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

**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

1. 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 deja 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 ne-am aflat într-un oraș necunoscut și am căutat poziția noastră pe hartă folosind Google Maps? Sau de câte ori 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 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.

2. Studiul literaturii din domeniu

//TODO

**2.1 Elemente teoretice generale**

**2.1.1 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.

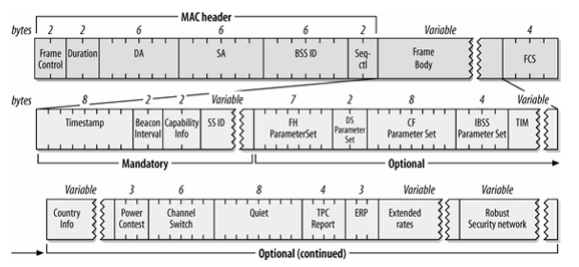


Fig. 1. Componența unui cadru Beacon [4]

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 [4]. Întrebarea ce apare aici este dacă dispozitivul Android de antrenament va putea recepționa toate aceste beacon-uri. Această întrebare primește un răspuns, cel puțin parțial, în secțiunea 3.1.

**2.1.2 Radiotap headers**

Radiotap este standardul de facto pentru recepționarea și injectarea cadrelor 802.11. Mai multe sisteme de operare suportă această tehnologie, printre care: FreeBSD, Linux, NetBSD, OpenBSD. Android fiind un sistem de operare bazat pe kernelul Linux, dispune și el de această tehnologie (needs citation).

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[[1]](#footnote-2), 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 în secțiunea TODO.

**2.2 Articole în domeniu**

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 [1]. 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 din baza de date 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 următoare.

De menționat este și [2], poate cel mai reprezentativ articol pentru a doua categorie de metode de localizare, cele probabilistice.

2.3 Rezultate

În [3] 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.

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.

Eroarea de poziționare a fiecărui algoritm

|  |  |
| --- | --- |
| Algoritm | Eroarea de poziționare |
| NN | 100% |
| KNN | 70% |
| WKNN | 50% |
| EWKNN | 37% |

Date fiind aceste rezultate, lucrarea de față își propune folosirea algoritmului EWKNN pentru determinarea locației, întrucât oferă cea mai mică eroare de poziționare, iar complexitatea algoritmului nu reprezintă o problemă (echipamentul folosit va fi un telefon mobil Android ce dispune de 4 nuclee).

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.

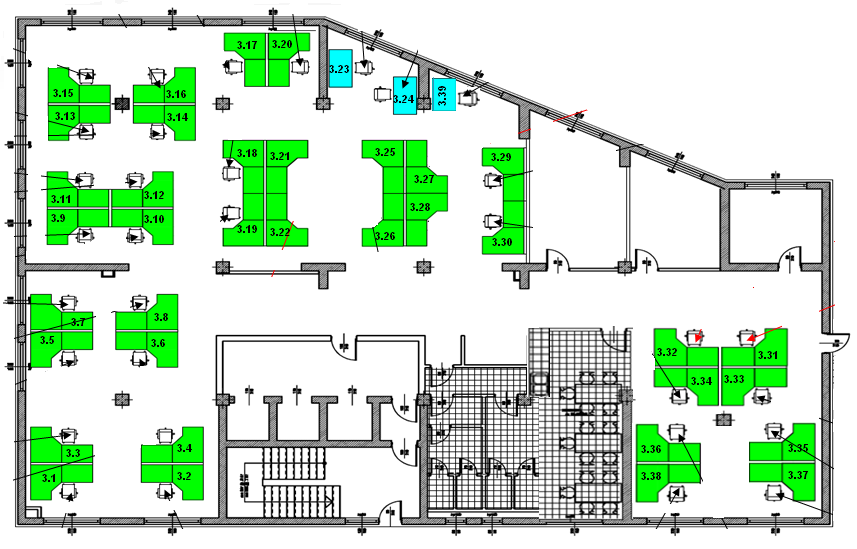


Fig. 1. Harta încăperii

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**

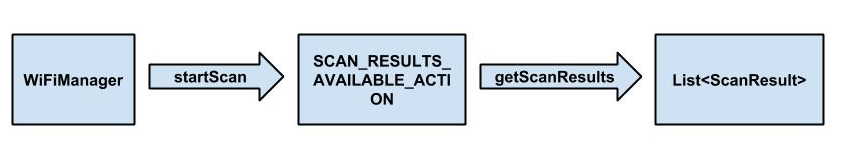
Preluarea datelor despre mediul Wi-Fi pe un dizpozitiv Android se face cu ușurință prin intermediul API-ului expus. Sistemul expune un serviciu numit **WiFiManager[[2]](#footnote-3)** ce se ocupă de toate aspectele conectivității 802.11. Obținerea unei instanțe a acestui serviciu se efectuează prin apelul **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 AP-urilor. 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.

**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**[[3]](#footnote-4)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**[[4]](#footnote-5)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 10 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).

O dată creat acest fișier, în perioada de dezvoltare se pot analiza diverse caracteristici ale mediului. Cu ajutorul unor scripturi Python am obținut Figurile 2 și 3.

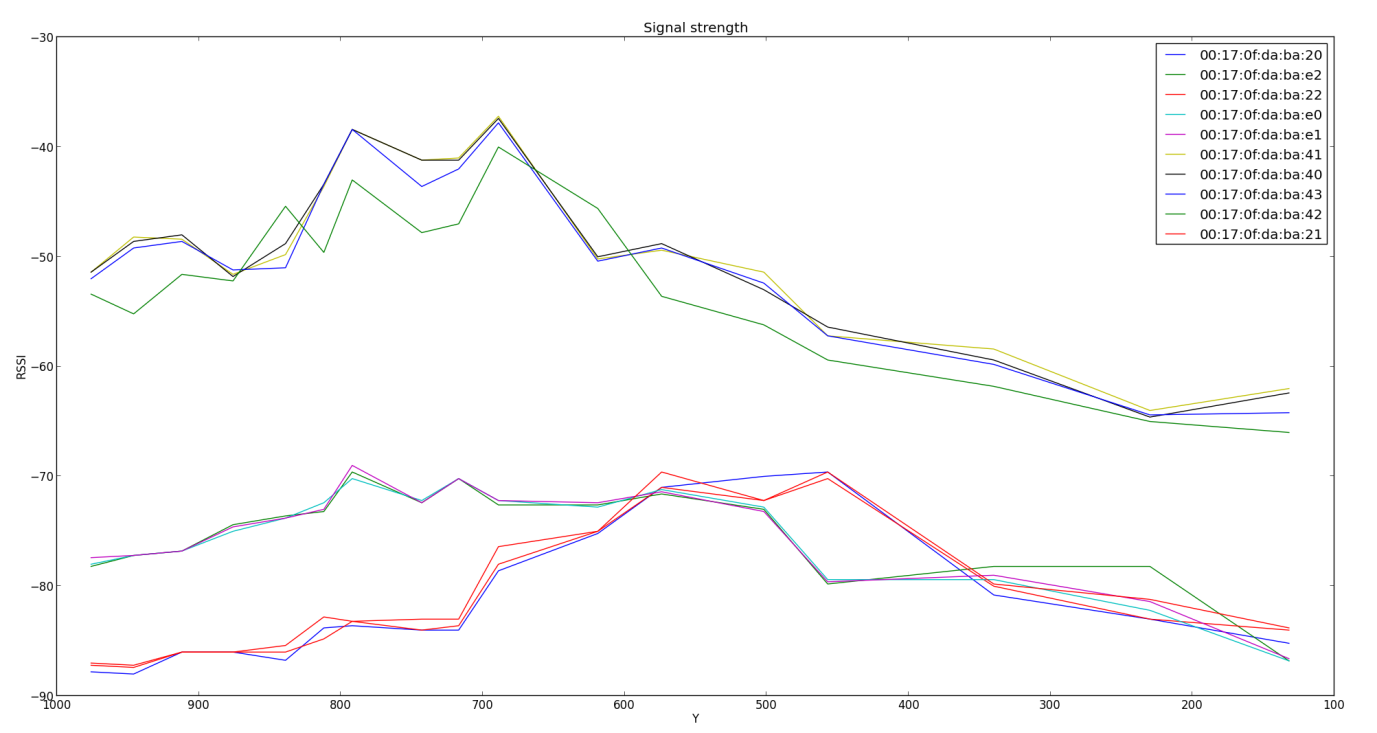


Fig. 2. Puterea semnalului pentru fiecare AP

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

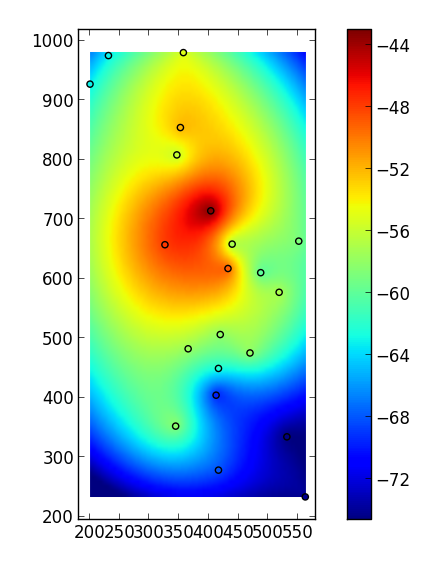


Fig. 3. Harta semnalului pentru un singur AP.

În figura 3 se observă că poziția fizică a unui AP, în caz că nu este cunoscută, poate fi aflată analizând harta semnalului generat de el și interpolând printre valori.

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[[5]](#footnote-6) 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:

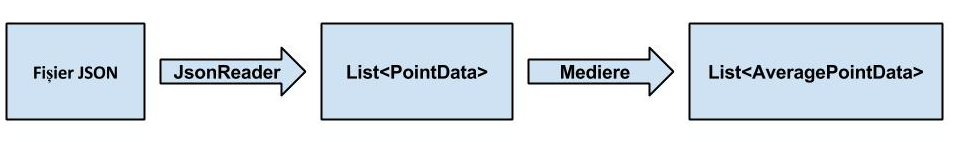
* int x – abscisa punctului de coordonate
* int y – ordonata punctului de coordonate
* List<Map<String, Integer>> samples – lista ce conține toate mostrele recoltate într-un anumit punct (x, y) sub formă de dicționar cu tupluri (BSSID, RSSI); după cum am mai menționat, în fiecare punct se fac mai multe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

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:

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

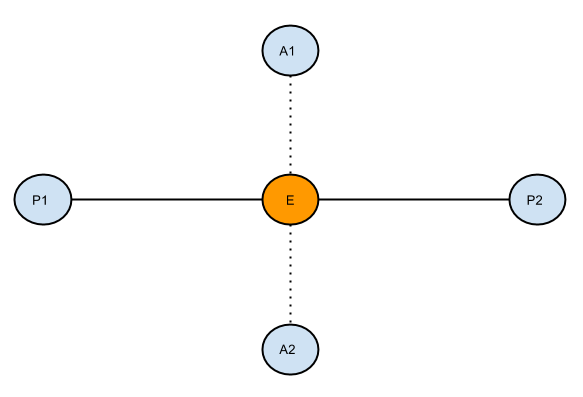
Algoritmul Nearest Neighbour, după cum îi spune și numele, va considera că poziția în care se află utilizatorul este cea care minimizează distanța față de datele online. Acestea fiind spuse, poziția întoarsă va fi:

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.

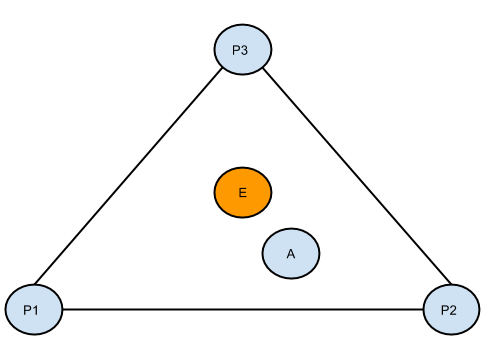


În figura de mai sus, avem:

* 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:



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:

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)

4. Concluzii

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

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

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

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

1. http://www.radiotap.org/defined-fields [↑](#footnote-ref-2)
2. http://developer.android.com/reference/android/net/wifi/WifiManager.html [↑](#footnote-ref-3)
3. http://developer.android.com/reference/android/app/DialogFragment.html [↑](#footnote-ref-4)
4. http://developer.android.com/reference/android/graphics/Bitmap.html [↑](#footnote-ref-5)
5. http://developer.android.com/reference/android/util/JsonReader.html [↑](#footnote-ref-6)