();

drawPosition();

}

private void setOfflineData() {

offlineData = new ArrayList<AveragePointData>();

for (int i = 0; i < Parser.points.size(); i++) {

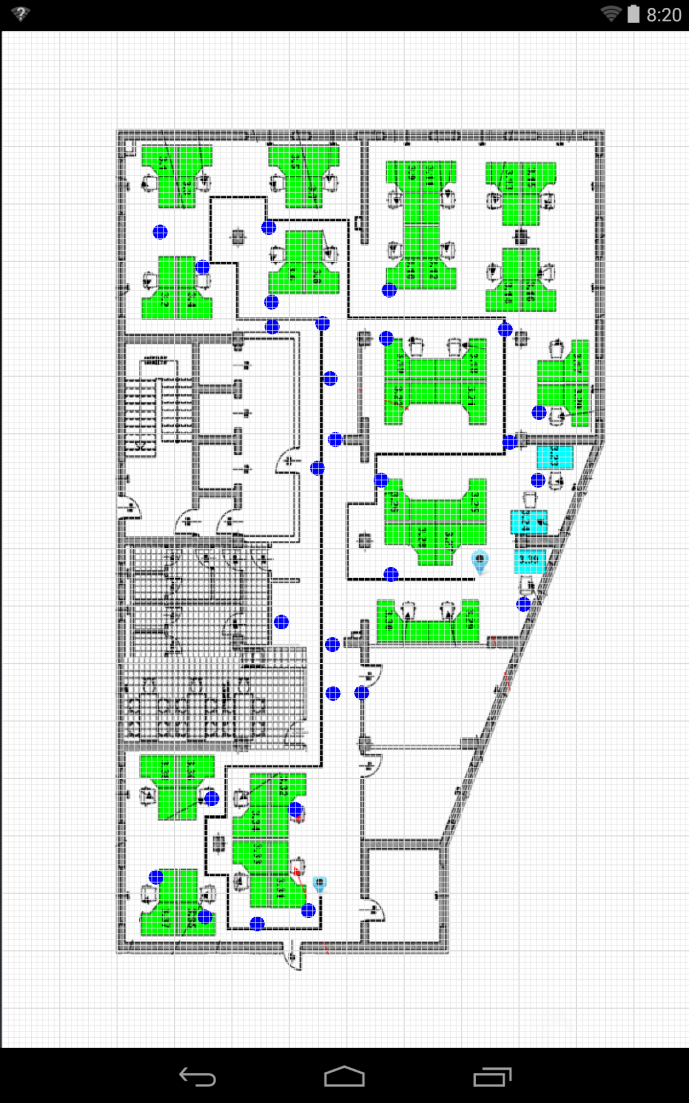
PointData pd = Parser.points.get(i);

offlineData.add(new AveragePointData(pd.getX(), pd.getY(), pd.getAverage()));

}

}

}



1. http://en.wikipedia.org/wiki/IBeacon [↑](#footnote-ref-2)
2. https://source.android.com/source/downloading.html [↑](#footnote-ref-3)
3. http://www.radiotap.org/defined-fields [↑](#footnote-ref-4)
4. http://spectrum.ieee.org/telecom/wireless/new-indoor-navigation-technologies-work-where-gps-cant [↑](#footnote-ref-5)
5. http://googleenterprise.blogspot.com/2012/10/two-new-ways-to-add-location.html [↑](#footnote-ref-6)
6. https://support.google.com/maps/answer/1725632?hl=en [↑](#footnote-ref-7)
7. http://developer.android.com/reference/android/net/wifi/WifiManager.html [↑](#footnote-ref-8)
8. http://developer.android.com/reference/android/app/DialogFragment.html [↑](#footnote-ref-9)
9. http://developer.android.com/reference/android/graphics/Bitmap.html [↑](#footnote-ref-10)
10. http://developer.android.com/reference/android/util/JsonReader.html [↑](#footnote-ref-11)
11. http://www.numpy.org/ [↑](#footnote-ref-12)
12. http://www.scipy.org/ [↑](#footnote-ref-13)
13. http://matplotlib.org/ [↑](#footnote-ref-14)
14. http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve\_fit.html [↑](#footnote-ref-15)