POLITECNICO DI TORINO

Collegio di Ingegneria Gestionale

Corso di Laurea
in Ingegneria Gestionale

Tesi di Laurea

RaspBeacon

Relatore

Prof. Luigi De Russis

Candidato

Fabio Tessarollo

Ottobre 2018

Indice

Esplorazione delle funzionalità dei Beacon Bluetooth con il Raspberry PI

Introduzione

Il progetto trattato riguarda la realizzazione di un programma scritto in Python che eseguito sul Raspberry PI, un computer delle dimensioni del palmo di una mano, restituisce all'utente informazioni riguardo ai dispositivi Bluetooth circostanti, in particolare riguardo ai Beacon, piccoli "Bluetooth Low Energy Devices", vale a dire emettitori di segnali broadcast in Bluetooth a basso consumo energetico. Maggiore attenzione è stata prestata nel calcolo della distanza tra Raspberry e Beacon sulla base della qualità di segnale rilevata.

Raspberry PI

Questo mini computer è stato pensato appositamente per gli sviluppatori. È ingegnerizzato in modo tale da essere il più semplice e piccolo possibile, pur mantenendo tutte le funzionalità di un computer. Tutto quello che serve per usarlo è uno schermo e il suo alimentatore. Il suo sistema operativo nativo è Linux, e con Linux è stato svolto questo progetto. In secondo luogo si è fatto uso di "LXTerminal", che come tutti i terminali virtuali consiste in un programma che permettere di sfruttare le diverse funzionalità di un computer in base a input testuali dell'utente. Nel progetto lo script in Python realizzato è stato eseguito da terminale con il comando:

```
sudo python /home/pi/RaspBeacon/GettingRSSI.py
```

Sudo: grazie a questa espressione otteniamo le autorizzazioni come amministratore per eseguire i comandi a seguire.
Python: utile a indicare al computer che stiamo per accedere a un file eseguibile scritto in Python.
/home/pi/RaspBeacon/GettingRSSI.py: il percorso all'interno dei file di sistema per accedere al file da eseguire. *py* è l'estensione che indica codici scritti in Python, *GettingRSSI* è il nome del file e *RaspBeacon* è la cartella che lo contiene.

Raspberry PI 3, il modello utilizzato nel progetto

Alla pressione del tasto invio quindi il programma viene eseguito e ne vengono stampati a schermo i risultati, ovvero dati riguardanti i dispositivi Bluetooth circostanti. Sempre da terminale poi il programma sviluppato accetta altri dati specifici come input testuale, ovvero le

scelte dell'utente che usa l'applicazione. Tra gli altri software del Raspberry usati in questo lavoro in fine troviamo l'IDE (Integrated Development Enviroment) per Python "Thonny", utile a modificare o aggiungere linee di codice allo script.

Beacon Bluetooth

I Beacon Bluetooth sono stati pensati per comunicare con le persone in base alla loro posizione, vale a dire inviare un messaggio a tutte le persone che sono in loro prossimità. Questi piccoli dispositivi inviano ripetutamente delle stringhe alfanumeriche tramite Bluetooth a tutti i ricevitori che si trovano nel loro campo d'azione. Gli smartphone delle persone nei paraggi, se abilitati alla ricezione di questi segnali, capteranno quindi i dati inviati. Questi dati sono la chiave che un'applicazione apposita userà per scaricare dalla rete le informazioni relative al Beacon mittente. Le informazioni così ottenute sono il messaggio che chi ha posizionato e configurato il Beacon intende trasmettere a tutte le persone che gli si avvicinano.

Scendiamo più nei dettagli con un esempio:

Una catena di negozi vuole notificare ai suoi clienti delle offerte speciali per alcuni prodotti facendo uso dei Beacon. Per ottenere ciò verrà assegnato uno stesso indirizzo, un codice detto UUID, a tutti i Beacon da usare, che identifichi univocamente la catena di negozi, e due codici, il major e il minor, che indichino rispettivamente in quale negozio e in quale zona del negozio ci troviamo. I Beacon posizionati vicino a un certo tipo di prodotti invieranno il minor relativo all'offerta di questi, il major per indicare il negozio e l'UUID per indicare la catena e distinguerla da altre società che facciano uso di Beacon. Grazie a questi codici si può quindi risalire a una posizione specifica, ovvero quella occupata dal Beacon secondo la configurazione del suo utilizzatore. Ogni posizione specifica avrà un suo messaggio. I codici citati sono trasmessi a un server dall'applicazione per mezzo dello smartphone che li riceve. Il server risponderà inviando l'informazione relativa a quei codici. Quello che succede a livello pratico è che potenziali clienti dotati dell'app su smartphone con Bluetooth acceso, riceveranno una notifica sulla disponibilità di un certo prodotto esposto nelle vicinanze. Questo è solo un esempio di una possibile applicazione dei Beacon Bluetooth, che possono essere usati in molteplici altri contesti di diversa utilità. Un'altra applicazione dei dispositivi è nei musei, dove si può fare in modo che i visitatori ricevano sul telefono informazioni sull'opera che stanno guardando semplicemente avvicinandosi ad essa. Nel caso del museo il major potrebbe distinguere i piani mentre il minor le singole opere. Grazie all'identificativo diviso in 3 parti quindi, si può creare una gerarchia di Beacon molto utile soprattutto quando se ne usano molti in una stessa struttura.

- UUID: 16 byte, ad es. "B9407F30-F5F8-466E-AFF9-25556B57FE6D".
- Major: 2 byte, quindi un numero compreso tra 1 e 65.535.
- Minor: uguale al major.

Beacon della Estimote

I Beacon sono stati concepiti per rimanere sempre accesi e sono alimentati da una batteria interna. Implementano la tecnologia BLE, Bluetooth Low Energy, grazie alla quale possono trasmettere in Bluetooth con un consumo energetico relativamente basso, tant'è che hanno un tempo di vita di circa 3 anni, la cui fine è segnata dal consumo totale della batteria.

BLE è un protocollo diverso dal classico Bluetooth, quindi scansionando i dispositivi nei paraggi dalle impostazioni del nostro smartphone i Beacon non verranno segnalati. Le frequenze radio utilizzate però sono sempre le stesse (2,4 GHz), infatti anche l'antenna dei dispositivi, che usino BLE o Bluetooth classico, è sempre la stessa e gli smartphone sono abilitati a captare anche questi segnali. A prova di ciò esistono applicazioni smartphone che consentono di trasformare il proprio telefono in un simulatore di BLE, ovvero trasformarlo in un Beacon fittizio. Possiamo dire per tanto che la differenziazione tra i due tipi di Bluetooth è percepita a un livello protocollare più alto.

Inoltre, stabilire una connessione permanente con uno di questi oggetti non è scontato come con un classico dispositivo Bluetooth. Infatti, la tecnologia BLE per i Beacon non funziona secondo un accoppiamento di due dispositivi e lo stabilimento di una connessione persistente tra di essi, ma prevede che uno dei due condivida ininterrottamente in modalità broadcast (chiaramente il Beacon). Trasmissione broadcast significa trasmissione verso tutti, non vengono quindi mandati messaggi con uno specifico destinatario, non c'è nessun accordo preliminare tra le parti e nessuna verifica che i pacchetti di informazioni inviati arrivino integri al destinatario. L'unica connessione con i Beacon è stabilita quando il proprietario deve modificarne le impostazioni di configurazione.

Python

Il linguaggio di programmazione usato nel progetto è Python 2. L'aggiornamento all'ultima versione per Raspberry è stato effettuato da linea di comando, si parla della 2.7.13. Questa stessa versione è stata installata anche sull'altro computer usato nella creazione del programma per poter trasferire il codice in modo pulito. Sarebbe stato possibile anche usare Python 3, ma la versione 2 era già sufficiente allo scopo della tesi.

Bluepy

Per creare uno script che consentisse al mini computer di interpretare i segnali dei Beacon, è stato fatto uso di un modulo Python chiamato "bluepy". Questa libreria contiene delle classi e dei metodi che semplificano molto la creazione di un programma utile a questo obbiettivo. Funziona solo in ambiente Linux e si appoggia su Bluez, programma che per l'appunto

implementa il protocollo Bluetooth in Linux. Difatti per usare bluepy è stato necessario prima installare Bluez, eseguendo

```
sudo apt-get install bluez bluez-hcidump
```

da LXTerminal sul Raspberry. Infine, seguendo le indicazioni dello sviluppatore di bluepy, si è potuto installare bluepy con

```
sudo apt-get install python-pip libglib2.0-dev
sudo pip install bluepy
```

Analizziamo ora le classi della libreria e i relativi metodi e attributi più importanti.

- `Scanner`: Istanze di questa classe sono usate per trovare i dispositivi a bassa energia che stanno trasmettendo.
  - `withDelegate(Delegate)`: Questo metodo associa all'oggetto su cui è chiamato un oggetto delegato, che riceve "callbacks" (richiami) quando sono inviati segnali dai dispositivi. Senza questo passaggio la scansione non può funzionare.
  - `scan([timeout])`: Chiamando questa funzione sull'oggetto Scanner facciamo partire una scansione che terminerà dopo il tempo inserito come parametro. A fine scansione verrà restituita una lista degli oggetti *scanEntry*, che rappresentano i dispositivi.
- `DefaultDelegate`: classe adibita alla ricezione asincrona di segnali Bluetooth, come i messaggi inviati dai Beacon. Come consigliato dallo sviluppatore di bluepy, si deve creare un oggetto delegato con questa classe e sovrascrivere i suoi metodi a seconda delle nostre esigenze.
  - `handleDiscovery(`*`scanEntry`*`, `*`isNewDev`*`, `*`isNewData`*`)`: Questo è l'unico dei metodi sovrascritti nel codice sviluppato. La sua funzione è gestire la ricezione di segnali da nuovi dispositivi o nuovi segnali dagli stessi. Nell'app i rinnovamenti dei segnali, rilevati con *isNewData*, non sono stari presi in considerazione.
- `ScanEntry`: Gli oggetti di questa classe, come già accennato, sono un riferimento ai dispositivi che ci permette di risalire alle loro caratteristiche. La libreria non prevede un costruttore per questa classe, perché i suoi oggetti devono essere creati solo a seguito di un effettivo rilevamento eseguito con un oggetto *Scanner* e non dallo sviluppatore.
  - `addr`: Indirizzo MAC. Attributo di tipo stringa alfanumerica (numeri esadeciamali separati da ":")
  - `rssi`: Numero intero negativo. La potenza di segnale del dispositivo captato sul quale sarà basato il calcolo della distanza.

- `connectable`: Attributo booleano, vero se è possibile stabilire una connessione, falso altrimenti. In fase di sperimentazione questa proprietà dell'oggetto scanEntry ha assunto il valore *False* quando riferita al Beacon, come ci si aspettava.
- `getScanData()`: Questo metodo, come l'attributo *connectable* non è usato nell'applicazione finale ma è stato interessante provarlo. Quello che restituisce sono delle tuple consistenti in tipo (espresso tramite un codice), descrizione e valore di alcuni campi dei pacchetti inviati dal dispositivo, dati di servizio. Esempi di tuple ottenute col Raspberry sono qui riportati:

```
(1, 'Flags', '04')
(3, 'Complete 16b Services', '0000feaa-0000-1000-8000-00805f9b34fb')
(22, '16b Service Data', 'aafe10ee037477697474657200657374696d6f7465')
```

Con il primo codice della tupla si può risalire alla descrizione del dato di servizio secondo le convenzioni del protocollo Bluetooth, specificate in una tabella sul sito bluetooth.com (per esempio, come nell'output riportato sopra, al codice 22 corrispondono i codici a 16 bit). Tra i possibili valori restituiti da questo metodo c'è il già citato UUID.

La libreria bluepy è utilizzabile anche da linea di comando. Di seguito viene riportato un esempio di scansione:

```
pi@raspberrypi:~ $ sudo blescan
Scanning for devices...
    Device (new): 08:57:28:1a:1a:28 (random), -59 dBm (not connectable)
      Manufacturer: <06000109200233f89f1045f726b1664e53eca7bdfd5591a91df30182ef>
    Device (new): ec:96:b8:de:39:fa (random), -67 dBm (not connectable)
      Flags: <06>
      Manufacturer: <4c000215e1f54e021e2344e09c3d512eb56adec900640064b9>
    Device (new): e8:0f:30:0f:c3:74 (random), -59 dBm
      Flags: <06>
      Complete 16b Services: <0000fe9a-0000-1000-8000-00805f9b34fb>
      16b Service Data: <9afe00c5d6680cca9d7be42fc64d2a6b2c632cc09440>
    Device (new): c8:69:cd:d1:f7:0e (public), -61 dBm
      Flags: <1a>
      Manufacturer: <4c0009060307c0a80167>
    Device (new): e9:10:31:10:c4:75 (random), -65 dBm (not connectable)
      Flags: <04>
      Complete 16b Services: <0000feaa-0000-1000-8000-00805f9b34fb>
      16b Service Data: <aafe10ee037477697474657200657374696d6f7465>
```

Nel contenuto del campo "16b Service Data" è stato riconosciuto l'UUID di un Beacon.

PyCharm

PyCharm è l'IDE utilizzato per lo sviluppo del codice, usato su un computer diverso dal Raspberry, per poter operare più comodamente. Gran parte del corpo del codice infatti è stato scritto su questa piattaforma per via dei vantaggi che offre. Questo software identifica e segnala alcuni degli errori presenti nel codice e anticipa allo sviluppatore i metodi chiamabili sui diversi oggetti, velocizzando la scrittura. Inoltre, lavorare sul Raspberry non è comodo come su un computer normale, per motivi di prestazioni o "comodità personale".

Il sistema operativo del computer su cui è stato installato PyCharm però è Windows, che non è compatibile con la libreria bluepy, come spiegato nella prima riga del file "README" creato dal suo sviluppatore. Di conseguenza l'unico dispositivo usato per l'esecuzione del codice è stato il Raspberry, come d'altronde è richiesto nella consegna del progetto finale. Ciò ha implicato la necessità di dover trasferire il codice dal computer Windows al Raspberry ogni volta che serviva verificare il funzionamento delle modifiche effettuate sullo script. Per effettuare questi trasferimenti inizialmente si è pensato a Github, sito sul quale è stato caricato il codice dal computer per poi riottenerlo tramite download sul Raspberry con il comando da terminale *git clone* seguito dalla URL del *repository*. Successivamente però si è rivelato più pratico e veloce trasportare il codice tramite chiavetta USB. Un'altra opzione poteva essere quella di eseguire il codice in remoto. Si tratta di una funzionalità di PyCharm che permette di eseguire il codice scritto in esso su un'altra macchina. Per fare ciò però era necessario installare una *local virtual machine* che simulasse Linux all'interno di Windows, per questo si è preferito evitare questa strada.

Si è dunque accettato di dover impiegare relativamente molto tempo ogni volta che era necessario eseguire il codice per testare il suo funzionamento. Questo scenario ha impedito un approccio a tentativi, imponendo di lavorare attentamente e senza errori, per evitare di perdere tempo a effettuare troppi trasferimenti del codice. Ad ogni modo, nel caso di imperfezioni che non richiedessero modifiche consistenti, si è proceduto con correzioni tramite l'IDE del Raspberry.

Calcolo della distanza con RSSI

RSSI sta per Recieved Signal Strength Indication, una grandezza scalare che indica la qualità del segnale radio. La prima utilità della valutazione di questa grandezza è capire quanto efficiente può essere la trasmissione. Chiaramente, un alto valore significherà maggiore velocità di trasmissione e meno probabilità di errori, mentre bassi valori implicheranno che le nostre trasmissioni potrebbero non andare a buon fine. Nel caso dei Beacon questo valore è molto importante per capire qual è quello a cui siamo più vicini. Solo con questa misurazione infatti possiamo delimitare con precisione gli spazi dei diversi Beacon. Inoltre si può anche fare in modo che l'informazione captata da un Beacon sia presa in considerazione solo se gli RSSI sono

più alti di un certo valore, quindi idealmente solo se ci avviciniamo quanto basta al Beacon. Così facendo restringiamo il campo d'azione del Beacon, fattore modificabile in fase di configurazione.

Tramite gli RSSI infatti è anche possibile ottenere un'indicazione sulla distanza del dispositivo tracciato. Lo scopo del progetto è proprio capire la distanza tra Raspberry e Beacon (o altro BLE device) in base agli RSSI rilevati sul Raspberry. La distanza così percepita però è molto inaccurata, la qualità del segnale infatti è una variabile connessa a molti altri fattori oltre che alla distanza. Nell'ambiente di una casa per esempio, ci possono essere molti ostacoli come muri o mobili che abbassano la potenza di segnale e, a fronte di un rilevamento, leggeremo una distanza più alta di quella reale. Ad aumentare la variabilità inoltre ci sono riflessioni del segnale stesso e interferenze di alti segnali. Per ottenere maggiore precisione l'ideale sarebbe quindi operare in uno spazio ampio e privo di ostacoli.

Nel progetto gli RSSI sono misurati in decibel milliwatt (dB_{mW}, abbreviati "dBm"), calcolati come logaritmo del rapporto di due potenze espresse in milliwatt. Questo valore, inteso come potenza captata, è sempre negativo e proporzionale al segnale. I valori più alti misurati in fase di sperimentazione si sono aggirati intorno ai -15 dBm, misurati con il Beacon a meno di un centimetro dal Raspberry. I più bassi introno ai -90 dBm, quando la distanza era di una decina di metri.

Si potrebbe pensare che esista una formula unica per ottenere la distanza con la sola conoscenza degli RSSI, invece bisogna considerare un'altra variabile, vale a dire i dispositivi di trasmissione e ricezione. Questo perché con antenne diverse avremo diverse percezioni del segnale. Esiste però una formula generica con cui ottenere la formula per due dispositivi specifici:

Se $RSSI > Txpower$

$$d = A\left(\frac{RSSI}{Txpower}\right)^{B} + C$$

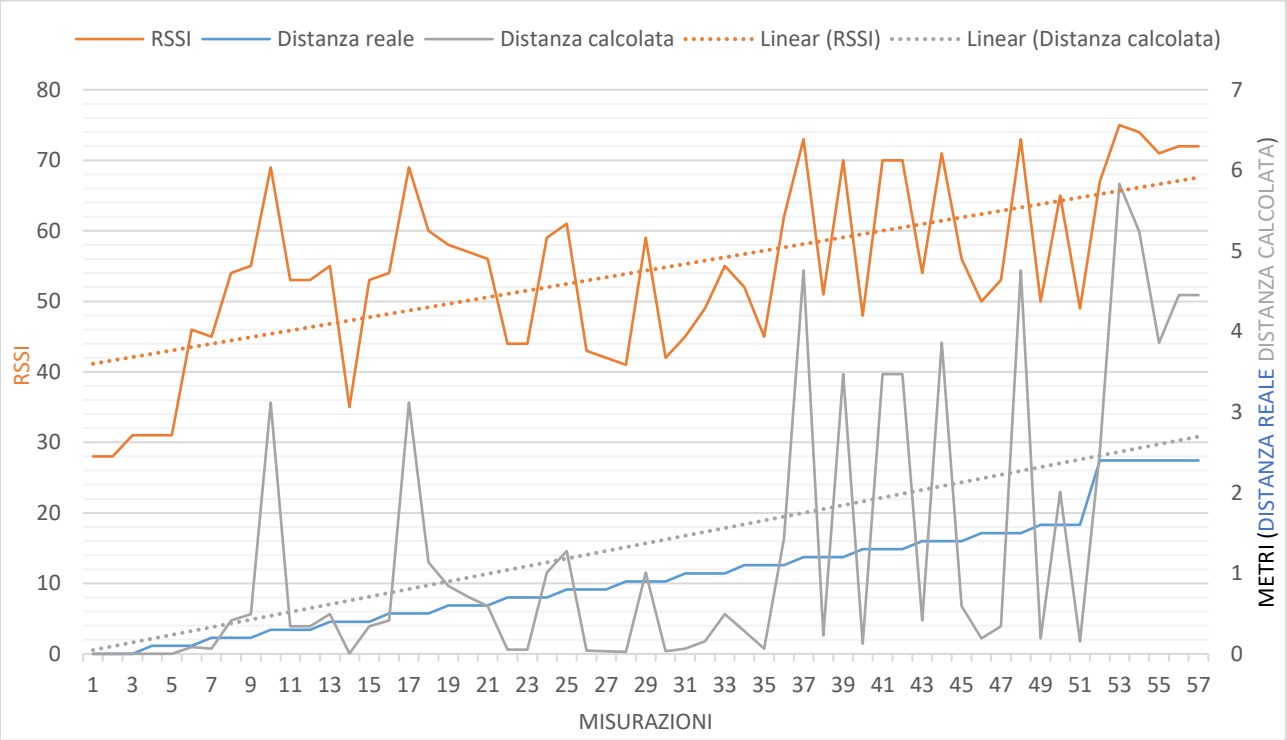
Altrimenti

$$d = \left(\frac{RSSI}{Txpower}\right)^{10}$$

In cui i coefficienti *A*, *B* e *C* sono da trovare risolvendo un sistema le cui equazioni conoscono i diversi valori di *d* e RSSI, calcolati sperimentalmente con un programma che potrebbe essere, per esempio, lo script oggetto di questa tesi. *Txpower* nella formula è un valore uguale agli RSSI misurati a un metro di distanza. In realtà nelle impostazioni di configurazione del Beacon con questo termine ci si riferisce alla potenza di trasmissione del Beacon, impostata in questo caso

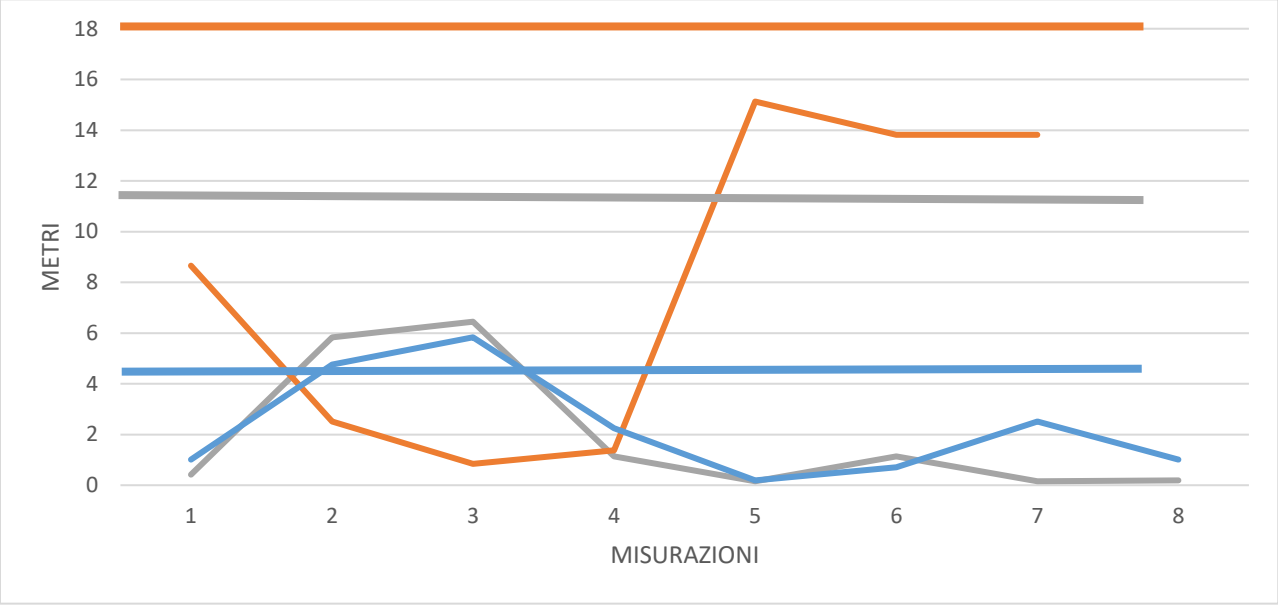
a 4 dBm. Secondo quanto trovato in rete, per Raspberry e Beacon si possono ottenere dei risultati accettabili con i parametri: A = 0.89976, B = 7.7095, C = 0.111 e *Txpower* = -59 dBm.

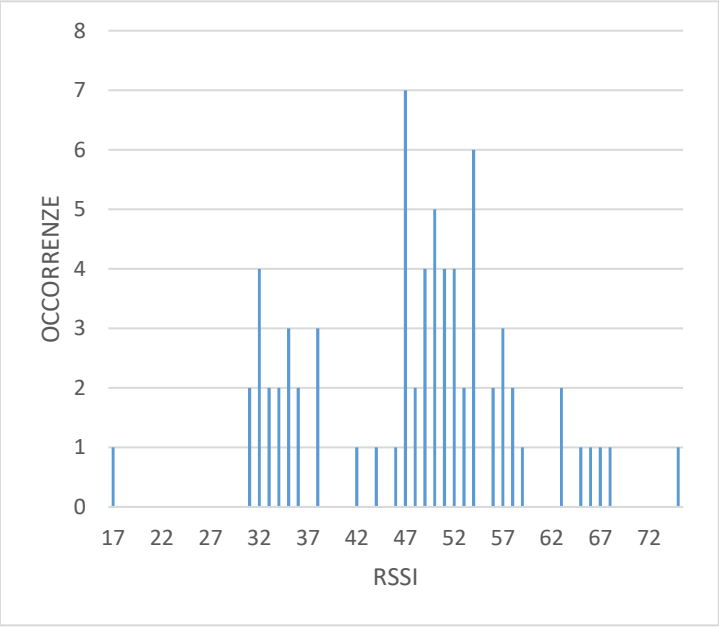
Il grafico sottostante è stato ottenuto plottando su Excel i risultati dei calcoli sulla distanza del Raspberry rispetto a un Beacon Estimote. La distanza tra i due, nelle prime misurazioni (asse delle ascisse) era di pochi centimetri. Successivamente il Beacon è stato allontanato progressivamente, di 10 cm ogni 3 scansioni. È facile notare osservando il grafico quanto le linee di RSSI e distanza calcolata (chiaramente proporzionali tra loro) siano ben distanti come andamento rispetto alla linea azzurra, che rappresenta l'allontanamento approssimativamente lineare del Beacon. Questo grafico infatti dimostra bene l'imprecisione nel rilevamento della distanza, evidenziando anche come questa imprecisione aumenti notevolmente con l'aumentare della distanza, basti osservare i valori degli RSSI a seguito della 35esima misurazione. Ma, le linee rette a puntini arancione e grigia, le *trend line* lineari di RSSI e distanza calcolata ottenute con Excel, d'altro canto dimostrano che il trend è in crescita e anche abbastanza parallelo alla linea della distanza reale, com'è giusto che sia.



Dal grafico si capisce anche che alcuni valori sono molto lontani dalla media. Un calcolo della distanza effettuato con la rimozione di questi valori estremi e con la media dei rimanenti RSSI potrebbe portare a risultati più precisi.

In questo secondo grafico il Beacon è stato posizionato a 6, 12 e 18 metri dal Raspberry. Le distanze calcolate dal Raspberry sono indicate rispettivamente con le linee blu, grigia e arancione. Le linee orizzontali sono le rispettive distanze reali. Si deduce una notevole imprecisione.





Quest'ultimo grafico ci può dare un'idea sull'instabilità degli RSSI. Sulle ascisse possiamo leggere i diversi valori letti con il Beacon posizionato a un metro di distanza. Per la precisione sono stati effettuati 71 rilevamenti, la cui media è risultata essere -46 dBm, ben 13 dBm in più rispetto alla *txPower* che ci si aspettava (-59 dBm). In seguito a questo esperimento quindi il programma è stato testato usando come *txPower* della formula -46 dBm, ma i risultati non sono affatto migliorati.

Primo approccio

Prima di scrivere le prime linee di codice, è stato considerato utile provare a interagire con i Beacon attraverso il terminale del Raspberry con dei comandi preesistenti appositi che richiedono solo l'installazione di Bluez. Il primo di questi è:

```
sudo hcitool lescan
```

Hcitool serve a configurare connessioni Bluetooth, operazioni effettuabili solo se precedute da *sudo*. Aggiungendo *lescan*, Low Energy scan, richiediamo di scansionare i dispositivi Bluetooth attivi a bassa energia raggiungibili dal Raspberry. Premendo invio il terminale ci mostra uno dopo l'altro gli indirizzi MAC di questi dispositivi.

```
LE Scan ...
E9:10:31:10:C4:75 (unknown)
C8:69:CD:D1:F7:0E (unknown)
C8:69:CD:D1:F7:0E (unknown)
08:57:28:1A:1A:28 (unknown)
EC:96:B8:DE:39:FA (unknown)
```

Un secondo comando,

```
gatttool -I -b MAC-address
```

dovrebbe permettere di stabilire una connessione con il dispositivo di cui viene specificato l'indirizzo. Provando a effettuare questa operazione con il Beacon però il terminale ha sempre segnalato errore:

```
^Cpi@raspberrypi:~ $ gatttool -I -b E9:10:31:10:C4:75
[E9:10:31:10:C4:75][LE]> connect
Attempting to connect to E9:10:31:10:C4:75
Error: connect error: Connection refused (111)
```

Questo perché i Beacon accettano una connessione solo sotto autorizzazione, vale a dire con l'identificazione tramite username e password del loro utilizzatore (ovvero, nell'esempio fatto in precedenza, l'esercente della catena di negozi). Questa connessione, in un utilizzo normale di questi oggetti, teoricamente è stabilita solo al momento della configurazione iniziale, quando vengono decise UUID, major, minor e altre impostazioni. Successivamente, il Beacon condividerà con tutti i passanti il messaggio scelto fino all'esaurimento della sua batteria. Connettersi per cambiare le impostazioni invece sarà un'operazione riservata a chi li ha posizionati. Se qualcuno provasse a connettersi come è stato tentato con il comando *gatttool*, il Beacon si disconnetterà dopo 10 secondi, il tempo d'attesa per l'autenticazione del suo possessore.

La libreria bluepy non contempla nessun metodo che preveda la connessione a un Beacon attraverso username e password dell'account Estimote. Quindi è inutile tentare di usarla per ottenere una connessione. Nonostante questo sono state effettuate delle prove usando la classe *Peripheral*, che hanno portato al risultato previsto segnalando il seguente errore:

```
bluepy.btle.BTLEException: Failed to connect to peripheral
e9:10:31:10:c4:75, addr type: public
```

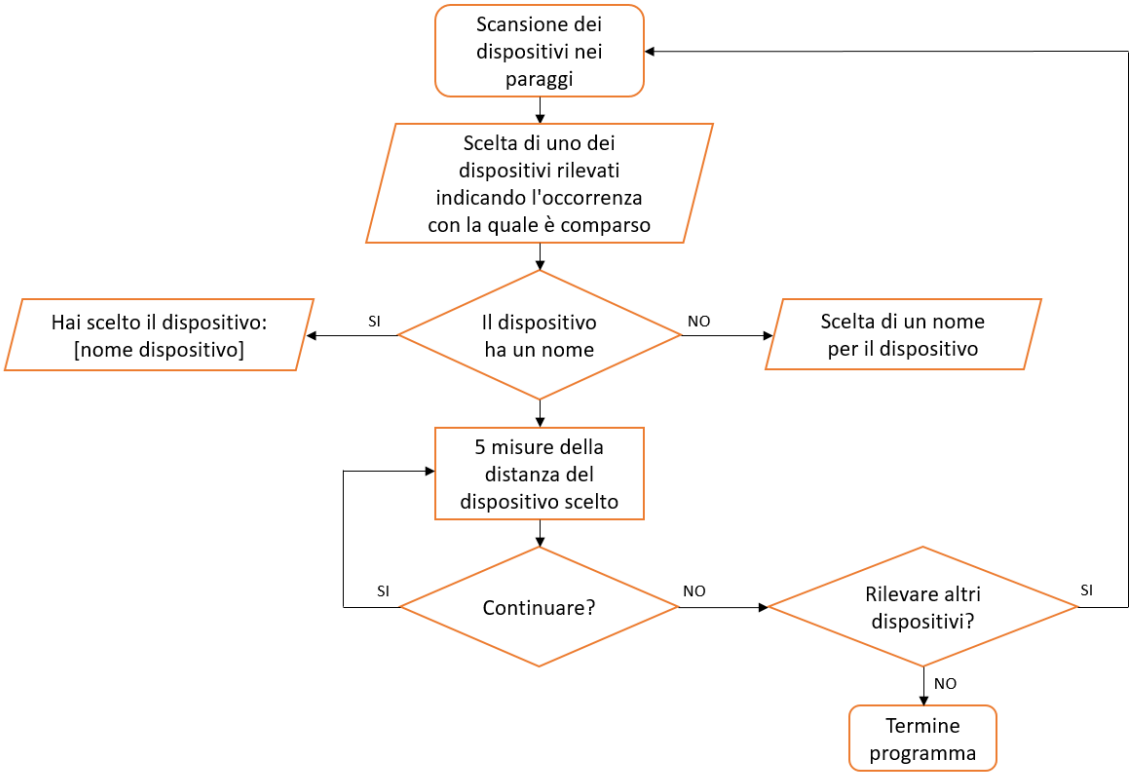
L'unica connessione stabilita con il Beacon quindi è stata ottenuta attraverso uno smartphone con l'app della casa produttrice, chiamata con il nome dell'azienda stessa: Estimote. Dopo essersi autenticati con essa è stato possibile configurare diverse funzionalità del Beacon e scoprire informazioni a riguardo, come la durata della batteria rimanente, la versione del *firmware*, la frequenza di trasmissione dei segnali e chiaramente UUID, major e minor.

Altre prove sono state fatte usando PyBeacon, un'altra app scritta in Python trovata su Github che non è basata su bluepy. I Beacon possono inviare una URL al posto dei codici citati, che potrà essere usata dal browser degli smartphone che la riceveranno per collegarsi al sito dell'indirizzo web. PyBeacon, con un apposito comando deciso dal suo sviluppatore (*PyBeacon –scan*) permette di rilevare queste URL. Il comando è stato provato e ha restituito correttamente la stessa URL visibile dalle impostazioni di configurazione dello stesso Beacon con l'app Estimote.

Siccome con quest'app non è possibile visualizzare gli indirizzi MAC, per distinguere quali di quelli rilevati appartenessero al Beacon, sono state eseguite due scansioni. Una prima con il Beacon in prossimità, una seconda dopo averlo portato fuori dal raggio d'azione del Raspberry. Gli indirizzi che la prima scansione aveva in più rispetto alla seconda sono stati così riconosciuti come appartenenti al Beacon. L'operazione è stata eseguita più volte per sicurezza.

Il programma finale, lato utente

L'esecuzione dell'applicazione, rappresentata in un flow chart. Esempi di output.



La scansione iniziale avviene nell'arco di 5 secondi e stampa a video un elenco dei dispositivi nel raggio d'azione del Raspberry con il loro indirizzo MAC o con il loro nome se glien'è stato assegnato uno in precedenza.

```
1) Dispositivo rilevato c8:69:cd:d1:f7:0e
2) Dispositivo rilevato 26:7f:2d:78:96:5f
3) Dispositivo rilevato e9:10:31:10:c4:75
4) Dispositivo rilevato e8:0f:30:0f:c3:74
5) Dispositivo rilevato ec:96:b8:de:39:fa
```

Digitando il numero dell'occorrenza l'utente sceglie quale dispositivo iniziare a tracciare. Nell'esempio il dispositivo scelto, il 3, non ha un nome quindi viene chiesto di assegnargliene uno come input da tastiera, in questo caso "BeaconVerde"

```
Selezionare uno dei dispositivi rilevati indicando il numero della sua
occorrenza: 3
Selezionato dispositivo con indirizzo: e9:10:31:10:c4:75
Scegliere un nome per il dispositivo scelto: BeaconVerde
```

```
Nome assegnato: BeaconVerde
```

A questo punto quindi vengono effettuate in successione 5 scansioni. Nell'esempio considerato il Beacon è stato allontanato e il programma lo ha correttamente rilevato nelle ultime scansioni. Tra una scansione è l'altra intercorrono circa 4 secondi. Questa è la fase di "tracciamento" e dopo ogni ciclo di scansioni viene richiesto se si vuole proseguire a tracciare.

```
Inizio della scansione
RSSI=-68 dB, Distanza=2.799256 m
RSSI=-61 dB, Distanza=1.274439 m
RSSI=-64 dB, Distanza=1.795564 m
RSSI=-63 dB, Distanza=1.602966 m
RSSI=-68 dB, Distanza=2.799256 m
Continuare a tracciare questo dispositivo? Digitare Si/No: Si
RSSI=-72 dB, Distanza=4.287841 m
RSSI=-78 dB, Distanza=7.852880 m
RSSI=-79 dB, Distanza=8.651812 m
RSSI=-87 dB, Distanza=18.077636 m
RSSI=-86 dB, Distanza=16.545591 m
```

Come spiegato nel flow chart, se l'utente rifiuta di continuare a tracciare lo stesso dispositivo, può scegliere di far ripartire l'algoritmo. In questo caso, a una seconda scansione di tutti i dispositivi, se quelli precedentemente analizzati sono ancora nei paraggi vengono segnalati con il nome scelto. Nell'esempio infatti si può leggere "BeaconVerde".

```
Continuare a tracciare questo dispositivo? Digitare Si/No: No
Rilevare un nuovo dispositivo? Si/No: Si
6) Dispositivo rilevato 47:e1:c5:ba:47:05
7) Dispositivo rilevato 26:7f:2d:78:96:5f
8) Dispositivo rilevato c8:69:cd:d1:f7:0e
9) BeaconVerde
10) Dispositivo rilevato ec:96:b8:de:39:fa
11) Dispositivo rilevato e8:0f:30:0f:c3:74
```

L'occorrenza di rilevamento cambia da scansione a scansione per quanto il dispositivo sia nella stessa posizione per gli stessi motivi legati all'imprecisione della distanza.

Lo script *GettingRSSI.py*

Il codice, spiegato in modo sequenziale.

La prima riga di codice è il comando per importare la libreria bluepy, o più precisamente le classi da usare nel codice appartenenti alla libreria: *Scanner* e *DefaultDelegate*.

```
from bluepy.btle import Scanner, DefaultDelegate
```

Creazione di due puntatori generici a vettori. In Python si possono dichiarare variabili senza specificarne il tipo. In questo caso si tratta di due array, che andranno a indicare, nell'arco dell'esecuzione del codice, il primo tutti i dispositivi rilevati con le scansioni, il secondo quelli a cui l'utente ha dato un nome. Si tratta di variabili globali, che potranno essere richiamate in qualsiasi altra parte del codice.

```
discoveredDevices = []
namedDevices = []
```

Definizione di una classe "Device". Questa classe ha lo scopo di collegare tra loro informazioni riguardo ai dispositivi. La funzione *init* è il costruttore degli oggetti in Python, con cui vengono creati gli oggetti della classe in cui è definito. La creazione avviene tramite l'assegnazione di alcuni attributi: "devi", ovvero l'oggetto scanEntry della libreria bluepy; "number", un numero univoco di un dispositivo, occorrenza di rilevamento durante la scansione; e il nome, "name", che al momento della creazione è "default" per tutti, ma sarà successivamente modificabile grazie alla funzione "setname". "setnewnumber" invece verrà usata per rimpiazzare l'attributo "number" con un nuovo numero quando un oggetto scanEntry viene scansionato una seconda volta.

```
class Device:
    def __init__(self, devi, number):
        self.devi = devi  # riferimento all'oggetto scanEntry
        self.number = number  # occorrenza
        self.name = "default"

    def setname(self, name):
        self.name = name

    def setnewnumber(self, number):
        self.number = number
```

La funzione "getdevice" è stata creata per poter risalire a un oggetto "Device" tra quelli nominati dato il suo indirizzo.

```
def getdevice(addr):
    for device in namedDevices:
        if device.devi.addr == addr:
            return device
```

La classe ScanDelegate è una classe che fa riferimento alla classe DefaultDelegate di bluepy. Istanze di questa classe hanno lo scopo di ricevere e gestire la ricezione dei segnali. Questa in particolare viene usata nella prima scansione, quella per trovare la lista di dispositivi tracciabili. La variabile numerica “i” è usata per contare i diversi dispositivi rilevati. È in questa classe che vengono creati gli oggetti Device e aggiunti alla lista dei dispositivi scoperti, contenuti nell’array “discoveredDevices”. Da notare che nel caso in cui questi siano già stati nominati, non vengono ricreati, ma recuperati tra quelli creati in precedenza attraverso il loro indirizzo con “getdevice”.

```
class ScanDelegate(DefaultDelegate):
    i = 1

    def __init__(self):
        DefaultDelegate.__init__(self)

    def handleDiscovery(self, dev, isNewDev, isNewData):
        if isNewDev:
            flag = False
            if not namedDevices:
                print"%d) Dispositivo rilevato" % ScanDelegate.i, dev.addr
                discoveredDevices.append(Device(dev, ScanDelegate.i))
            else:
                for device in namedDevices:
                    if device.devi.addr == dev.addr:
                        flag = True

                if flag:
                    print"%d) %s" % (ScanDelegate.i,getdevice(dev.addr).name)
                    getdevice(dev.addr).setnewnumber(ScanDelegate.i)
                    discoveredDevices.append(getdevice(dev.addr))
                else:
                    print"%d) Dispositivo rilevato" % ScanDelegate.i,dev.addr
                    discoveredDevices.append(Device(dev, ScanDelegate.i))

            ScanDelegate.i += 1
```

Una seconda classe della stessa tipologia della precedente, chiamata “ScanDelegateTracking”, è stata scritta per le scansioni di rilevamento di un singolo dispositivo. In questa classe non viene effettuata nessuna stampa né creato alcuno oggetto esplicitamente.

```
class ScanDelegateTracking(DefaultDelegate):

    def __init__(self):
        DefaultDelegate.__init__(self)
```

```
    def handleDiscovery(self, dev, isNewDev, isNewData): pass
```

Troviamo ora la funzione per calcolare la distanza. Il parametro passato sono gli RSSI, lasciati in negativo. Su questi viene applicata la formula per restituire la distanza in metri analizzata in precedenza.

```
def calcoladistanza(rssi):
    txpower = -59   # rssi a un metro di distanza
    if rssi == 0:
        return -1
    else:
        ratio = rssi*1.0 / txpower
        if ratio < 1:
            return ratio ** 10
        else:
            return 0.89976 * ratio**7.7095 + 0.111
```

Quindi manca solo la creazione degli oggetti di tipo Scanner. Il primo, “scanner” è creato per la prima scansione ed è associato a “ScanDelegate”. Il secondo, “scannerTracking”, sarà contenuto in un ciclo allo scopo di fare più rilevamenti consecutivi di uno stesso dispositivo e sarà associato a “ScanDelegateTracking”.

Da notare quindi che in questo programma non viene stabilita una connessione con un certo dispostivo per rilevarne la distanza, come potrebbe apparire lato utente. In realtà il tipo di scansione è sempre lo stesso, ovvero rivolta verso tutte le antenne Bluetooth nella zona. Questa scansione sarà semplicemente associata alla condizione di considerare solo i risultati relativi al dispositivo da tracciare, distinguibile grazie al suo indirizzo. Quello che avviene quindi è che il Raspberry percepisce sempre tutti i segnali, ma al livello dell’applicazione verranno prelevati solo quelli provenienti dalla periferica Bluetooth che si vuole tracciare.

```
scanner = Scanner().withDelegate(ScanDelegate())
scannerTracking = Scanner().withDelegate(ScanDelegateTracking())
```

A questo punto inizia il vero e proprio algoritmo spiegato nel flowchart. Il tutto è contenuto in una funzione ricorsiva per consentire al programma di tornare all’inizio facilemente quando necessario. Il primo comando è una scansione di 5 secondi, si tratta della prima scansione dell’algoritmo, atta ad individuare i dispositivi disponibili. “selectednumber” è una variabile che referenzierà la scelta dell’utente.

```
def ricorsiva():
    scanner.scan(5.0)
    named = False
    selectednumber = input("Selezionare uno dei dispositivi rilevati
```

```
indicando il numero della sua occorrenza: ")
    selecteddevice = None
```

Per poter risalire a quale dispositivo l'utente ha scelto e al relativo oggetto Device, è sufficiente ciclare la lista dei dispositivi scoperti, e identificare l'unico che avrà l'attributo "number" uguale al numero scelto in input.

```
for device in discoveredDevices:
    if device.number == selectednumber:
        selecteddevice = device
```

Quindi bisogna dare conferma all'utente della scelta effettuata. Per farlo si è pensato di riconoscere se il dispositivo è tra quelli nominati, e nel caso stampare a video il nome del dispositivo.

```
for device in namedDevices:
    if selecteddevice.devi.addr == device.devi.addr:
        print"Hai scelto %s" % device.name
        named = True
```

Diversamente, se il dispositivo non è tra quelli nominati, si richiede di assegnargli un nome e lo si aggiunge alla lista dei dispositivi nominati.

```
if not named:
    print"Scelto dispositivo con indirizzo: %s" % selecteddevice.devi.addr
    devname = raw_input("Scegliere un nome per il dispositivo scelto: ")
    selecteddevice.setname(devname)
    namedDevices.append(selecteddevice)
    print"Nome assegnato: %s" % devname
```

A queso punto, come segalato a video, iniziano le scansioni consecutive. Il codice che effettua le scansioni è contenuto all'interno di un *while* che si ripeterà finchè il *flag* che lo condiziona diventa vero.

La prima espressione del ciclo è un altro ciclo che attua 5 scansioni consecutive. Questa volta il metodo *scan* è chiamato sull'oggetto scannerTracking, il cui oggetto delegato è il secondo *DefaultDelegate* creato, che non contiene funzioni di stampa e non crea alti oggetti Device come il primo.

Dopo ogni scansione, l'array "devices" viene rimpiazzato con nuovi oggetti *scanEntry* dal metodo *scan*, che ogni volta verranno ciclati per trovare quello scelto dall'utente. Una volta trovato, vengono stampati a video i suoi valori di RSSI e distanza calcolata. Gli RSSI sono un attributo dell'oggetto *scanEntry* chiamato "dev", la distanza sarà ottenuta con la funzione

“calcolaDistanza”. A seguire troviamo l’istruzione *break*, attraverso cui si interrompe il *for* che cicla “devices” , perchè è solo una la corrispondenza cercata e una volta trovata non ha più senso cercare tra le rimanenti non ancora analizzate.

Finite le 5 scansioni consecutive, si richiede se si vuole continuare a scansionare lo stesso dispositivo eseguendo nuovamente le stesse scansioni. Se l’utente digiterà “No” il *flag* diventerà *True* e il *while* si interromperà, con “Si” invece continuerà e si ripeteranno le 5 scansioni.

```
    print"Inizio della scansione"
    stop = False

    while stop is False:
        for n in range(5):  # numero di scansioni
            devices = scannerTracking.scan(4.0)
            for dev in devices:
                if dev.addr == selecteddevice.devi.addr:
                    print("RSSI=%d dB, Distanza=%f m" % (dev.rssi,
calcoladistanza(dev.rssi)))
                    break
        a = raw_input("Continuare a tracciare questo dispositivo? Digitare
Si/No: ")
        if a == "No":
            stop = True
        if a == "Si":
            pass
```

Uscendo dal *while* troviamo un *if* per capire se l’utente vuole continuare a usare il programma o se intende terminarlo. Gli sarà chiesto di decidere in modo analogo a come gli viene chiesto se continuare le scansioni. L’espressione “ricorsiva()” dentro l’*if* fa ricominciare il programma alla linea in cui è presente la dichiarazione della funzione stessa, quindi “def ricorsiva()”, che segna l’inizio dell’algoritmo. Si tratta del caso in cui l’utente non voglia terminare il programma. Con *return* invece si uscirà dalla funzione. In questo caso, siccome non è presente nessun’altra istruzione a seguito della chiamata della funzione “ricorsiva()”, il programma terminerà.

Il compilatore infatti esegue l’algortimo analizzato dopo aver letto tutte le relative istruzioni solo grazie all’ultima riga del codice, in cui la funzione ricorsiva() è chiamata per la prima volta.

```
    b = raw_input("Rilevare un nuovo dispositivo? Si/No: ")
    if b == "Si":
        ricorsiva()
    else:
        return


ricorsiva()
```

Il progetto è *open source*, disponibile (in lingua inglese) su Github all'inidrizzo:

https://github.com/FabioTessarollo/RaspBeacon