

Ingegneria dei Sistemi Software
Adattativi Complessi
Stima della distanza utilizzando tecnologie BLE

Federico Torsello - Matr. 702619

5 ottobre 2016

Indice

1	Introduzione	2
1.1	Vision	2
1.2	Goals	2
1.2.1	<u>Goals principali</u>	2
1.2.2	<u>Goals secondari</u>	3
2	Requisiti	4
2.1	Requisiti funzionali	4
2.2	Requisiti funzionali secondari	4
2.3	Analisi dei requisiti	5
2.3.1	Casi d'uso	5
2.3.2	Scenari	5
3	Accenni alla tecnologia BLE	7
4	Stima della distanza con RSSI	8
4.1	Attenuazione dei segnali elettromagnetici	8
4.2	Received Signal Strength Indicator - RSSI	8
4.3	Calcolo di RSSI	8
4.3.1	Equazione di trasmissione di Friis	9
4.3.2	Conversione della potenza	10
4.3.3	Potenza media a distanza di riferimento d_0	10
4.3.4	Equazione di RSSI	10
4.4	Calcolo della distanza	10
4.5	Problematiche della stima della distanza con RSSI	11
5	Stima della distanza con Arduino	12
5.1	Progetto della stima della distanza con Arduino	12
5.2	Progetto di test della comunicazione seriale PC - Arduino	13
6	UML	16
6.1	Gerarchia delle classi	16
6.1.1	ApplicationActivity	16
6.1.2	Fragment e Observer	17

6.1.3	Observable	17
6.1.4	KFilter	18
6.1.5	KFilterBuilder	18
6.1.6	MyArmaRssiFilter	18
6.1.7	Estimation	19
6.1.8	Device	19
6.1.9	DeviceCardViewAdapter	19
6.1.10	DeviceCardViewAdapter.DeviceViewHolder	20
6.1.11	StatePagerAdapter	20
6.1.12	DeviceConstants	21
6.1.13	KFilterConstants	21
6.1.14	SettingConstants	21
6.1.15	CameraPreviewUtil	22
6.1.16	SaveImageTask	22
6.1.17	ChartUtil	22
6.1.18	UsbUtil	23
6.1.19	FABBehavior	23

Sommario

Capitolo 1

Capitolo 1

Introduzione

Stimare con precisione la distanza che intercorre tra un utente ed un punto target è, secondo molti, una sfida tecnologica. Sempre più sistemi con applicazioni mobile e non necessitano di questa informazione per poter funzionare in modo corretto.

Per stimare la distanza esistono diverse tecnologie più o meno precise e costose. In questo senso l'avvento delle tecnologie IoT (*Internet of things*) ha influito positivamente abbassando il costo dell'hardware necessario, creando community di hobbisti e professionisti, quindi ampliando il numero di librerie software (spesso open source) disponibili e progetti da cui prendere spunto.

1.1 Vision

- Realizzare un sistema software mobile per stimare al meglio la distanza che intercorre tra l'utente e gli iBeacon disposti in un ambiente indoor.
- Utilizzare solo tecnologie open source per realizzare il tutto, ribadendone l'utilità e l'efficienza.

1.2 Goals

1.2.1 Goals principali

1. Sviluppare un'app Android in grado di interagire con degli iBeacon disposti in una stanza.
2. Realizzare un'app compatibile con tutte le API Android 18 e superiori.
3. L'app deve utilizzare i valori RSSI dei iBeacon per determinare la distanza trasmettitore-ricevitore.
4. Implementare diversi filtri per ridurre gli effetti indesiderati del *multi-path fading* sulla stima della distanza.

5. Realizzare e testare un **filtro di Kalman**, un **filtro ARMA** (*Auto Regressive Moving Average*) e un **filtro RunningAverageRssi** per limitare gli effetti indesiderati sopracitati.
6. Visualizzare dei grafici sull'app che in tempo reale descrivano l'andamento della distanza stimata.
7. Curare la *User Experience* realizzando una GUI chiara e responsive utilizzando librerie *com.android.support*
8. Sviluppare l'intero sistema utilizzando GNU/Linux e FOSS (Free and open-source software).

1.2.2 Goals secondari

1. Realizzare un programma C++/Wiring in grado di determinare la distanza percepita da un sensore ultrasonico collegato ad una board Arduino.
2. Realizzare un mini progetto per testare il sensore ultrasonico. Nello specifico si considera un Arduino connesso al PC attraverso la porta USB e una view di feedback per visualizzare a schermo la distanza "reale".
3. Abilitare la comunicazione seriale mediante tecnologia **USB OTG**, facendo diventare lo smartphone un host USB.
4. Implementare la visualizzazione della distanza percepita dall'Arduino direttamente nell'app. L'obiettivo è dare un feedback della reale distanza iBeacon-utente e poterla mettere a confronto con quella stimata utilizzando gli RSSI.

In questo caso si considera un **Arduino connesso allo smartphone** attraverso la porta USB.

Capitolo 2

Requisiti

2.1 Requisiti funzionali

Per la realizzare del sistema si necessita:

- di un utente in possesso di uno smartphone con sistema operativo Android versione 4.3 o superiore;
- uno o più iBeacon;
- un ambiente chiuso (come una stanza) in cui disporre gli iBeacon.

Nello specifico, per non alterare la ricezione dei segnali inviati dagli iBeacon, questi devono essere disposti:

- su pareti regolari a circa 1 metro da terra;
- lontano da apparecchi che emettono onde elettromagnetiche (ad esempio router wifi);
- al sicuro da raffiche di vento;
- in aree in cui non si frappongano oggetti o persone tra lo smartphone e l'iBeacon target.

2.2 Requisiti funzionali secondari

Per poter avere un feedback a schermo della distanza reale utente-iBeacon si necessita di:

- una board Arduino UNO (o superiore);
- un sensore ultrasonico;
- cavi di collegamento maschio-femmina;

- un cavo USB da stampante;
- uno smartphone che supporti la tecnologia **USB OTG (On-The-Go)**¹
- un cavo USB OTG.

2.3 Analisi dei requisiti

2.3.1 Casi d'uso

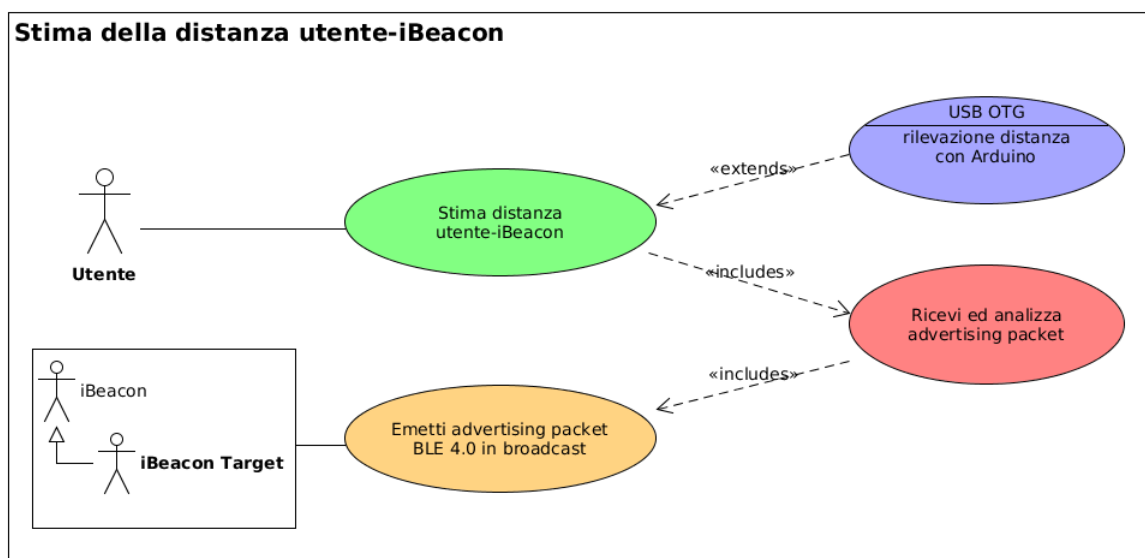


Figura 2.1: Use case - Stima della distanza utente-iBeacon

2.3.2 Scenari

Scenario classico

Nello scenario classico l'utente utilizza l'app per scansionare l'ambiente alla ricerca di iBeacon e leggere delle informazioni a schermo.

Nello specifico in questo scenario:

- Si considera un utente in una stanza con uno smartphone Android in mano.
- La stanza in questione può presentare uno o più iBeacon disposti sui muri ad un'altezza di circa 1 metro dal suolo.

¹USB OTG - https://it.wikipedia.org/wiki/USB_On-The-Go

- Sullo smartphone viene avviata l'app.
- Nel caso in cui la radio Bluetooth dello smartphone fosse spenta o si dovesse spegnere, l'app stessa la accenderà/riaccenderà indicando all'utente l'avvenuto switch.
- L'utente preme su un bottone per avviare la scansione degli iBeacon presenti.
- Se vengono rilevati iBeacon corrispondenti con la lista di indirizzi MAC presente nell'app, allora viene caricata una lista di informazioni a schermo (*friendly name*, immagine, distanza stimata, ecc...).

Scenario classico avanzato

- Una volta che l'utente ha visualizzato l'iBeacon di suo interesse (detto *target*), lo seleziona dalla lista facendo click sull'item corrispondente.
- A questo punto all'utente vengono proposti solo i dettagli relativi al target, isolandoli dagli altri.
- In questa sezione l'utente potrebbe visualizzare un feedback della distanza reale (se tutte le condizioni a contorno sono soddisfatte) in metri.
- Trascinando il dito da destra a sinistra all'utente viene proposta un'altra area ancora più dettagliata dove è possibile mettere a confronto le varie stime della distanza (filtrata, non filtrata e reale se presente), in dei grafici realizzati in tempo reale.

Capitolo 3

Accenni alla tecnologia BLE

Capitolo 4

Stima della distanza con RSSI

4.1 Attenuazione dei segnali elettromagnetici

Il fenomeno dell'**attenuazione** dei segnali elettromagnetici si manifesta nella decrescita della potenza di segnale ricevuto dal ricevitore in relazione all'aumentare della distanza dalla sorgente emittente di tale segnale.

Nota questa relazione, se si conosce la potenza di segnale del trasmettitore P è possibile creare un modello per legare l'**attenuazione di segnale A** e la distanza col ricevitore al fine di stimare la **distanza relativa** trasmettitore-ricevitore d .

$$P = f(d) \tag{4.1}$$

4.2 Received Signal Strength Indicator - RSSI

Per stimare la distanza relativa tra trasmettitore e ricevitore si può sfruttare l'attenuazione di segnale indicata dal valore RSSI il cui calcolo è poco costoso e non necessita di hardware aggiuntivo.

RSSI è utile per smartphone che implementano radio Bluetooth in quanto permette di sviluppare proximity app o localizzazione indoor. Il suo valore viene calcolato in automatico dal sistema operativo dello smartphone.

Purtroppo questo approccio non è particolarmente preciso in quanto l'ambiente influisce sulla potenza del segnale ricevuto e quindi sull'attenuazione reale che rende il valore RSSI oscillante.

4.3 Calcolo di RSSI

Per il calcolo di RSSI (in dBm) si assume:

1. che un segnale radio emesso in nello spazio libero da ostacoli e rumore decade di un fattore d^{-2} , dove d è la distanza relativa trasmettitore-ricevitore

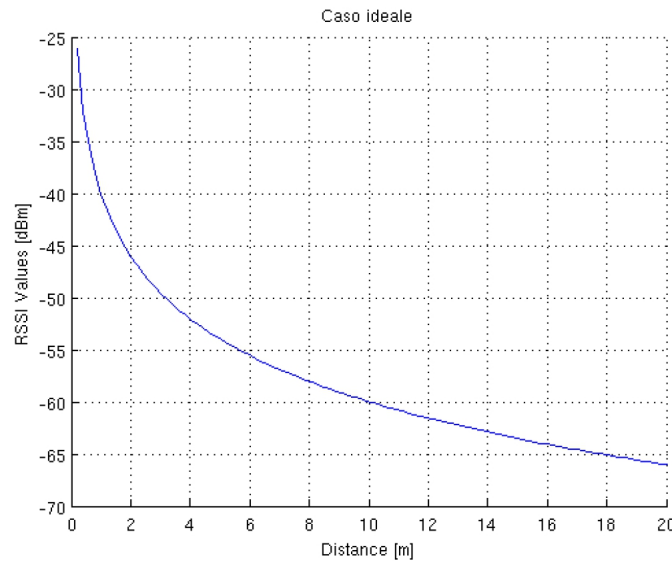


Figura 4.1: RSSI - Modello dell'andamento di RSSI in funzione della distanza

2. che la potenza media ricevuta attraverso un canale reale decade proporzionalmente a $n = d^{-n}$, dove **n** è detto **l'esponente path-loss**. Tipicamente n è compreso tra 2 e 4.

4.3.1 Equazione di trasmissione di Friis

La distanza dal trasmettitore viene valutata utilizzando l'**equazione di trasmissione di Friis**:

$$P_R = P_T \frac{G_T G_R \lambda^2}{(4\pi)^2 d^n} \quad (4.2)$$

dove:

- P_R : potenza del segnale ricevuto (espressa in Watt)
- P_T : potenza del segnale trasmesso (espressa in Watt)
- G_R : guadagno dell'antenna ricevente
- G_T : guadagno dell'antenna trasmittente
- $\lambda = \frac{v}{f}$: lunghezza d'onda
 - v : velocità di propagazione
 - f : frequenza dell'onda
- d : distanza in metri

- n : costante di propagazione del segnale che dipende dall'ambiente

L'equazione (4.2) calcola il rapporto tra la potenza ricevuta da un'antenna e la potenza trasmessa, in condizioni ideali.

4.3.2 Conversione della potenza

Conversione dalla potenza espressa in watt alla potenza espressa in dBm

$$1_{[dBm]} = 0.001258925_{[W]} \quad (4.3)$$

$$1_{[W]} = 30_{[dBm]} \quad (4.4)$$

$$P_{[dBm]} = 10 \log_{10}(10^3 P_{[W]} / 1_{[W]}) \quad (4.5)$$

4.3.3 Potenza media a distanza di riferimento d_0

$$P(d)_{[dBm]} = P_0_{[dBm]} \left(\frac{d}{d_0} \right)^{-n} \quad (4.6)$$

dove P_0 è la potenza ricevuta (dBm) a una piccola distanza di riferimento d_0 .

4.3.4 Equazione di RSSI

Combinando la (4.2) e la (4.5), applicando le proprietà dei logaritmi si ottiene:

$$RSSI = -(10 n \log_{10} d - A) \quad (4.7)$$

dove A è la potenza del segnale ricevuto (dBm) a distanza di un metro considerando una costante di propagazione n .

4.4 Calcolo della distanza

La seguente equazione permette di stimare la distanza tra l'utente ed un target conoscendo il valore RSSI ed i parametri A ed n :

$$RSSI = P - 10 * n * \log_{10}(d) * n = 2(infreespace) * d = 10^{(TxPower - RSSI) / (10 * n)} \quad (4.8)$$

$$d = 10^{\left(\frac{A - RSSI}{10 n} \right)} \quad (4.9)$$

Con questa formula si può stimare la distanza .

4.5 Problematiche della stima della distanza con RSSI

I principali fenomeni negativi che inficiano l'utilizzo di RSSI come approccio per determinare la distanza tra due punti sono:

- **Riflessione:** il segnale si propaga anche attraverso un percorso riflesso, provocando un **multi-path fading**. Al ricevitore giungono segnali con ampiezze e fasi differenti che vanno a sommarsi o sottrarsi in funzione della frequenza, causando un fading selettivo. Può essere causato da metalli e altri materiali riflettenti.
- **Intralcio:** shadowing che altera il normale decadimento dell'intensità che si avrebbe in spazio libero. L'attenuazione improvvisa del segnale è causata dagli ostacoli (mobili, muri, alberi, edifici, ecc.) nel cammino trasmettitore-ricevitore
- **Assorbimento:** oggetti, come elementi liquidi o corpi umani, che assorbono la potenza del segnale.
- **Altezza:** la differenza di altezza può falsare la stima
- **Orientamento relativo:** il segnale decade se il ricevitore non è direzionato verso l'emettitore

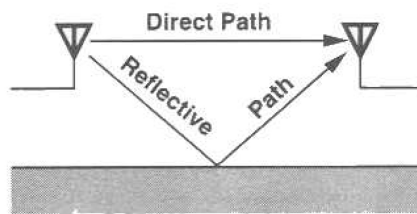


Figura 4.2: RSSI - Problemi di riflessione

Capitolo 5

Stima della distanza con Arduino

5.1 Progetto della stima della distanza con Arduino

- Implementare un progetto Arduino che faccia uso della libreria [NewPing](#)¹;
- Sfruttare la libreria [usb-serial-for-android](#)² per connettere l'Arduino ad Android e ricevere i dati seriali in tempo reale.
- Sviluppare un mini progetto su [NetBeans](#)³ ed impiegare la libreria [JS-SC](#)⁴ (*java-simple-serial-connector*) per testare il sensore di prossimità collegando l'Arduino al PC e visualizzando la distanza a schermo.

NewPing

Caratteristiche

- Compatibile con diversi modelli di sensori ad ultrasuoni: SR04, SRF05, SRF06, DYP-ME007 e Parallax PingTM.
- Non soffre di **lag** di un secondo se non si riceve un ping di eco.
- Produce un ping coerente e affidabile fino a 30 volte al secondo.
- Timer interrupt method per sketch event-driven.
- Metodo di filtro digitale built-in `ping_median()` per facilitare la correzione degli errori.

¹[NewPing](http://playground.arduino.cc/Code/NewPing) - <http://playground.arduino.cc/Code/NewPing>

²[usb-serial-for-android](https://github.com/mik3y/usb-serial-for-android) - <https://github.com/mik3y/usb-serial-for-android>

³[NetBeans](https://netbeans.org/) - <https://netbeans.org/>

⁴[JSSC](https://github.com/scream3r/java-simple-serial-connector) - <https://github.com/scream3r/java-simple-serial-connector>

- Utilizzo dei registri delle porte durante l'accesso ai pin per avere un'esecuzione più veloce e dimensioni del codice ridotte.
- Consente l'impostazione di una massima distanza di lettura del ping "in chiaro".
- Facilita l'utilizzo di più sensori.
- Calcolo distanza preciso, in centimetri, pollici e uS.
- Non fa uso di `pulseIn`, metodo che risulterebbe lento e che con alcuni modelli di sensore a ultrasuoni restituisce risultati errati.
- Attualmente in sviluppo, con caratteristiche che vengono aggiunte e bug/issues affrontati.

5.2 Progetto di test della comunicazione seriale PC - Arduino

Il metodo più importante di questo mini progetto è `updateDistance()`. Questo metodo è di tipo **void**. Il suo scopo è quello di ricevere in input i *byte* dalla porta seriale e convertirli in stringhe da visualizzare a schermo su una *JLabel*.

Per poter fare I/O sulla porta seriale si deve istanziare e configurare l'oggetto `SerialPort`, abilitando la comunicazione via USB con una board Arduino.

Per avviare la comunicazione:

- si settano i parametri di ingaggio (baud rate, numero di bit dei pacchetti, numero dei bit di stop e se è presente un controllo di parità)
- si imposta l'event mask in modo da controllare se sul canale sono presenti *char*
- si registra l'istanza di `SerialPort` in un listener di eventi di I/O della seriale per poi considerare solo i dati di tipo *char* e scartare tutti gli altri.

Metodo `updateDistance()`

```
private void updateDistance() {
    SerialPort serialPort = new SerialPort("/dev/ttyACM0");
    try {
        serialPort.openPort();

        serialPort.setParams(
            SerialPort.BAUDRATE_115200,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
```



```

        SerialPort.PARITY_NONE);

        serialPort.setEventsMask(SerialPort.MASK_RXCHAR);

        serialPort.addEventListener((SerialPortEvent
        serialPortEvent) -> {
            if (serialPortEvent.isRXCHAR()) {
                try {
                    Thread.sleep(20);
                    String distance =
                        serialPort.readString();
                    jLabel1.setText(distance);
                } catch (SerialPortException |
                    InterruptedException ex) {
                }
            }
        });
    } catch (SerialPortException ex) {
        System.out.println("SerialPortException: " +
            ex.toString());
    }
}

```

- Interagire con gli iBeacon sfruttando la libreria [AltBeacon](#)⁵.
- Implementare un **filtro di Kalman** per ridurre gli effetti indesiderati del *multipath fading* sulla stima della distanza.
- Implementare un **filtro ARMA** (*Auto Regressive Moving Average*) per stimare al meglio il valore RSSI.
- Implementare un **filtro RunningAverageRssi** da confrontare con i precedenti.
- Visualizzare grafici in tempo reale utilizzando la libreria [MPAndroidChart](#)⁶

⁵[AltBeacon](https://github.com/AltBeacon/android-beacon-library) - <https://github.com/AltBeacon/android-beacon-library>

⁶[MPAndroidChart](https://github.com/PhilJay/MPAndroidChart) - <https://github.com/PhilJay/MPAndroidChart>

Capitolo 6

UML

6.1 Gerarchia delle classi

6.1.1 ApplicationActivity

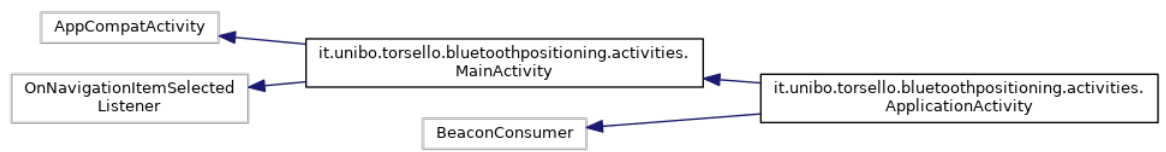


Figura 6.1

6.1.2 Fragment e Observer

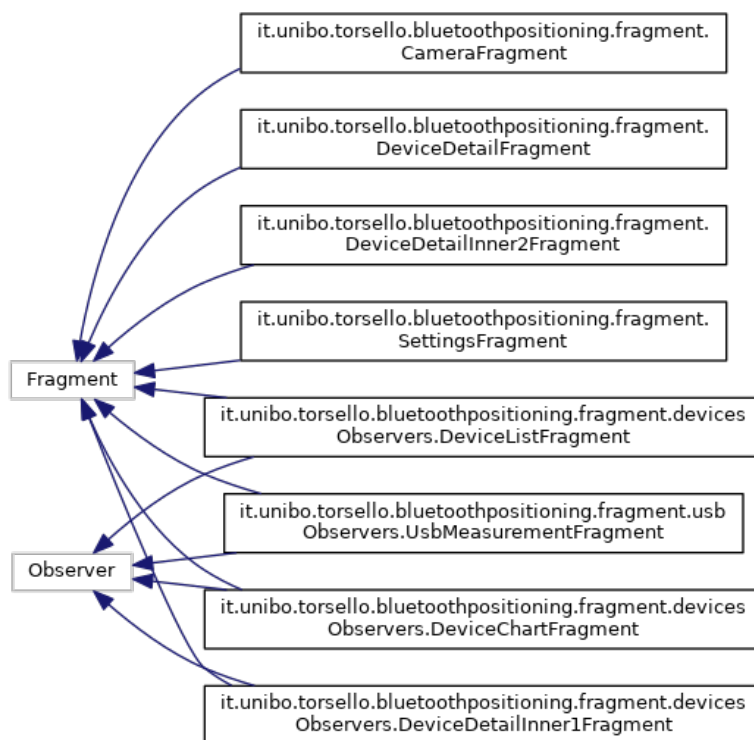


Figura 6.2

6.1.3 Observable

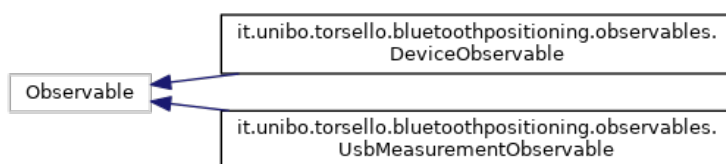
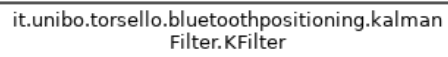


Figura 6.3

6.1.4 KFilter

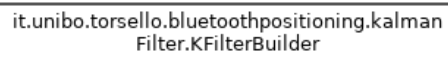


```
classDiagram
    class KFilter {
        it.unibo.torsello.bluetoothpositioning.kalman
    }
    class Filter {
        KFilter
    }
```

it.unibo.torsello.bluetoothpositioning.kalman
Filter.KFilter

Figura 6.4

6.1.5 KFilterBuilder



```
classDiagram
    class KFilterBuilder {
        it.unibo.torsello.bluetoothpositioning.kalman
    }
    class Filter {
        KFilterBuilder
    }
```

it.unibo.torsello.bluetoothpositioning.kalman
Filter.KFilterBuilder

Figura 6.5

6.1.6 MyArmaRssiFilter



Figura 6.6

6.1.7 Estimation

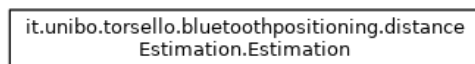


Figura 6.7

6.1.8 Device



Figura 6.8

6.1.9 DeviceCardViewAdapter



Figura 6.9

6.1.10 DeviceCardViewAdapter.DeviceViewHolder



Figura 6.10

6.1.11 StatePagerAdapter



Figura 6.11

6.1.12 DeviceConstants

it.unibo.torsello.bluetoothpositioning.constant. DeviceConstants

Figura 6.12

6.1.13 KFilterConstants

it.unibo.torsello.bluetoothpositioning.constant. KFilterConstants
--

Figura 6.13

6.1.14 SettingConstants

it.unibo.torsello.bluetoothpositioning.constant. SettingConstants
--

Figura 6.14

6.1.15 CameraPreviewUtil



Figura 6.15

6.1.16 SaveImageTask



Figura 6.16

6.1.17 ChartUtil



Figura 6.17

6.1.18 UsbUtil



Figura 6.18

6.1.19 FABBehavior



Figura 6.19