UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Elettronica

Studio e test di applicazione web basata su beacon Bluetooth

Study and test of web application based on Bluetooth beacons

Relatore:
Prof.ssa Paola Pierleoni

Tesi di Laurea di:
Michele Matè

Correlatore:
Dott. Lorenzo Palma

A.A. 2019/2020

Indice

1-INTRODUZIONE

L'indoor positioning in questi ultimi anni ha avuto una grande e rapida evoluzione.

Diverse tecnologie sono state sviluppate con lo scopo di provvedere l'esatta posizione in luoghi al coperto. Il principale vantaggio di queste tecnologie è l'opportunità della facile navigazione indoor, soprattutto nelle grandi strutture come gli aeroporti, centri commerciali, università, ecc.

L'interazione tra il sistema di posizionamento indoor (IPS) e la persona, avviene attraverso diverse tipologie di reti:

-Wi-Fi

-Bluetooth

-Infrarosso

-VlC

-altre

Questi sistemi sono composti da una rete di dispositivi (beacon, router, lampadine) che interagiscono con l'utente attraverso il cellulare, un particolare che comporta una grande opportunità commerciale per le grandi aziende.

Poter sapere dove si trovano le persone dentro le strutture, non solo semplifica lo spostamento, muoversi da un punto A ad un punto B con la strada già indicata; ma offre anche l'opportunità di pubblicizzare i negozi, gli sconti, le marche, presenti nel percorso che deve compiere la persona dentro la struttura.

Purtroppo, sono varie le sfide che affrontano questi sistemi. Di solito queste strutture presentano ostacoli, e le superfici di materiale e dimensioni particolari che ostruiscono e alterano la propagazione del segnale.

Diversi metodi sono stati proposti per la risoluzione di queste problematiche.

2-TIPOLOGIA DI RETI

In questo lavoro di tesi è stato trattato il metodo del *fingerprinting*, adottato dai sistemi che si basano su reti Wi-Fi e Bluetooth.

2.1 – RETI WIFI

Wireless Local Area Network (WLAN), conosciuta come Wi-Fi, trasmette e riceve dati usando onde elettromagnetiche le quali permettono una connessione wireless dentro l'area di copertura.

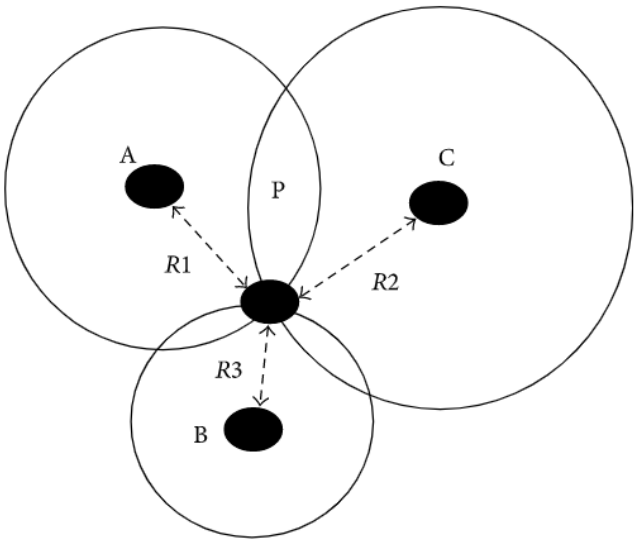
Per la localizzazione indoor le reti WI-FI sfruttano un sistema di identificazione chiamato access point, il quale permette che gli utenti usino i loro dispositivi mobili per sfruttare la rete e stimare la loro posizione attraverso degli algoritmi appositamente creati. Le due tecnologie usate dalle reti Wi-Fi sono:

1.) La "Time and space attributes of received signal" (TSARS):

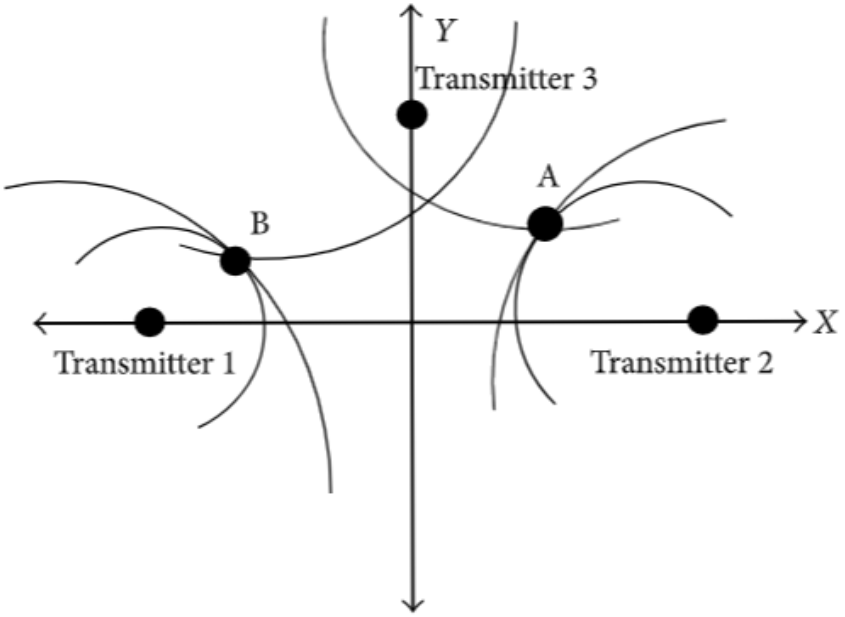
Basta sugli attributi di tempo e spazio del segnale ricevuto.

La quale per il calcolo della posizione usa diversi metodi:

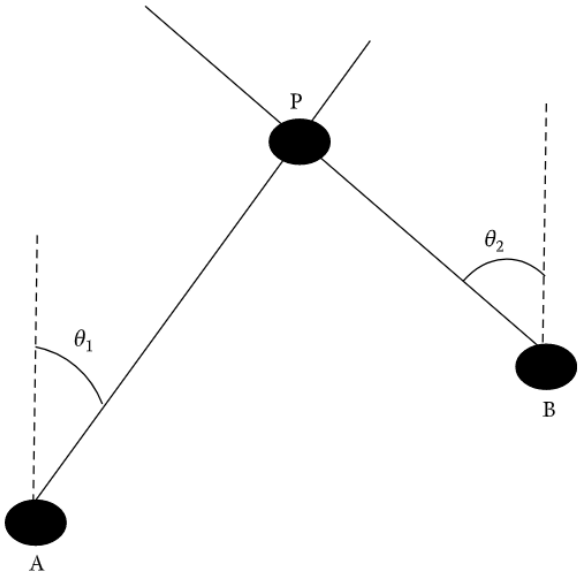
Time of Arrival (ToA). Il tempo che impiega il segnale per andare dal trasmettitore al ricettore è usato per il calcolo della distanza rispettiva tra dispositivi. Un confronto posteriore con altri trasmettitori rispetto ad un unico ricettore permette di calcolare la posizione stimata.



Time Difference of Arrival (TDoA). Simile al ToA, in quanto considera il tempo necessario che impiega il segnale per andare dal trasmettitore al ricettore, però applicato ai casi in cui non c'è sincronia tra i trasmettitori e non si conosce il tempo di emissione di ciascuno. Si usa solo la differenza tra i tempi di ogni trasmettitore rispetto al ricettore.



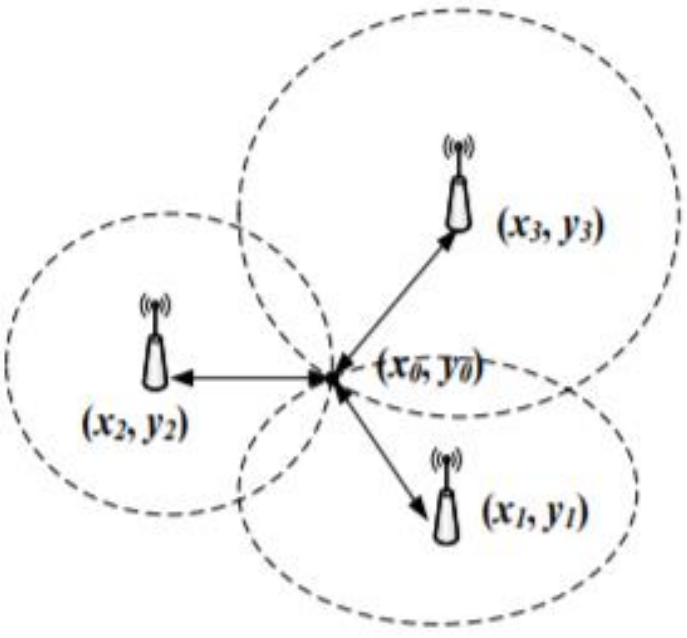
Angle of Arrival (AoA). Ottiene la misura dell'angolo con cui il segnale arriva sul ricettore, rispetto ad un trasmettitore. La combinazione di diversi angoli dei diversi trasmettitori permette calcolare l'intersezione tra i segnali e di conseguenza calcolare la posizione. Non richiede sincronia tra i trasmettitori ma invece necessità di hardware complessi.



2.) La “Recieved Signal Strenght-Based Positioning Technology (RSSI)

È l’intensità del segnale misurato dal ricettore. La distanza del segnale è calcolata usando l’equazione di Friis la quale considera il rapporto tra la potenza ricevuta da un’antenna e la potenza trasmessa. Per ottenere poi il posizionamento l’RSSI usa metodi come quelli della:

Trilaterazione. Questo metodo usa minimo 3 punti di riferimento per inviare il segnale al dispositivo mobile. Sfruttando la distanza spaziale questa viene usata come i raggi dei cerchi per ogni punto di riferimento, l’intersezione tra questi, di conseguenza indicherà la posizione stimata del ricettore.



2.2 – RETI BLUETOOTH

Bluetooth è una tecnologia di comunicazione wireless sviluppata con lo scopo di facilitare la comunicazione tra dispositivi mobili e la sincronia tra di essi.

E' la competenza principale delle reti Wi-Fi in quanto al posizionamento indoor, specialmente da quando si è sparsa l'implementazione dei dispositivi Bluetooth Low Energy (BLE) i quali sono supportati dai dispositivi mobili. Essi hanno un basso costo e un basso consumo energetico.

Questa tecnologia usa beacon Bluetooth come trasmettitori di segnale. Questi beacon sono dispositivi embedded di piccole dimensioni i quali trasmettono il loro segnale identificativo a tutti i dispostivi portatili attraverso una apposita applicazione. Il segnale univoco contiene non solo l'id, conosciuto come UUID (Universally Unique Identifier) ma può anche contenere altri parametri come, la posizione esatta e valori di RSSI.

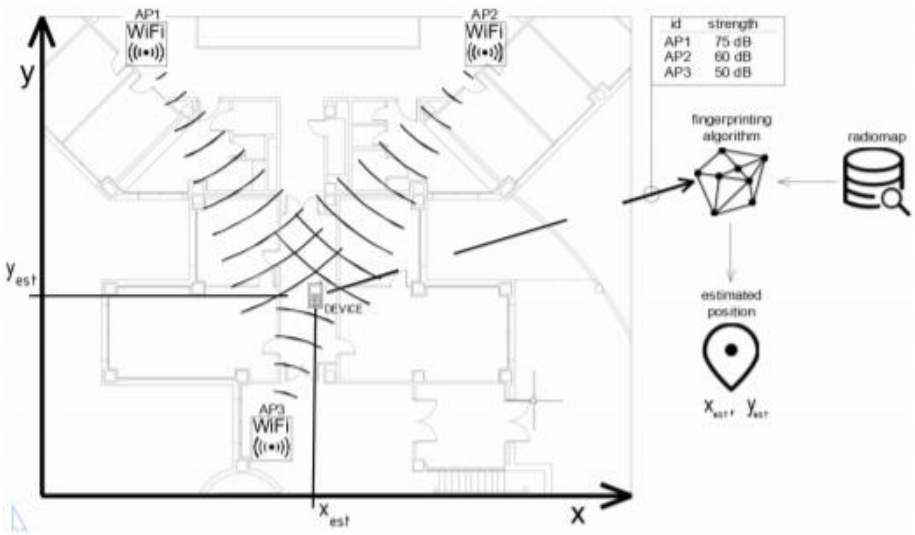
3- TECNICA DI LOCALIZZAZIONE

Il *fingerprinting* è un metodo usato per la stima della posizione indoor. Applicato su una rete Bluetooth ed insieme alla tecnologia RSSI questo metodo opera in due fasi:

1.) Training Phase: Nella prima fase, anche conosciuta come *off-line phase*, i beacon Bluetooth vengono distribuiti sulla superficie e registrati dentro un *radio map*, nel quale ognuno di questi contiene la posizione esatta, l'id e informazione extra necessaria di cui abbia bisogno il sistema. Insieme ai beacon dentro al *radio map* vengono registrate posizioni note con i valori di RSSI rispettivi ad ogni beacon. Viene così generata una mappa con valori esatti sia di posizione che di RSSI.

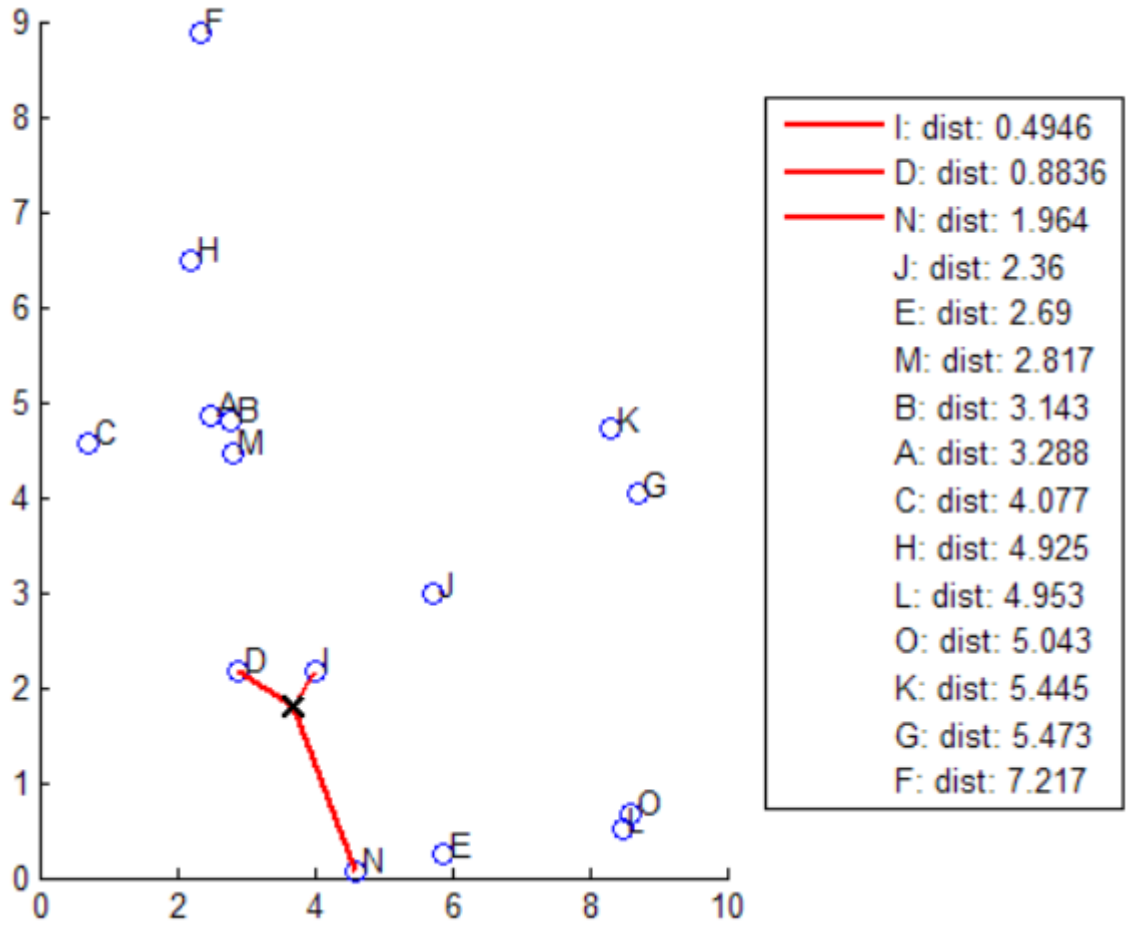
Radio map. È un insieme di dati composti da informazioni ricevute dai beacon o dai dispositivi mobili tramite fingerprinting. Le misure della posizione registrati tramite fingerprinting possono contenere variabili quali tipo di dispositivo usato, tempo della misura, e qualsiasi altra informazione necessaria per la stima della posizione. Un'alternativa ai *radio map* sono le basi di dati.

2.) Positioning Phase: Nella seconda fase, l'*online phase*, il sistema è operativo e disponibile agli utenti. Ogni singola persona con a disposizione un cellulare può misurare i valori di RSSI e compararli con quelli registrati precedentemente per dare così una stima della posizione.



3.1 – ALGORITMO PER LA LOCALIZZAZIONE:

Per la stima della posizione con il metodo del *fingerprinting* si usa l'algoritmo del *kNN* (k-Nearest-Neighbour). Questo algoritmo confronta i dati ottenuti nel *radio map* durante la positioning phase e quelli nella training phase. La stima è fatta in base alla vicinanza tra i dati. Ciò può essere fatto se la distanza tra beacon e ricettore è considerata come una distanza euclidea. Di conseguenza, paragonando la misura nella online phase con quelle già esistenti, basterà trovare la distanza minima e assegnare a al ricettore la stima della posizione.

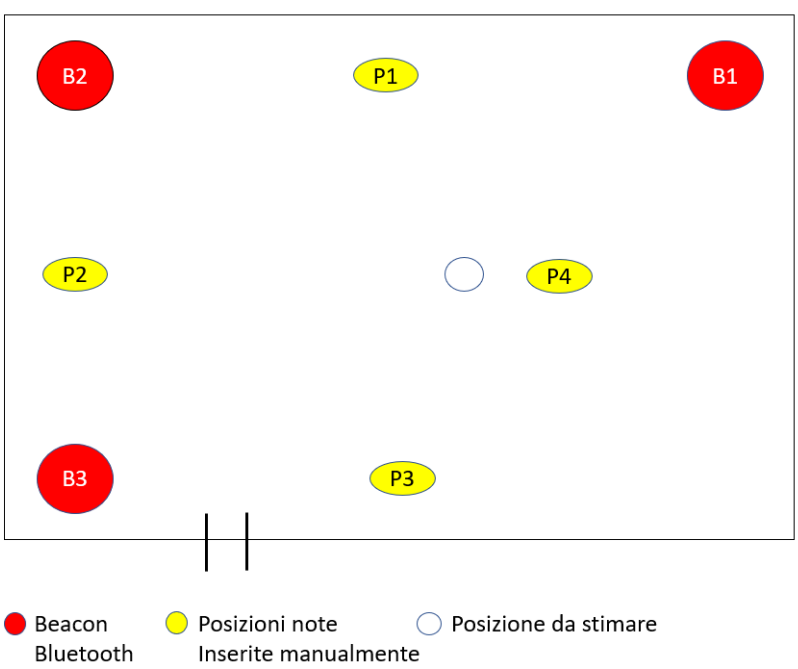


4- IL PROGETTO

Per lo sviluppo di questo lavoro di tesi abbiamo deciso di lavorare con sensori Bluetooth (BLE) e abbiamo trattato un metodo alternativo per l'implementazione del metodo del *fingerprinting*.

I beacon Bluetooth vengono sistemati dentro una struttura per coprire la maggior parte dell'area possibile. Ognuno di questi beacon ha un ID identificativo e una posizione precisa misurata tramite GPS. Questa informazione viene salvata in un database il quale sarà l'incaricato di gestire tutta l'informazione e posteriormente anche la stima della posizione. Durante l'*offline phase* oltre ai dati dei beacon, vengono anche registrate diverse posizioni, acquisite tramite GPS, e i loro valori di RSSI rispettivamente.

Per maggior accuratezza i test del progetto sono stati svolti in una stanza con modeste dimensioni (4mq) e per lo più libera da ostacoli per non interferire con il segnale dei beacon.
3 beacon Bluetooth sono stati sistemati in 3 vertici della stanza per coprire la maggior parte della superficie.



- P1: lat: 43.59986 lng: 13.51348
- P2: lat: 43.59984 lng: 13.51350
- P3: lat: 43.59986 lng: 13.51352
- P4: lat: 43.599865 lng: 13.51350

- B1: lat: 43.59987 lng: 13.51348
- B2: lat: 43.59984 lng: 13.51348
- B3: lat: 43.59984 lng: 13.51352

Insieme ai beacon sono state anche registrate 4 posizioni note le quali saranno usate come punti di riferimento per la stima della posizione nella *online phase*. Ogni beacon e ogni posizione hanno un

attributo latitudine e un attributo longitudine, però ogni posizione ha anche un attributo con il valore dell'RSSI rispetto ad ogni beacon.

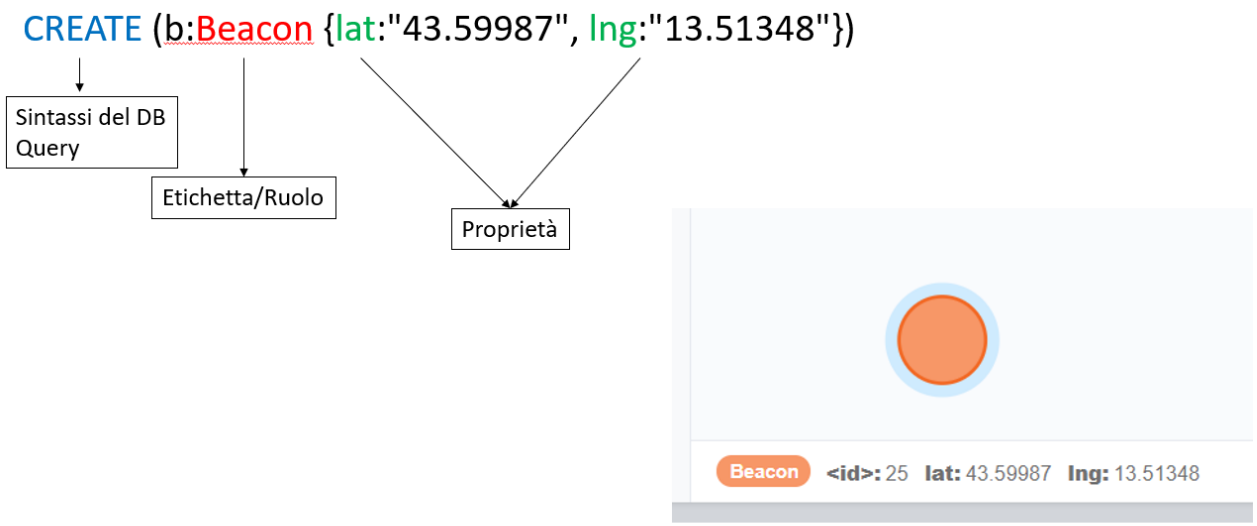
Durante l'*online phase*, i dispositivi mobili si interfacciano con il database attraverso un'applicazione mobile dalla quale si stima la posizione dell'utente.

A differenza dei metodi precedentemente introdotti, in questo lavoro di tesi si è considerata la possibilità di stimare la posizione attraverso i dati raccolti nel database, paragonando tutti i dati presenti. Quando l'utente desidera sapere la sua posizione attuale, l'applicazione interroga il database, con un algoritmo appositamente creato, ed ottiene una stima dell'ubicazione attuale.

5- IL DATABASE

Il database scelto per questo lavoro di tesi è stato Neo4j, un database a grafo.

I database di questa tipologia si basano su una struttura dati a grafo dotata di interfaccia CRUD (Create, Read, Update, Delete), offre quindi operazioni di creazione, lettura, aggiornamento e cancellazione dei dati.
Un grafo è costituito da nodi e relazioni che stabiliscono collegamenti tra di essi. Le informazioni sono presenti in entrambi.
Il linguaggio di query dichiarativo del database è denominato Cypher.



Questo è un esempio di query per la creazione di un nodo. Sopra a sinistra si mostra la struttura del cypher. Con la funzione **CREATE** si crea un nuovo nodo, identificato con le parentesi tonde.

Struttura di un nodo:

ID: Ad ogni nodo viene assegnato un ID univoco il quale potrà anche essere usato per interrogare il DB.

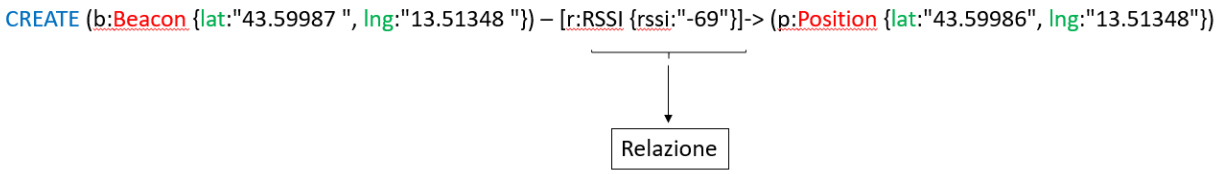
LABEL: L'etichetta serve a classificare i nodi e indicizzarli, ciò vuol dire che d'ora in poi dentro il db ci sarà un gruppo di nodi che saranno classificati come nodi ai quali graficamente verrà attribuito un colore. Come mostrato in figura. Un nodo può avere più di una label.

PROPRIETA': Dentro le parentesi graffe si possono inserire tutti i dati che si vogliono avere per il nodo, (posizione, nome, città, numero, ecc)

Per il lavoro di tesi come anticipato, si sono implementati 3 beacon Bluetooth i quali sono stati inseriti nel DB. Ognuno di questi è stato classificato con il label "Beacon" e gli sono state assegnate 2 proprietà, latitudine e longitudine.

Il seguente passaggio è stato la registrazione delle posizioni note. Per far ciò si sono creati dei nodi con label "Posizione" che hanno come relazione il valore di RSSI rispetto ad ogni beacon.

Le relazioni identificate con le parentesi quadre, svolgono il compito di mettere in rapporto i nodi. Sotto in figura si vede la sua rappresentazione dentro il database come una freccia che parte da un nodo e va a finire in un altro.

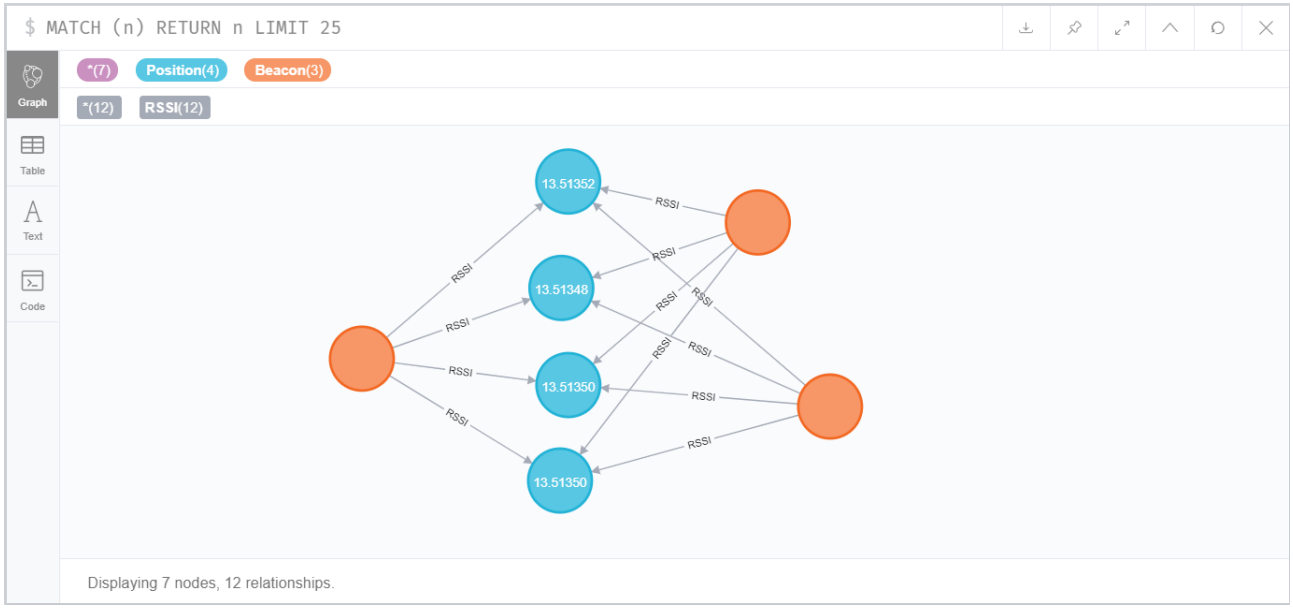


():Nodo /Oggetto []:Relazione

Questa figura mostra un "full path", si stanno creando due nodi con una relazione in comune, con le rispettive label e proprietà.

C'è da tener in conto che le relazioni devono sempre avere un **TIPO** che in questo caso è "RSSI" e hanno necessariamente una direzione. Non si possono creare relazioni senza verso però al

momento di percorrere o interrogare il DB, si possono fare query senza tener conto di questo. Posso arrivare al nodo Beacon attraverso il nodo Posizione anche se il verso della relazione va in direzione contraria.



Rappresentazione del layout dentro Neo4j.

5.1 – INTEGRAZIONE DB E APP

Il database può essere usato sia in modalità embedded che server. Nella modalità embedded si incorpora il database nell'applicazione e questo viene eseguito all'interno. Nella modalità server invece il database è un sistema separato *stand-alone*, accessibile tramite browser. Per poterlo usare con applicazioni mobili si devono importare specifiche librerie ed usare funzioni chiamate transazioni. Una volta aperta una transazione è possibile creare nodi, assegnare delle proprietà, relazioni ma è anche possibile eseguire comandi specifici del DB così da ricevere dati in remoto.

In questo lavoro di tesi si è scelto di lasciare il DB come *stand alone*.

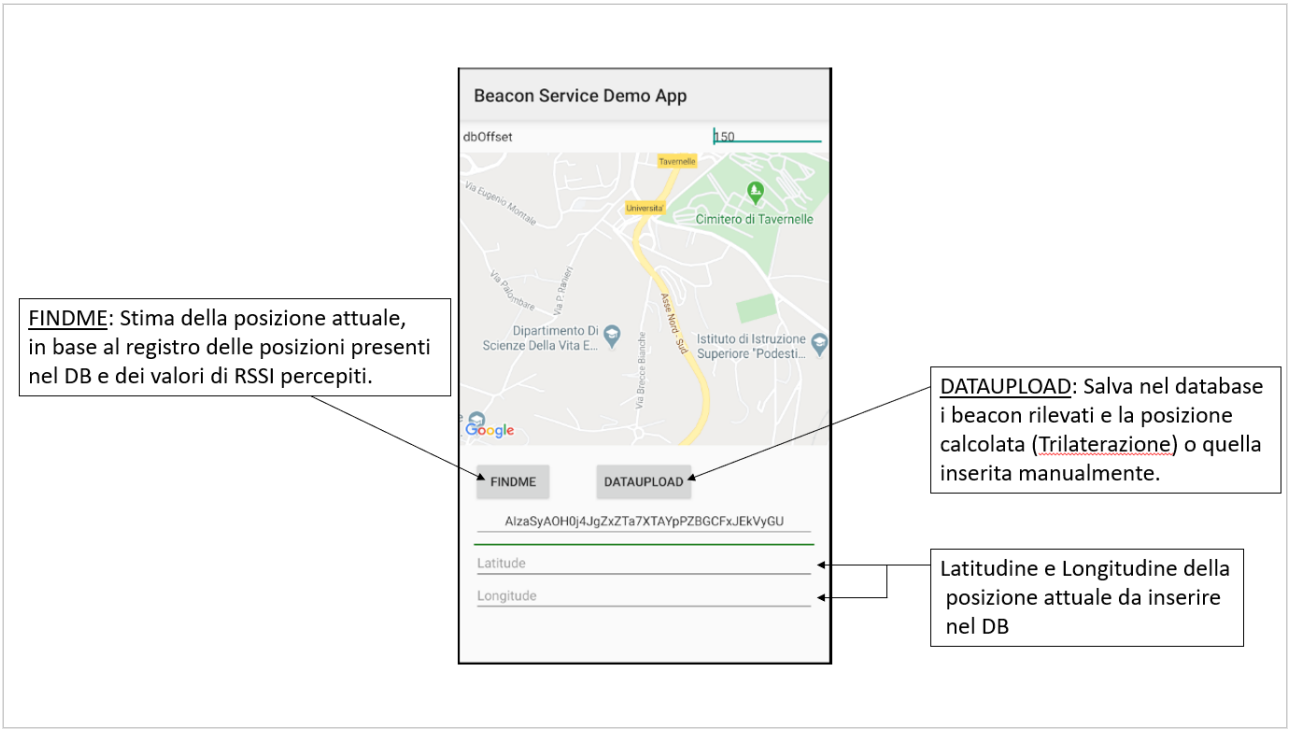
Questa figura mostra il sistema che è stato usato per poter accedere al DB.

L'applicazione usata come collegamento tra DB e dispositivi è stata sviluppata in JAVA e il Framework usato è Android Studio 3.6.1.
Per poter usare le library di Neo4j il targetSdkVersion deve essere minimo 26.

Come affermato prima per poter accedere al DB si deve creare una "transazione" dentro la quale potremmo inviare una query. Questo avviene dentro il metodo "Neo4j", mostrato in figura, il quale per prima cosa andrà ad accedere al DB attraverso una IP passando i token di username e password. Se tutto è andato a buon fine, si istaura una sessione dentro la quale è possibile scrivere la query. Ogni volta che accedo ed invio una query, la sessione chiude l'accesso e si deve ripetere tutto da capo. Questo vuol dire che per ogni singolo nodo, relazione o informazione che voglio creare o avere si deve aprire una nuova transazione.

6- APPLICAZIONE JAVA

Questo lavoro di tesi si è sviluppata su una applicazione già esistente, la quale fa una stima della posizione indoor tramite il metodo di trilaterazione usando come rete i beacon Bluetooth.



Questa è la nuova interfaccia, il sistema della stima della posizione è stato lasciato invariato ma si è integrato a quello nuovo il quale utilizza il database Neo4j.

L'interfaccia è suddivisa in 2 parti principali. La prima è la mappa nella quale verrà mostrata la nostra posizione stimata. La seconda invece sono i bottoni con i quali possiamo interagire con l'app e anche con il database.

6.1- BOTTONI E CODICE

Dai due bottoni possiamo sia ottenere i valori stimati tramite trilaterazione che interagire con il database.

6.1.i- DATAUPLOAD

Il bottone centrale chiamato “DATAUPLOAD” svolge il compito principale di salvare tutta l’informazione che otteniamo dai beacon e dal metodo di trilaterazione, ma permette anche la possibilità di aggiungere posizioni manualmente, attraverso due *blank spaces* indicati con il nome di “Latitude” e “Longitude”. Quest’ultima opzione è stata aggiunta per mappare manualmente tutto lo spazio che coprono i nostri beacon così da ridurre, nel possibile, l’errore che introduce il calcolo della posizione mediante il metodo della trilaterazione.

```
scanButton.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Handler hndl = new Handler();
    hndl.post(runThis);
    System.out.println("discovered: " + discovered);
  }});
```

Quando il bottone “DATAUPLOAD” viene cliccato, viene eseguito un thread tramite il *runThis*.

```
runThis = new Runnable() {
  @Override
  public void run() {

    Utils.setEnabledViews( enabled: false, scanButton);
    Utils.setEnabledViews( enabled: true, rssiButton);
    sharedPreferences.edit().putString( s: "apikey",apiKeyView.getText().toString()).apply();
    arrayAdapter.clear();
    scanner.startScan(SCAN_FILTERS, SCAN_SETTINGS, scanCallback);
    Log.i(TAG,  msg: "starting scan");
    //client = new ProximityBeaconImpl(getActivity(), accountNameView.getText().toString());//To
    CountDownTimer countDownTimer = new CountDownTimer(SCAN_TIME_MILLIS,  countDownInterval: 100) {
      @Override
      public void onTick(long millisUntilFinished) {
        double i = (1 - millisUntilFinished / (double) SCAN_TIME_MILLIS) * 100;
        progressBar.setProgress((int) i);
      }

      @Override
      public void onFinish() {
        progressBar.setProgress(100);

      }
    };
      countDownTimer.start();
      Runnable stopScanning = new Runnable() {
        @Override
        public void run() {
          scanner.stopScan(scanCallback);
          Log.i(TAG,  msg: "stopped scan");
          Utils.setEnabledViews( enabled: true, scanButton);
          Utils.setEnabledViews( enabled: true, rssiButton);
          if(discovered.size()>=3) {//it really makes sense only to do this if we know we have a
              String latitude = lat.getText().toString();
              String longitude = lng.getText().toString();
              if (latitude == null || latitude.equals("") || longitude == null || longitude.equa
                  mCallback.onListUpdated(discovered,  latitude: null,  longitude: null);//moved here
              }
              else {
                  mCallback.onListUpdated(discovered, latitude, longitude);
              }
          }


        }
      };
```

Dentro il “runThis” si eseguono dei controlli. Il primo check che si fa è quello di vedere se in effetti ci sono almeno tre beacon Bluetooth vicini al nostro dispositivo mobile (tre è il numero che è stato

scelto per questo lavoro di tesi, si possono usare quanti ne servano o quanti ce ne siano a diposizione).

Il seguente controllo che fa è verificare se sulle caselle vuote è stato inserito qualche valore. Qualunque sia il risultato, la lista con i beacon e i valori di latitudine e longitudine della posizione saranno inviati al metodo “mCallback” il quale a sua volta passa questi dati al metodo “onListUpdated” della classe “MainActivity”.

```
public void onListUpdated(ArrayList<Beacon> list, String latitude, String longitude){
  TextView dbOffset=(TextView)findViewById(R.id.dboffset);//il valore dovrebbe essere intorno ai 60 ma va meglio con 150
  MapsFragment mMap =(MapsFragment)getFragmentManager().findFragmentById(R.id.mapFragment);
  for( Beacon b:list ) {
    mMap.spawnBeacon(b);
  }
  ///Todo: commentato perché prende troppe risorse
    double[][] position =new double[list.size()][3];
    double distance[] = new double[list.size()];
    //writeLogFile("\n\ntime:"+ System.nanoTime()+"\n\n",logFile,this);
  for(Beacon beacon: list){
    writeLogFile( message: "\n"+counter+";\t"+"#"+System.nanoTime()+";\t"+beacon.getHexId()+";\t"+beacon.latitude+";\t"+beacon.longitude+";\t"+beacon.getRssi()+";\t"
    double[] arr = convertToCartesian(beacon.getLatLng());

    position[list.indexOf(beacon)][0] = arr[0];//x
    position[list.indexOf(beacon)][1] = arr[1];//y
    position[list.indexOf(beacon)][2] = arr[2];//z

    distance[list.indexOf(beacon)] = calculateDistance(Integer.parseInt(dbOffset.getText().toString()),beacon.getRssi());
  }
  if (latitude == null && longitude == null) {

    if ((actualPosition == new LatLng( v: 0,  v1: 0) | true)) { //ToDo: il true bypassa il check sull'ultima posizione (forza il calcolo)
      NonLinearLeastSquaresSolver solver = new NonLinearLeastSquaresSolver(new TrilaterationFunction(position, distance), new LevenbergMarquardtOptimizer());
      try {
        Optimum optimum = solver.solve();
        double[] centroid = optimum.getPoint().toArray();
        if (convertToLatLng(centroid) != actualPosition) {
          Log.i(TAG,  msg: "centroid:  (" + centroid[0] + "," + centroid[1] + "," + centroid[2] + ")");
          actualPosition = convertToLatLng(centroid);
          mMap.spawnMe(actualPosition);
          Integer iterations = optimum.getIterations();
          writeLogFile( message: "found;\t" + actualPosition.latitude + ";\t" + actualPosition.longitude + ";\t" + "iterations" + ";\t" + iterations, logFile,
          Log.i(TAG,  msg: "\n iterations: " + iterations.toString());
          //Log.i(TAG, "Spawning position: " + actualPosition.toString());
          String lat = actualPosition.latitude + "";
          String lng = actualPosition.longitude + "";
          //Neo4j.main(lat, lng);

          for (Beacon beacon : list) {

            String blat = beacon.latitude.toString();
            String blng = beacon.longitude.toString();
            Integer rssi = beacon.rssi;

            try {
              Neo4j.main(blat, blng, lat, lng, rssi);
            } catch (Exception e) {
              System.out.println("Error");
            }
          }
        }
      } catch (Exception e) {

      }
      writeLogFile( message: "\n", logFile,  ctx: this);
      counter++;
    }
  }

  else {
    for (Beacon beacon : list) {

      String lat = latitude;
      String lng = longitude;
      String blat = beacon.latitude.toString();
      String blng = beacon.longitude.toString();
      Integer rssi = beacon.rssi;

      try {
        Neo4j.main(blat, blng, lat, lng, rssi);
      } catch (Exception e) {
        System.out.println("Error");
      }
    }

  }

}
```

“OnListUpdated” è l’incaricato del calcolo della posizione tramite trilaterazione e dell’invio dell’informazione dei beacon e della posizione al metodo “Neo4j.main” il quale comunica direttamente con il database. Come indicato precedentemente se si sceglie di non ottenere i dati necessari tramite il metodo matematico si usano quelli inseriti manualmente.

6.1.ii- FINDME

Il secondo bottone che si vede nell’interfaccia chiamato “FINDME” è l’incaricato di stimare la posizione a partire dai dati presenti nel database, interrogandolo attraverso un algoritmo appositamente creato.

```
rssiButton.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    try {
      System.out.println("discovered: " + discovered);
      Handler hndl = new Handler();
      hndl.post(runThis2);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
});
```

```
runThis2 = new Runnable() {
  @Override
  public void run() {
    Utils.setEnabledViews( enabled: true, rssiButton);
    sharedPreferences.edit().putString( s: "apikey",apiKeyView.getText().toString()).apply();
    arrayAdapter.clear();
    scanner.startScan(SCAN_FILTERS, SCAN_SETTINGS, scanCallback);
    Log.i(TAG,  msg: "starting scan");
    //client = new ProximityBeaconImpl(getActivity(), accountNameView.getText().toString());//ToDo:ripristinato per
    CountDownTimer countDownTimer = new CountDownTimer(SCAN_TIME_MILLIS,  countDownInterval: 100) {
      @Override
      public void onTick(long millisUntilFinished) {
        double i = (1 - millisUntilFinished / (double) SCAN_TIME_MILLIS) * 100;
        progressBar.setProgress((int) i);
      }

      @Override
      public void onFinish() {
        progressBar.setProgress(100);

      }
    };
    countDownTimer.start();
    Runnable stopScanning = new Runnable() {
      @Override
      public void run() {
        scanner.stopScan(scanCallback);
        Log.i(TAG,  msg: "stopped scan");
        Utils.setEnabledViews( enabled: true, rssiButton);
        if(discovered.size()>=3) {//it really makes sense only to do this if we know we have at least 3 beacons
          mCallback2.callNeo4j(discovered);
        }
```

Come si vede dalle figure, dentro alla classe “MainActivityFragment” questo nuovo bottone esegue dei check simili a quelli del “DATAUPLOAD”. Si fa uno scan dei beacon rilevati e si controlla che il numero sia il minimo necessario, dopo di che, invia i dati a “mCallback2” il quale a sua volta invia i dati al metodo “callNeo4j” della “MainActivity” class.

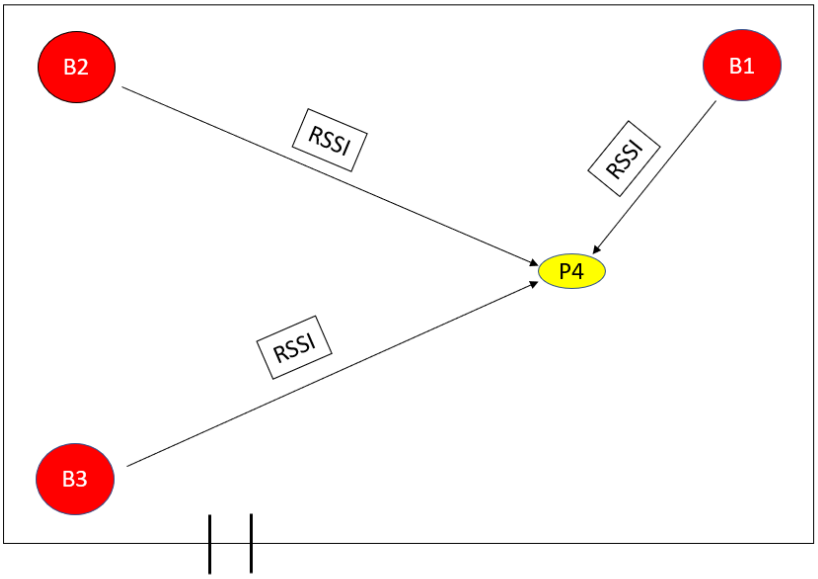
```
public void callNeo4j(ArrayList<Beacon> list) {
    System.out.println("list: " + list.size());

    Beacon b1 = list.get(0);
    Beacon b2 = list.get(1);
    Beacon b3 = list.get(2);
    try {
        Neo4j.main2(b1, b2, b3);
    } catch (Exception e) {
    }

}
```

Il compito di “callNeo4j” è quello di inviare i dati dei beacon rilevati e i valori degli RSSI per poter paragonare la posizione attuale con quelle già registrate nel database.

6- ALGORITMO DI LOCALIZZAZIONE

Il metodo sviluppato per la stima della posizione in questo lavoro di tesi è basato sul paragone dei dati presenti nel database. Un algoritmo appositamente creato interroga il DB e dalle posizioni registrate durante l'*offline phase* si ricava la stima finale.

Per poter effettuare questo calcolo devono già essere stati registrati i beacon con le rispettive informazioni di latitudine e di longitudine e aver effettuato degli scan per poter inserire delle posizioni di riferimento con i loro valori di latitudine, longitudine e RSSI.



DATAUPLOAD:

1. Scan + Intensità RSSI
2. Query al DB:

```
MERGE (b:Beacon {lat:"blat", lng:"blng"})
MERGE (p:Position {lat:"lat", lng:"lng"})
WITH b, p
MATCH (b:Beacon), (p:Position)
CREATE (b) - [r:RSSI {rssi:"rssi"] -> (p)
RETURN p.lat
```

Può anche calcolare la posizione tramite **trilaterazione**.

Come detto prima, tutto ciò è gestito dal bottone "DATAUPLOAD" ed in figura si vede la rappresentazione grafica di quello che succede quando si inserisce un nuovo nodo di tipo Posizione dentro al DB e la rispettiva query usata per registrarlo.

Una volta che abbiamo registrato tutti i dati necessari si può passare all'*online phase* ed interrogare il database per ottenere la stima della posizione.

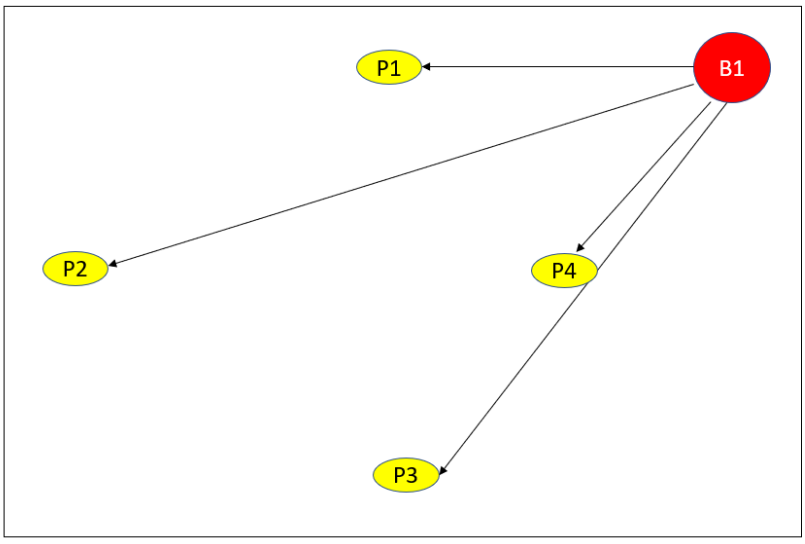
L'algoritmo creato è una query che confronta i dati registrati, in particolare i valori di RSSI di ogni nodo posizione con dei dati ottenuti successivamente nell'*online phase.*

Questa è la query incaricata della stima della posizione gestita dentro l'app dal bottone "FINDME"

FINDME:

```
MATCH (b1:Beacon{lat:"43.59987", lng:"13.51348"})-[r1:RSSI]->(p1)
MATCH (b2:Beacon{lat:"43.59984", lng:"13.51348"})-[r2:RSSI]->(p2)
MATCH (b3:Beacon{lat:"43.59984", lng:"13.51352"})-[r3:RSSI]->(p3)

WHERE ID(p1) = ID(p2)
AND ID(p1) = ID(p3)

RETURN abs(r1.rssi-"rss1"), abs(r2.rssi-"rssi2"), abs(r3.rssi-"rssi3"),
(abs(r1.rssi-"rss1") + abs(r2.rssi-"rssi2") + abs(r3.rssi-"rssi3")) as tot

ORDER BY tot LIMIT 1
```

Si procede con una spiegazione grafica per capire ogni singola linea della query.

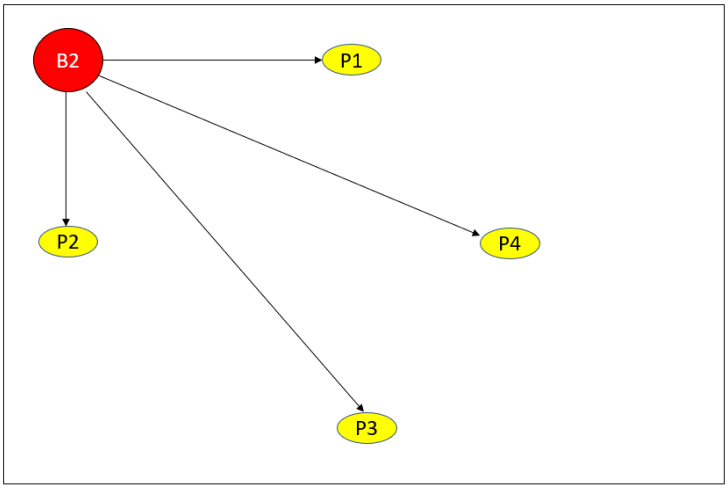


MATCH B1			
ID(P1)	lat	lng	r1:RSSI
13	43,59986	13,51348	-60
20	43,59984	13,51350	-45
15	43,59986	13,51352	-55
23	43,599865	13,51350	-70

```
MATCH (b1:Beacon{lat:" 43.59987", lng:" 13.51348"})-[r1:RSSI]->(p1)
```

La prima riga si ripete 3 volte e come si vede dalla figura, serve per ottenere tutti i dati dei nodi “Posizione” associati ad un nodo di tipo “Beacon”. In questo caso il nodo B1 corrisponde al primo beacon registrato. Il risultato della prima parte della query è riassunta nella tabella di destra. Abbiamo così ottenuto, ID, latitudine, longitudine e RSSI delle posizioni già presenti nel DB.

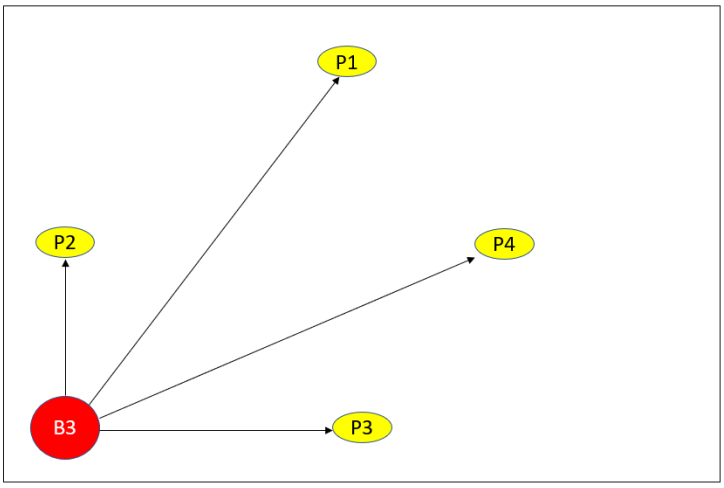
Questo si ripete *n* volte, dove *n* dipende dal numero di beacon rilevati dall’app.



MATCH B2			
ID(P2)	lat	lng	r2:RSSI
13	43,59986	13,51348	-52
20	43,59984	13,51350	-60
15	43,59986	13,51352	-43
23	43,599865	13,51350	-66

```
MATCH (b2:Beacon{lat:"43.59984 ", lng:"13.51348 "})-[r2:RSSI]->(p2)
```

Rispetto a B2

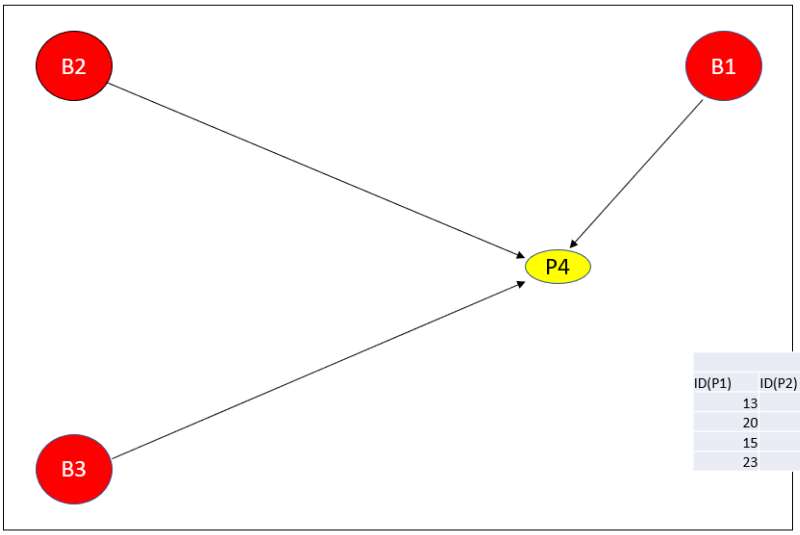


MATCH B3			
ID(P3)	lat	lng	r3:RSSI
13	43,59986	13,51348	-55
20	43,59984	13,51350	-62
15	43,59986	13,51352	-49
23	43,599865	13,51350	-60

```
MATCH (b3:Beacon{lat:"43.59984", lng:" 13.51352"})-[r3:RSSI]->(p3)
```

Rispetto a B3

Abbiamo ottenuto tutti di dati dei nodi *Posizione* però per poter fare il confronto dobbiamo essere sicuri di considerare solo un nodo alla volta, con i suoi rispettivi dati.



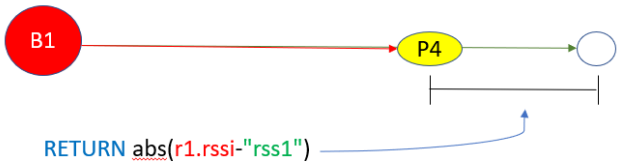
WHERE ID(P1) = ID(P2) = ID(P3)

ID(P1)	ID(P2)	ID(P3)	LAT	LNG	r1:RSSi	r2:RSSI	r3:RSSI
13	13	13	43,59986	13,51348	-60	-52	-55
20	20	20	43,59984	13,51350	-45	-60	-62
15	15	15	43,59986	13,51352	-55	-43	-49
23	23	23	43,599865	13,51350	-70	-66	-60

```
WHERE ID(p1) = ID(p2)
AND ID(p1) = ID(p3)
```
Sto considerando lo stesso nodo! (la stessa posizione)

Per far ciò si fa coincidere l'ID dei risultati ottenuti così da avere tutti i valori in un'unica tabella.

Dopo di che si procede con il confronto.




```
RETURN abs(r1.rssi-"rss1")
```

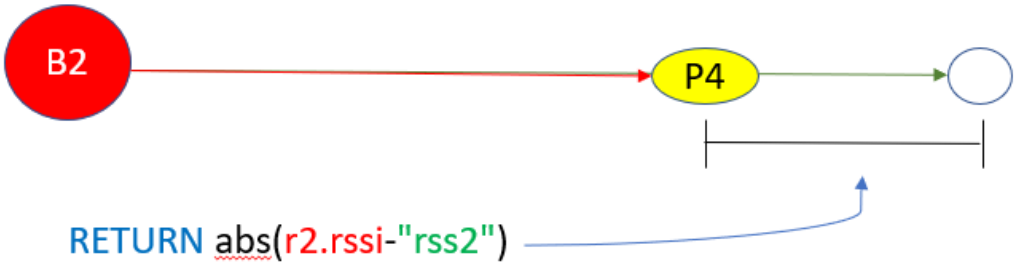
r1.rssi: Intensità RSSI della posizione registrata nel DB

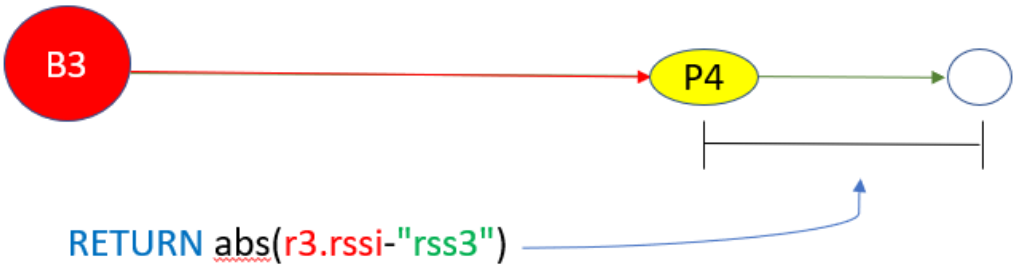
"rss1": Intensità RSSI della posizione da stimare

Beacon Bluetooth

Posizioni nota Inserita manualmente

Posizione da stimare

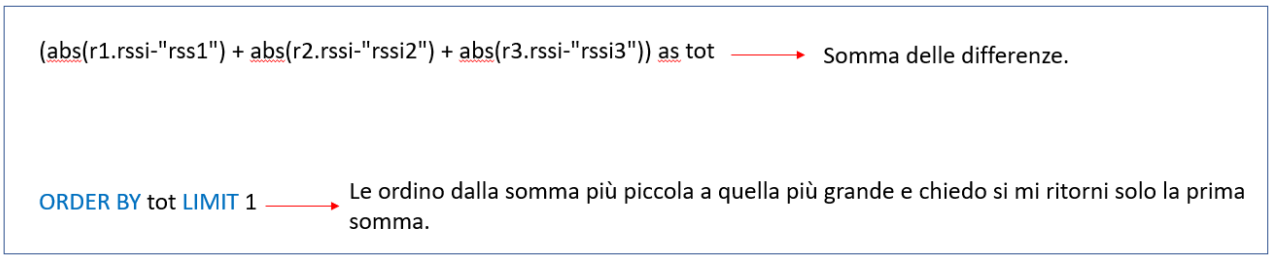




I dati che vengono paragonati sono i valori di RSSI.

Dopo che il bottone “FINDME” è stato cliccato ed è stato fatto lo scan dei beacon in prossimità, ottengo dei nuovi valori di RSSI per la posizione attuale da stimare. Questo valore è stato denominato nella query della figura come “rss1, rss2 e rss3”. Il seguente step dell’algoritmo è confrontare il valore attuare di RSSI con ognuno di quelli presenti nel DB, questo passaggio si ripete *n* volte, quanti siano i beacon rilevati in fase di scan.

Nell’esempio, mostrato nelle figure precedenti, la posizione da stimare, rappresentata dal cerchio bianco, viene confrontata con il nodo *Posizione* “P4”. Il valore attuale dell’RSSI rispetto al beacon B1 è sottratto al valore di RSSI del nodo “P4” sempre rispetto al beacon B1 (r1.rssi). Questo passaggio si ripete per ogni beacon rilevato.



Una volta effettuati tutti i calcoli rispetto a tutti i nodi “Posizione” presenti nel DB, il modo per stabilire la posizione è quello di sommare le differenze, organizzarle in ordine crescente e determinare il risultato più piccolo come il più vicino.

Si assegna alla posizione da stimare il valore di latitudine e longitudine del nodo più vicino.

7- CONCLUSIONI

Dai diversi test effettuati, nell'80% dei casi l'algoritmo ha effettivamente assegnato alla posizione da stimare, quella più vicina già registrata in precedenza. La scelta dell'RSSI come parametro per il paragone tra posizioni si è rivelato un metodo alternativo a quelli matematici citati nell'introduzione.

Il basso costo dei trasmettitori introduce anche la possibilità di ampliare la rete di sensori e di renderla più densa per coprire, il più possibile, tutta la struttura. Ciò comporterebbe un aumento nell'accuratezza della stima visto l'utilità di avere più nodi da confrontare.

Questo include anche il fatto che una maggior acquisizione di dati durante l'*offline phase*, quasi una mappatura totale della superficie, porterebbe errori minimi al momento di assegnare la posizione.

Tuttavia, il lavoro si è svolto in un ambiente il più possibile libero da oggetti, le misure effettuate dell'intensità dell'RSSI si sono dimostrate molto suscettibile a variazioni dovute sia a l'esistenza di piccoli oggetti nella stanza, che dalle riflessioni dei materiali presenti nello spazio dei test. L'errore oscilla attorno a ±5 dB.

L'eventuale introduzione di metodi per l'accuratezza nella misura dell'intensità del segnale ricevuto, ridurrebbe i problemi riscontrati e rinforzerebbe lo sviluppo di metodi di indoor positioning attraverso paragoni di valori di RSSI sostenuto da basi di dati.

8- BIBLIOGRAFIA

1 Shixiong Xia, Yi Liu, Guan Yuan, Mingjun Zhu and Zhaohui Wang. “Indoor Fingerprinting Based on Wi-Fi: An Overview”. 28 aprile 2017

2 Ramon F. Brena, Juan Pablo Garcìa-Vázquez, Carlos E. Galván-Tejada, David Muñoz-Rodriguez, Cesar Vargas-Rosales, and James Fangmeyer. “Evolution of Indoor Positioning Technologies: A Survey”. 29 marzo 2017

3 R. Faragher and R. Harle, "Location Fingerprinting With Bluetooth Low Energy Beacons," in *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 11, pp. 2418-2428, Nov. 2015, doi: 10.1109/JSAC.2015.2430281.

4 Emilio Sansano, Raúl Montoliu, Óscar Belmonte and Joaquín Torres-Sospedra. “Indoor Positioning and Fingerprinting: The R Package ipft” in The R Journal Vol. 11/01, June 2019

5 Christian Frost, Casper Svenning Jensen, Kasper Søe Luckow, Bent Thomsen, René Hansen. “Bluetooth Indoor Positioning System Using Fingerprinting” in Third International ICST Conference, MOBILIGHT 2011, Bilbao, Spain, May 9-10, 2011

6 https://neo4j.com/docs/pdf/neo4j-getting-started-4.1.pdf

7 https://neo4j.com/docs/cypher-manual/current/