

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

Federico Torsello - Matr. 702619

14 ottobre 2016

Indice

1	Introduzione	1
1.1	Vision	1
1.2	Goals	2
1.2.1	Goals principali	2
1.2.2	Goals secondari	2
I	Requisiti	3
2	Definizione dei requisiti	4
2.1	Requisiti funzionali	4
2.2	Requisiti funzionali secondari	4
2.3	Requisiti non funzionali	5
2.4	Analisi dei requisiti	5
2.4.1	Casi d'uso	5
2.4.2	Scenari	5
II	Analisi del problema	7
3	Breve introduzione alla tecnologia iBeacon	8
3.1	Il protocollo iBeacon	9
3.1.1	Broadcaster	9
3.1.2	Receiver	9
3.1.3	Advertising packet	9
3.2	Frame iBeacon	9
3.2.1	UUID (128 bits)	10
3.2.2	Major number (16 bits)	10
3.2.3	Minor number (16 bits)	10
3.3	RSSI - Received Signal Strength Indication	11
3.3.1	Advertising Interval	11
3.4	iBeacon Advertising Packet Contents	11
3.4.1	Measured power	12

4 Stima della distanza con RSSI	13
4.1 Attenuazione dei segnali elettromagnetici	13
4.2 Received Signal Strength Indicator - RSSI	13
4.3 Calcolo di RSSI	13
4.3.1 Equazione di trasmissione di Friis	14
4.3.2 Conversione della potenza	15
4.3.3 Potenza media a distanza di riferimento d_0	15
4.3.4 Equazione di RSSI	15
4.4 Calcolo della distanza	15
4.5 Problematiche della stima della distanza con RSSI	16
4.6 Filtro di Kalman	16
5 Stima della distanza con RSSI, Android e tecnologie BLE	18
5.1 Sviluppare un'app Android compatibile con BLE	18
5.2 Compatibilità con Android 5.0+	18
5.3 Libreria AltBeacon - android-beacon-library	20
5.4 Calcolo della distanza RAW	20
5.5 Filtri	20
5.5.1 Filtro RunningAverageRssi	21
5.5.2 Implementazione del filtro di Kalman	21
5.5.3 Implementazione del filtro ARMA	23
5.5.4 Grafici in tempo reale	25
6 Stima della distanza con Arduino	26
6.1 Progetto per stimare la distanza con un sensore ultrasonico ed un Arduino	26
6.2 Progetto di test della comunicazione seriale PC - Arduino . .	27
III Progetto	29
7 Gerarchia delle classi	30
7.1 MainActivity	30
7.2 ApplicationActivity	32
7.3 Device	34
7.4 DeviceConstants	35
7.5 DeviceObservable	36
7.6 Estimation	37
7.7 SettingsFragment	38
7.8 SettingConstants	40
7.9 DeviceListFragment	41
7.10 DeviceCardViewAdapter	44
7.11 DeviceViewHolder	46
7.12 MyArmaRssiFilter	48

7.13	KFilterBuildertFragment	49
7.14	KFilter	50
7.15	KFilterConstants	51
7.16	StatePagerAdapter	52
7.17	DeviceDetailFragment	53
7.18	CameraFragment	54
7.19	CameraPreviewUtil	56
7.20	SaveImageTask	58
7.21	DeviceDetailInner1Fragment	59
7.22	DeviceDetailInner2Fragment	60
7.23	DeviceChartFragment	61
7.24	ChartUtil	62
7.25	UsbMeasurementFragment	63
7.26	UsbUtil	65
7.27	UsbMeasurementObservable	68
7.28	FABBehavior	69
IV	Implementazione	70
8	GUI dell'app	71
8.1	Avvio dell'app	71
8.2	Abilitazione della radio Bluetooth	72
8.3	Scansione dei dispositivi	73
8.4	Menù a sinistra	74
8.5	Menù a destra	75
8.6	Foto ad un dispositivo	76
8.7	Dettagli e grafici real time	77
8.8	Connessione con Arduino via USB OTG	78
8.9	Feedback della stima della distanza con Arduino	79
9	Testing	80
9.1	Configurazione dell'ambiente indoor	80
9.2	Collegamento Arduino-smartphone	81
9.2.1	Materiale utilizzato	82
9.2.2	Dati raccolti	83

Sommario

Capitolo 1 In questo capitolo si viene introdotti al progetto, quindi vengono descritti la vision e i goals.

Capitolo 2 In questo capitolo si definiscono ed analizzano i requisiti funzionali e non funzionali, i casi d'uso e i scenari relativi al progetto.

Capitolo 3 In questo capitolo si analizza brevemente la tecnologia iBeacon ed in generale il BLE (Bluetooth Low Energy). Sono inoltre forniti alcuni esempi di utilizzo reali e caratteristiche del protocollo di comunicazione.

Capitolo 4 In questo capitolo si definisce cos'è RSSI, come calcolarlo e come sfruttarlo per stimare la distanza utente-iBeacon target.

Capitolo 5 In questo capitolo si affronta la stima della distanza con RSSI, Android e tecnologie Bluetooth Low Energy. In particolare si introduce all'utilizzo della libreria AltBeacon e di filtri su RSSI e sulle distanze stimate.

Capitolo 6 In questo capitolo si parla della stima della distanza utilizzando Arduino, prima con un progetto di test con questo connesso al PC e poi con l'implementazione legata all'app. Nel dettaglio si parlerà di come è possibile connettere direttamente l'Arduino allo smartphone per ricevere dati sensoristici.

Capitolo 7 In questo capitolo si struttura il progetto facendo l'analisi delle classi più alcuni codici sorgente di esempio.

Capitolo 8 In questo capitolo si passa all'implementazione vera e propria dell'app, riportando immagini e descrizioni per del suo utilizzo.

Capitolo 9 In questo capitolo si definisce il testing reale della stima delle distanze e gli strumenti necessari per eseguirlo.

Capitolo 1

Introduzione

Stimare con precisione la distanza che intercorre tra un utente ed un punto target utilizzando poco hardware ad un costo accessibile è tutt'ora una sfida tecnologica a cui si cerca una soluzione valida. Questa informazione è diventata via via sempre più importante in quanto vari sistemi (mobile e non) ne necessitano 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.

D'altro canto, vista la disponibilità pressoché capillare degli smartphone, sembra ovvio cercare di utilizzarli come risorsa per trovare una soluzione al problema citato. Tali dispositivi mobili implementano sensoristica, capacità di calcolo e software (inteso come OS o come librerie) sfruttabile direttamente dall'utente mediante un'app creata ad-hoc.

In questo senso i dispositivi iBeacon (grazie alla tecnologia BLE) possono essere utilizzati in relazione agli smartphone, aiutando a stimare la distanza utente-iBeacon con un costo limitato ed una buona efficienza.

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, ribadendo 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" (o meglio, di riferimento) stimata.
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 distanza di riferimento iBeacon-utente per poterla confrontare con quella stimata utilizzando gli RSSI.

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

Parte I

Requisiti

Capitolo 2

Definizione dei requisiti

2.1 Requisiti funzionali

Per la realizzazione 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 o Google Eddystone;
- un ambiente chiuso (come una stanza) in cui disporre gli iBeacon.

2.2 Requisiti funzionali secondari

Ai fini di un confronto tra la distanza stimata grazie agli iBeacon e quella "reale" si vuole utilizzare una board Arduino ed un sensore ultrasonico; quindi creare un feedback a schermo. Nello specifico 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.

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

2.3 Requisiti non funzionali

I requisiti funzionali di questo progetto sono molteplici:

- ottenere una GUI fluida e di facile utilizzo;
- realizzare dei filtri che non incidano sulle prestazioni del sistema;
- avere dei feedback reali di quello che succede, con numeri ben visibili e grafici in realtime con colori distinguibili;
- poter registrare i dati (sotto forma di immagine o testo) per analisi a posteriori;
- ottenere un'app che non vada in crash.

2.4 Analisi dei requisiti

2.4.1 Casi d'uso

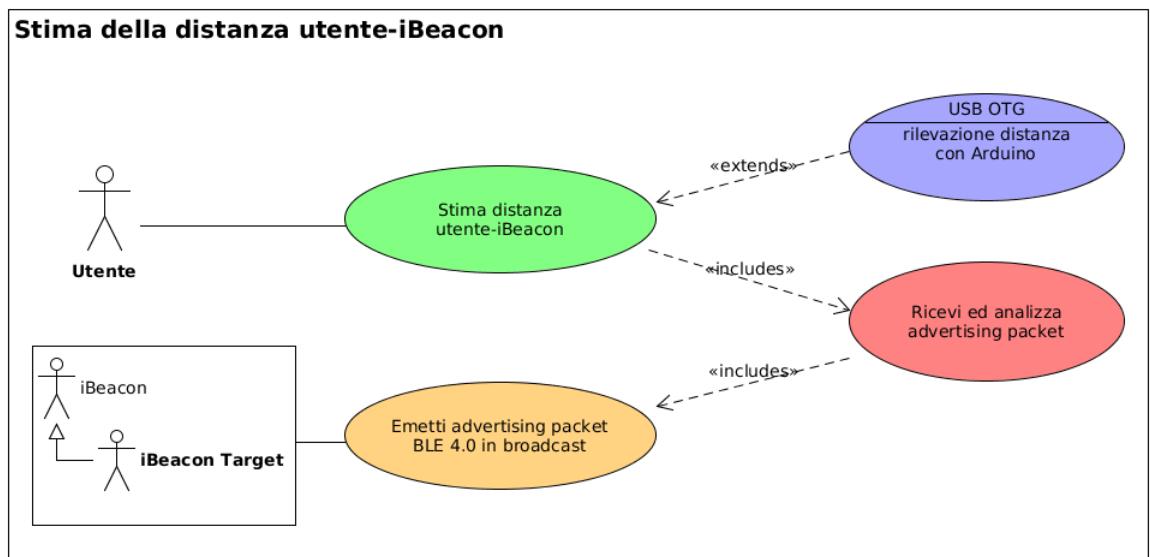


Figura 2.1: Use case - Stima della distanza utente-iBeacon

2.4.2 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.
- Sullo smartphone viene avviata l'app.
- Nel caso in cui la radio Bluetooth dello smarphone 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.

Parte II

Analisi del problema

Capitolo 3

Breve introduzione alla tecnologia iBeacon

Gli iBeacon (anche detti BLE Beacon) sono dispositivi portatili, spesso alimentati a batteria, che integrano una radio Bluetooth. Il loro scopo è definire **regioni** di spazio delimitato dall'estensione del segnale che emettono in broadcast ripetutamente con una frequenza predefinita. Si rendono utili per *micro-location system* in quanto offrendo una precisione molto senza forti ripercussioni sulla loro autonomia.

Spesso sono utilizzati come *trigger* per App che integrano *custom action*, reagendo alla prossimità, creando delle reazioni visibili sullo smartphone che utente appaiono come **interazioni col mondo fisico**.

La tecnologia chiave che permette agli iBeacon di funzionare è la **proximity specification BLE** (Bluetooth Low Energy), spesso indicata come **Bluetooth Smart**. BLE è un miglioramento della specifica Bluetooth che permette un consumo energetico ridotto che si ripercuote positivamente sull'autonomia dei dispositivi a batteria.

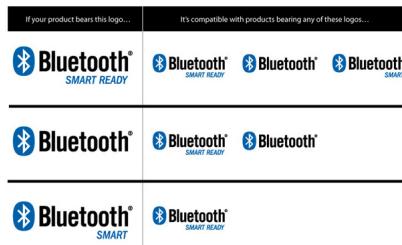


Figura 3.1: Compatibilità delle diverse tecnologie Bluetooth

Il limitato consumo energetico degli iBeacon si deve alla ridotta potenza di trasmissione di questa tecnologia che permette lunghi periodi di utilizzo (mesi o addirittura anni).

3.1 Il protocollo iBeacon

Il protocollo di comunicazione iBeacon è semplice e lineare. Si suddivide tra **broadcaster** (trasmettitore) e **receiver** (ricevitore).

3.1.1 Broadcaster

Nella specifica Bluetooth, un iBeacon è per definizione un *broadcaster*. I broadcaster trasmettono **advertising packet** periodici che contengono informazioni utilizzabili dai *receiver*.

3.1.2 Receiver

In questo protocollo i receiver non devono rispondere ai pacchetti che ricevono ed in generale i broadcaster non sono abilitati alla ricezione. La comunicazione degli advertising packet avviene sempre e solo attraverso una trasmissione unidirezionale.

3.1.3 Advertising packet

Essenzialmente gli advertising packet sono dispersi dai broadcaster nell'aria e i receiver possono scegliere se agire o no in base al loro contenuto.

La ricezione di un advertising packet da parte di un receiver provoca la creazione di un **advertisement event**. Per questo gli advertising packet e gli advertisement event genericamente sono denominati come **advertisement**.



Figura 3.2: Logo che identifica i dispositivi iBeacon

3.2 Frame iBeacon

Il formato del frame iBeacon è abbastanza semplice. Sono presenti solo un due parametri variabili che sono comunque sufficienti per supportare applicazioni complesse.

Le informazioni che un iBeacon emette si suddividono in tre identificativi:

- Universal Unique Identifier
- Major number
- Minor number

3.2.1 UUID (128 bits)

L'UUID identifica univocamente la società di cui l'iBeacon fa parte. A differenza di altri protocolli di rete come 802.11, l'UUID non è gestito centralmente per evitare conflitti. Il protocollo Bluetooth presuppone che UUID siano unici.

L'UUID è il numero più facile da elaborare perché dovrebbe essere unico. Un sistema che supporta più brand name può utilizzare diversi UUID.

La maggior parte degli UUID sono creati da generatori di numeri casuali, spesso integrando l'ora corrente e un identificatore del generatore (ad esempio l'indirizzo MAC). Molte applicazioni per la configurazione di iBeacon commerciali hanno un pulsante per generare un UUID pseudocasuale.

3.2.2 Major number (16 bits)

Le specifiche Bluetooth e iBeacon non dispongono di una struttura per l'uso dei major o minor numbers.

Il Major number viene utilizzato per identificare i Major groups di iBeacon di proprietà di un'unica entità. Nell'esempio della catena di negozi, il Major number tipicamente è utilizzato dai iBeacon che si trovano all'interno del negozio, identificando gruppi di proximity area in modo logico. Il campo a 16 bit permette di avere 65.000 possibilità.

3.2.3 Minor number (16 bits)

Il Minor number viene utilizzato per identificare il livello più basso della gerarchia all'interno di un insieme di iBeacon. Tornando all'esempio di una catena di negozi, il Minor number sarà utilizzato per i singoli iBeacon all'interno di una singola posizione del negozio, per esempio potrebbe identificare un prodotto esposto.

Il Minor number è una ulteriore suddivisione del raggruppamento definito dal Major number. I Minor number di solito si riferiscono ad una posizione geografica o POI all'interno di una posizione.

3.3 RSSI - Received Signal Strength Indication

La **proximity estimation** (stima di prossimità) dipende dal RSSI. RSSI non è trasmesso nel advertising packet, ma viene percepito dal receiver come il livello di potenza del segnale ricevuto.

3.3.1 Advertising Interval

Anche se le specifiche Bluetooth permettono di definire più *advertising interval*, la specifica iBeacon fissa l'advertising interval a 100 ms.

L'impostazione dell'advertising interval è bilanciato in modo da preservare la vita della batteria, ma comunque essere abbastanza lungo per consentire ai servizi basati sugli iBeacon di funzionare.

3.4 iBeacon Advertising Packet Contents

Tutti i advertising packet hanno sempre la stessa lunghezza e sono composti da una serie di campi fissi. L'ultima parte del frame contiene informazioni specifiche del costruttore definito da Apple. È comunque possibile definire un formato pacchetto personalizzato in modo da migliorare le capacità di puntamento un iBeacon.

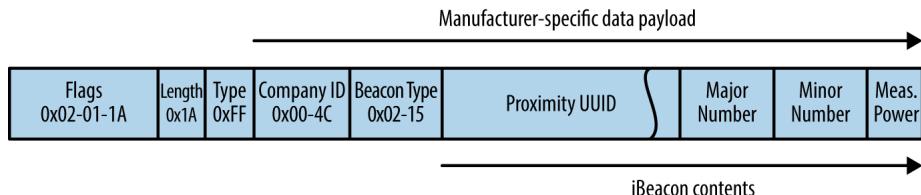


Figura 3.3: Formato di un iBeacon advertising packet format

3.4.1 Measured power

L'idea di **ranging** è implicita all'interno del protocollo iBeacon. Il *ranging* indica la distanza tra il receiver e l'iBeacon analizzando le variazioni di potenza della trasmissione ricevuta. La distanza da un iBeacon viene stimata in base alla costante di calibrazione adeguata, misurando il tempo che un segnale impiega per poter essere ricevuto e la sua potenza.

Il *Measured power* è un parametro impostato tenendo un receiver ad un metro dall'iBeacon e calcolando la media di RSSI. Questo campo contiene la potenza misurata come complemento a due.

Esempio: un valore di C5 indica che la potenza misurata a un metro è di -59 dBm.

Dati:

- $0xC5_{hex} = 197_{dec}$
- $0x100_{hex} = 256_{dec}$

$$(256 - 197) = -59 \text{ dBm}$$

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**.

$$P = f(d) \quad (4.1)$$

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

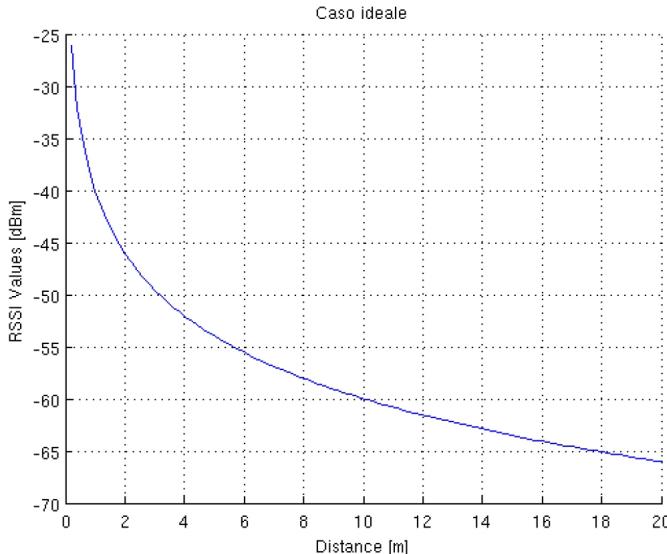


Figura 4.1: RSSI - Modello dell’andamento di RSSI in funzione della distanza

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
- $\lambda = \frac{c}{f}$: lunghezza d’onda
 c : 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 :

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

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

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 degli 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'emittitore

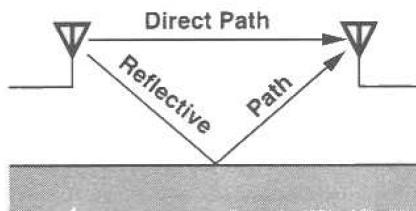


Figura 4.2: RSSI - Problemi di riflessione

4.6 Filtro di Kalman

Definizione 1: Il filtro di Kalman è stato sviluppato da Rudolf Kalman. La prima realizzazione pratica di questo filtro è stata sviluppata da Stanley Schmidt, che ne riconobbe l'applicabilità per stimare le traiettorie ed eliminare alcuni disturbi.

Teoricamente è uno strumento per stimare lo stato di un sistema dinamico lineare perturbato da rumore sulla base di misure (o osservazioni) che sono linearmente dipendenti dallo stato e corrotte da rumore.

Questo stimatore si definisce **ottimo** rispetto a qualunque funzione quadratica dell'errore di stima in quanto si basa su tutte le informazioni disponibili.

Definizione 2: Il filtro di Kalman è una tecnica per la risoluzione del problema Gaussiano lineare quadratico traducibile, indicato come stima dello stato istantaneo di un sistema dinamico a partire dalla sua uscita. Si considera statisticamente ottimale rispetto a qualunque funzione quadratica di stima dell'errore.

Supponendo di conoscere le variabili misurate in funzione delle variabili di interesse, questo filtro si occupa della soluzione del problema inverso, cioè stimando le variabili indipendenti come funzione inversa delle variabili dipendenti (misurabili).

Dato il sistema dinamico:

$$\begin{cases} dX = f(x(t), u(t), t) \\ y = h(x(t), u(t), t) \end{cases} \quad (4.10)$$

Il filtro si occupa di risalire allo stato $x(t)$ a partire dall'uscita $y(t)$ quando queste sono perturbate da rumore gaussiano.

La costruzione del filtraggio dei dati avviene sulla base di una media pesata tra il prossimo valore predetto e il successivo valore stimato.

L'algoritmo è composto da alcune equazioni matematiche che effettuano operazioni ricorsive per dare un efficace soluzione al metodo dei minimi quadrati.

Per la sua costruzione il filtro di Kalman necessita di **tre ingredienti fondamentali** vincolati al sistema che si sta studiando:

1. una serie di misure da stimare
2. un modello matematico descrittivo del sistema
3. la conoscenza del modello statistico del sistema

Capitolo 5

Stima della distanza con RSSI, Android e tecnologie BLE

5.1 Sviluppare un'app Android compatibile con BLE

L'obiettivo dell'app è far interagire un cellulare (con sistema operativo Android) con gli iBeacon disposti in una stanza. Dalle API 18 in poi Android integra nativamente la comunicazione e ricezione dati su protocollo Bluetooth LE.

Visto che l'applicazione non funziona senza questa specifica, nell'`AndroidManifest.xml` sono state imposte delle condizioni che inibiscono l'installazione dell'app a dispositivi che non hanno una radio Bluetooth Low Energy.

```
<uses-permission  
    android:name="android.permission.BLUETOOTH" />  
<uses-permission  
    android:name="android.permission.BLUETOOTH_ADMIN" />  
<uses-feature android:name="android.hardware.bluetooth_le"  
    android:required="true" />
```

Al fine di inibire l'installazione a smartphone con API minori a 18, nel file `build.gradle` è stata inserita la regola:

```
defaultConfig {  
    ...  
    minSdkVersion 18  
    ...  
}
```

5.2 Compatibilità con Android 5.0+

Per rendere eseguibile l'app a smarphone con Android dalla versione 5.0 in poi è stato aggiunto un metodo (`checkAndroidMPermission()`) per fare il

check dei permessi di Android. Senza questo accorgimento l'app sarebbe stata installata ma non avrebbe funzionato la scansione dei dispositivi BT.

Controllo della compatibilità con Android 5.0+

```
private void checkAndroidMPermission() {  
  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {  
        final List<String> permissions = new ArrayList<>();  
  
        if (checkSelfPermission(Manifest.permission  
            .ACCESS_FINE_LOCATION)  
            != PackageManager.PERMISSION_GRANTED) {  
            permissions.add(Manifest.permission  
                .ACCESS_FINE_LOCATION);  
        }  
  
        if (checkSelfPermission(Manifest.permission  
            .ACCESS_COARSE_LOCATION)  
            != PackageManager.PERMISSION_GRANTED) {  
            permissions.add(Manifest.permission  
                .ACCESS_COARSE_LOCATION);  
        }  
  
        if (!permissions.isEmpty()) {  
            new AlertDialog.Builder(this)  
                .setTitle(R.string.dialog_location_access_title)  
                .setMessage(R.string.dialog_bluetooth_text)  
                .setPositiveButton(android.R.string.ok, null)  
                .setOnDismissListener(new  
                    DialogInterface.OnDismissListener() {  
                        @TargetApi(23)  
                        @Override  
                        public void onDismiss(DialogInterface  
                            dialog) {  
                            requestPermissions(permissions.toArray(  
                                new String[permissions.size()] ),  
                                REQUEST_CODE_ASK_MULTIPLE_PERMISSIONS);  
                        }  
  
                }).show();  
        }  
    }  
}
```

5.3 Libreria AltBeacon - android-beacon-library

Per estendere il supporto nativo fornito alla connettività Bluetooth è stata utilizzata la libreria **AltBeacon**¹. Questa permette di generare delle regioni di interesse, degli avvisi all'utente, filtrare i dispositivi BT, ecc. Con essa è inoltre fornito un modulo per il risparmio energetico.

Per l'inclusione di questa libreria si inserita la seguente stringa nel file `build.gradle`.

```
dependencies {  
    ...  
    // android beacon library AltBeacon  
    compile 'org.altbeacon:android-beacon-library:2.9.1'  
    ...  
}
```

5.4 Calcolo della distanza RAW

La stima della distanza in base ai RSSI calcolati si esegue nel metodo `calculateDistance(double txPower, double rssi)`.

Stima della distanza in base agli RSSI

```
// radiosNetwork formula  
private double calculateDistance(double txPower, double  
rssi) {  
  
    if (rssi == 0.0D) {  
        return -1.0D; // if we cannot determine accuracy,  
        return -1.  
    }  
  
    double ratio = (rssi * 1.0D) / txPower;  
    if (ratio < 1.0D) {  
        return Math.pow(ratio, 10.0D);  
    }  
  
    return (0.89976D * Math.pow(ratio, 7.7095D)) + 0.111D;  
}
```

5.5 Filtri

Per ridurre le problematiche di rilevamento di RSSI(4.5) sono stati sviluppati e/o utilizzati tre filtri: RunningAverageRssi, filtro di Kalman, filtro ARMA.

¹ **AltBeacon** - <https://github.com/AltBeacon/android-beacon-library>

Questi possono essere combinati tra loro senza inficiare le prestazioni del sistema. Presentano delle parametrizzazioni controllabili dal *Settings* per adattarsi a tutti vari contesti di stima.

Per scelta progettuale il filtro di Kalman è sempre attivo. Si è fatto in modo che il valore filtrato col questo filtro sia confrontabile con la misurazione della distanza RAW e con quella filtrata dalla libreria AltBeacon.

5.5.1 Filtro RunningAverageRssi

Il filtro *RunningAverageRssi* calcola il valore RSSI sulla base di un elenco arbitrario di valori RSSI misurati. L'elenco viene troncato da una certa lunghezza all'inizio e alla fine. Il valore calcolato è una semplice media aritmetica. Viene fornito dalla libreria AltBeacon come filtro di default, ma non sembra essere particolarmente preciso.

5.5.2 Implementazione del filtro di Kalman

Per realizzare il filtro di Kalman sono state utilizzate tre classi:

- **KFilterBuilder**: per creare un filtro di Kalman unidimensionale;
- **KFilter**: per il filtraggio vero e proprio.
- **Estimation**: per implementare il filtro sui valori di input.

Parametri passati a KFilter:

- R = rumore di processo;
- Q = misurazione del rumore;
- A = vettore di stato;
- B = vettore di controllo;
- C = vettore delle misurazioni.

Questo filtro è inizializzato nella classe **Estimation** (istanziata volta alla creazione di degli oggetti *Device*). Ogni volta che viene richiamato il metodo `updateDistance(Beacon b, double processNoise)` di **ESTimation**;

- si aggiunge un nuovo valore alla variabile `recentRSSI`, istanza di un oggetto **DescriptiveStatistics**
- si aggiunge un nuovo valore alla variabile `recentTxPower`, istanza di un oggetto **DescriptiveStatistics**
- si calcola il nuovo valore di rumore in base ai nuovi input

- se il nuovo valore di rumore non è un numero infinito e non è un Not-a-Number, si passa al metodo `setMeasurementNoise(mNoise)` di `KFilter`
- si setta il nuovo rumore di processo, che dipende dall'input dato da `Settings`
- si calcola la distanza con i parametri appena calcolati

Filtraggio in base agli input

```
public void updateDistance(Beacon b, double processNoise) {

    recentRSSI.addValue(b.getRssi());
    recentTxPower.addValue(b.getTxPower());

    // Update measurement noise continually
    double mNoise = Math.sqrt((100 * 9 / Math.log(10)) *
        Math.log(1 + Math.pow(recentRSSI.getMean() /
            recentRSSI.getStandardDeviation(), 2)));

    if (!Double.isInfinite(mNoise) &&
        !Double.isNaN(mNoise)) {
        kf.setMeasurementNoise(mNoise);
    }

    kf.setProcessNoise(processNoise);
    double lastFilteredReading =
        kf.filter(recentRSSI.getPercentile(50));
    distanceEstimated =
        calculateDistance(recentTxPower.getPercentile(50),
            lastFilteredReading);
    rawDistanceEstimated =
        calculateDistance(b.getTxPower(), b.getRssi());
    WOSC = calculateDistance(b.getTxPower(),
        lastFilteredReading);
}
```

Filtering nella classe KFilter

```
/**
 * Filter a new value
 *
 * @param z Measurement
 * @param u Control
 * @return x
 */
```

```

public double filter(double z, double u) {

    if (Double.isNaN(x)) {
        x = (1 / C) * z;
        x1 = x;
        x2 = x1;
        cov = (1 / C) * Q * (1 / C);
    } else {

        // Calculate previous update step
        B = (x - x1) / 2;

        // Compute prediction
        double predX = (A * x) + (B * u);
        double predCov = ((A * cov) * A) + R;

        // Kalman gain
        double K = predCov * C * (1 / ((C * predCov * C) +
            Q));

        // Correction
        x1 = x;
        x = predX + K * (z - (C * predX));
        cov = predCov - (K * C * predCov);
    }

    return x;
}

```

5.5.3 Implementazione del filtro ARMA

Il filtro **ARMA** (*Auto Regressive Moving Average*) calcola gli RSSI in base al valore corrente. Questo filtro è tradotto in codice nella classe `MyArmaRssiFilter` che implementa la classe `RssiFilter`.

Per il filtraggio degli RSSI si utilizza la formula 5.1:

$$n(t) = n(t-1) - c(n(t-1) - n(t)) \quad (5.1)$$

dove c è il coefficiente che denota l'uniformità (*smoothness*). Più basso è questo valore più uniforme è la media.

Il metodo più importante di questa classe è `addMeasurement(Integer rssi)` dove si ricevono in input gli RSSI e si esegue il filtraggio in base al coefficiente c (qui tradotto nella variabile `armaSpeed`).

Aggiornamento del filtro ARMA

```

@Override
public void addMeasurement(Integer rssi) {

    if (isEnabled) {
        if (!isInitialized) {
            armaMeasurement = rssi;
            isInitialized = true;
        }
        armaMeasurement = (armaMeasurement - armaSpeed *
                           (armaMeasurement - rssi));
    } else {
        armaMeasurement = rssi;
    }
}

```

Per il settaggio dinamico del coefficiente c si utilizza il metodo `setArmaSpeed(double arma_speed)`. Visto che i segnali considerati tendono a variare piuttosto frequentemente (alla frequenza di 1Hz o superiore), il valore consigliato per 1HZ sarebbe 0,1 (cioè l'input viene modificato del 10% della differenza tra la misurazione attuale e la media effettiva). Per segnali a frequenza superiori (10Hz) si consiglia un valore compreso tra 0,25 e 0,5.

5.5.4 Grafici in tempo reale

Si è scelto di far visualizzare a schermo, direttamente dall'app, dei grafici in tempo reale in modo da rendere più chiaro l'andamento delle stime. Per far questo è stata utilizzata la libreria [MPAndroidChart²](#).

Per l'inclusione di questa libreria si inserita la seguente stringa nel file `build.gradle`.

```
dependencies {  
    ...  
    // chart library  
    compile 'com.github.PhilJay:MPAndroidChart:v3.0.0-beta1'  
    ...  
}
```

Dettaglio di un grafico in tempo reale



Figura 5.1: Esempio di grafico in tempo reale

²[MPAndroidChart](https://github.com/PhilJay/MPAndroidChart) - <https://github.com/PhilJay/MPAndroidChart>

Capitolo 6

Stima della distanza con Arduino

6.1 Progetto per stimare la distanza con un sensore ultrasonico ed un Arduino

Per poter stimare la distanza con Arduino è stato utilizzato un sensore ultrasonico. Questo sensore è stato connesso direttamente alla board attraverso cavi maschio-femmina.

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <action android:name="android.hardware.usb.action
        .USB_DEVICE_ATTACHED"/>

    <category
        android:name="android.intent.category.DEFAULT" />
    <category
        android:name="android.intent.category.LAUNCHER" />

</intent-filter>

<meta-data android:name="android.hardware.usb.action
    .USB_DEVICE_ATTACHED"
    android:resource="@xml/device_filter" />
```

- 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.

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>

- 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.
- Utilizzo dei registri delle porte durante l'accesso ai pin per avere un'escuzione 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.

6.2 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:

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>

- si settano i parametri di ingaggio (baund 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 prenti *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());
    }
}

```

Parte III

Progetto

Capitolo 7

Gerarchia delle classi

7.1 MainActivity

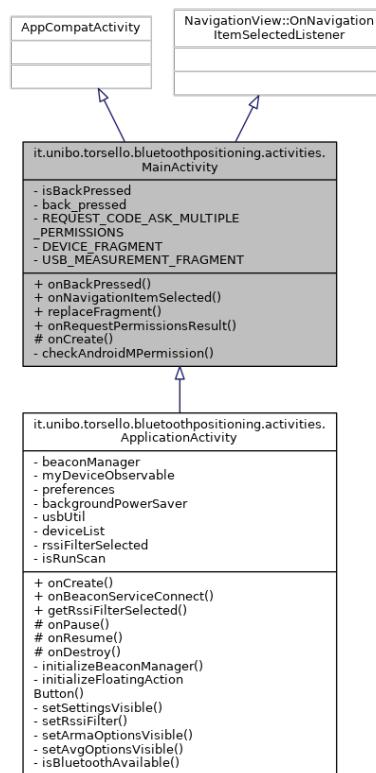


Figura 7.1: Classe - MainActivity

La classe `Main` viene istanziata per prima all'avvio dell'app. Serve ad inizializzare gli oggetti grafici, controllare la compatibilità con Android 5.0+ e rimpiazzare i fragment da visualizzare selezionati dal menu a sinistra.

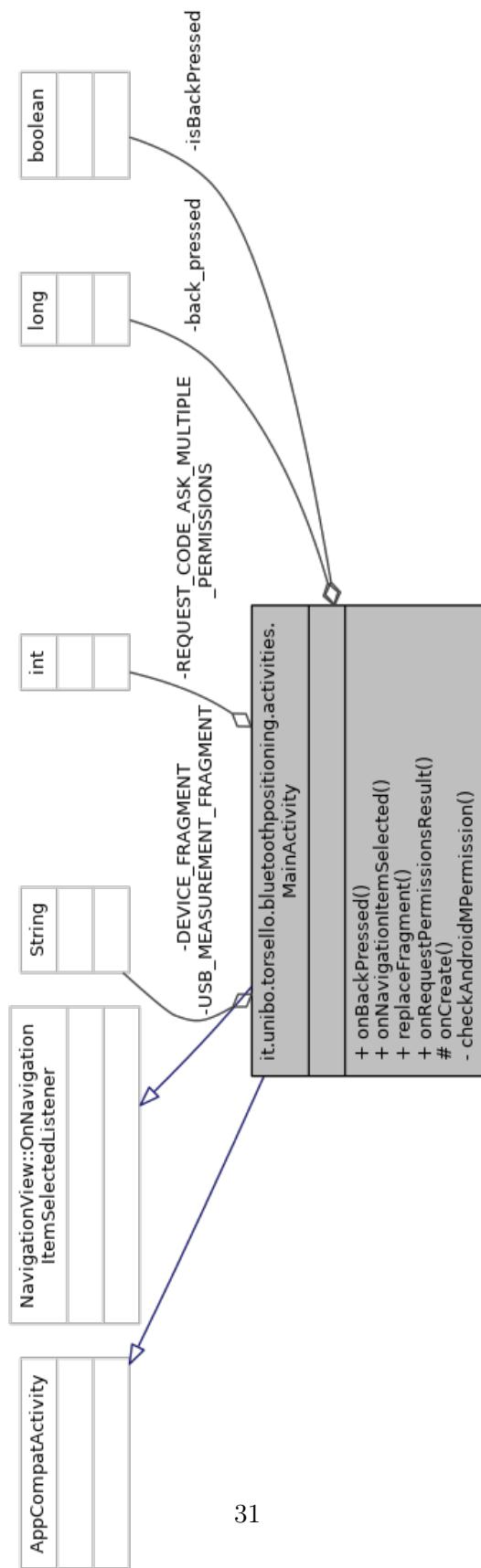


Figura 7.2: Collaborazione - MainActivity

7.2 ApplicationActivity

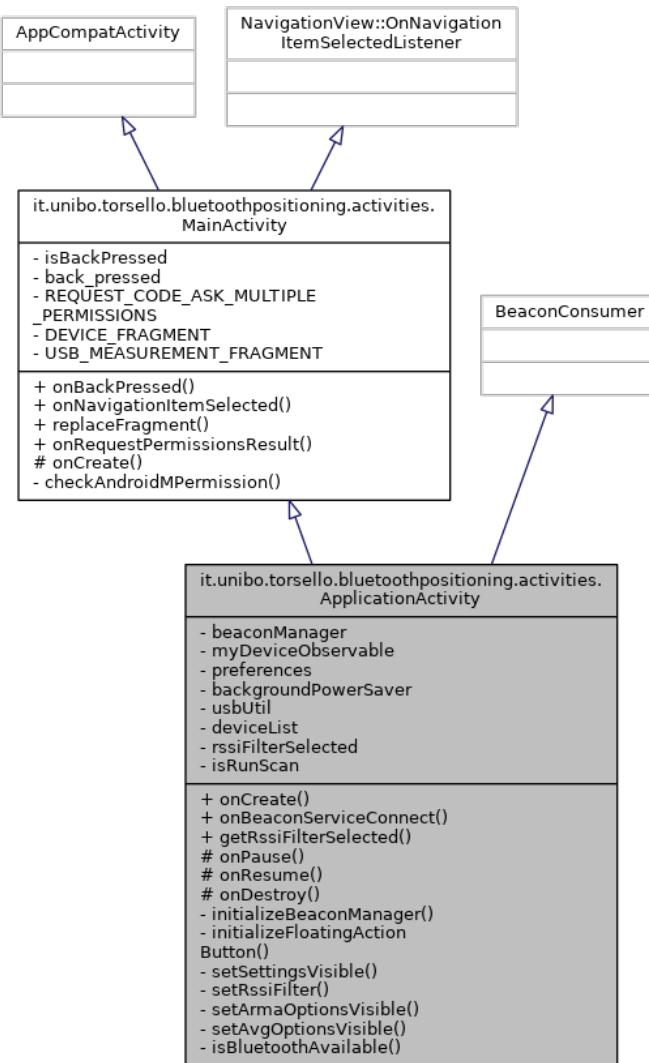


Figura 7.3: Classe - ApplicationActivity

Questa classe è una parte fondamentale del progetto. Qui si istanzia e controlla l'oggetto `BeaconManager` responsabile dello scanning su BT e quindi si ricevono le informazioni dei dispositivi vicini.

Alla ricezione dei dati, questi vengono spediti al `DeviceObservable` a cui tutti gli Observer interessati si registreranno in modo da ottenere tutti lo stesso valore aggiornato.

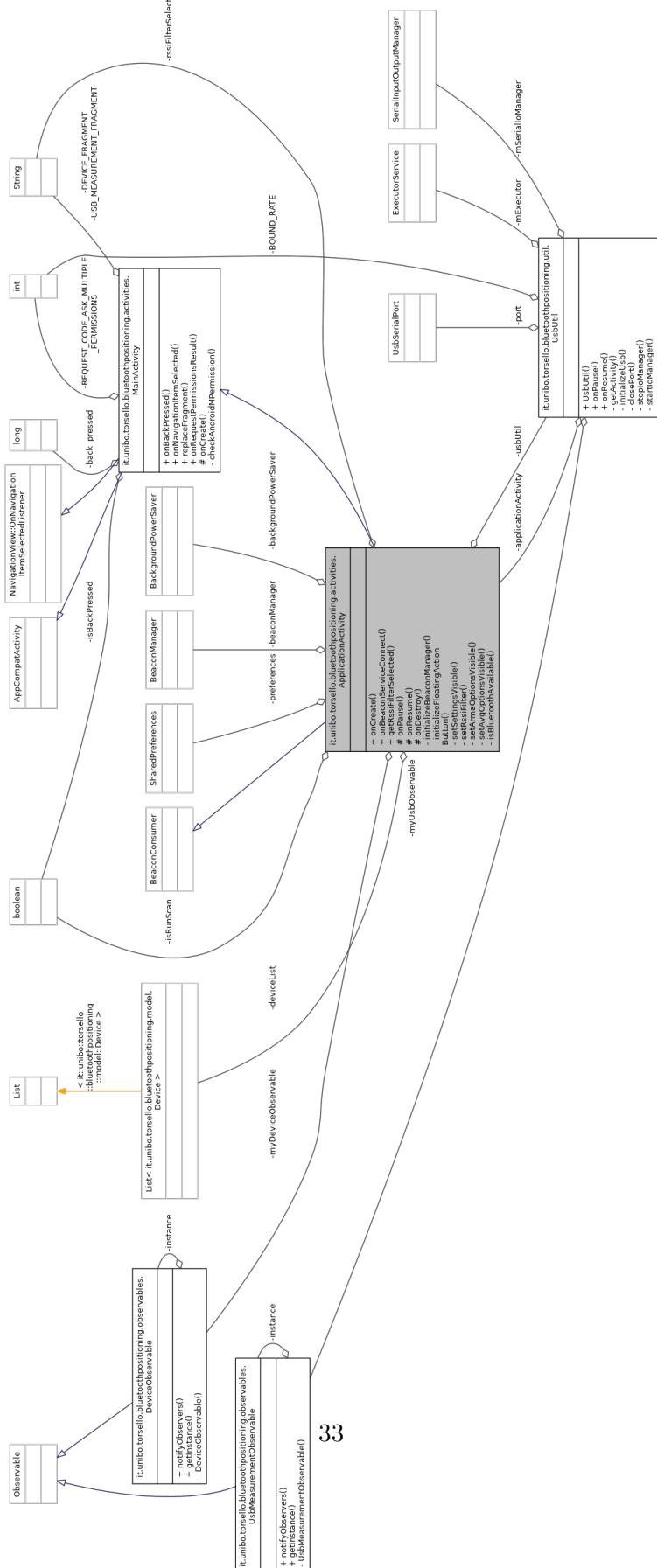


Figura 7.4: Collaborazione - ApplicationActivity

7.3 Device

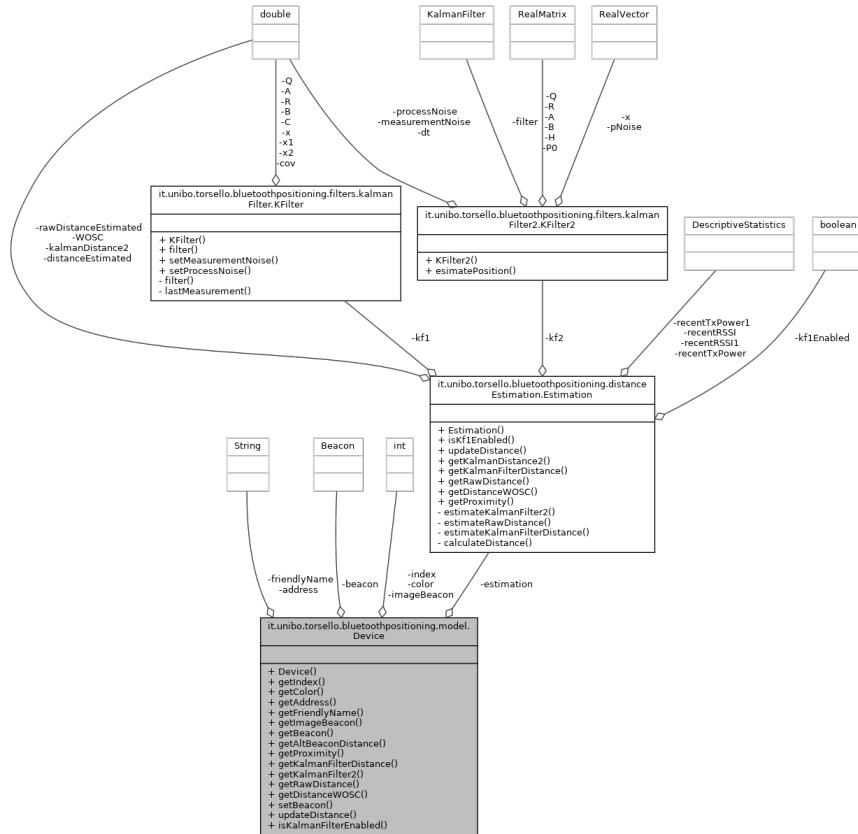


Figura 7.5: Collaborazione - Device

Classe che modella la definizione di dispositivo Bluetooth BLE nel sistema. Alcuni dei dati presenti nella classe sono inizializzati alla creazione che avviene nella classe `DeviceConstants`, altri sono invece calcolati durante l'esecuzione, come la stima della distanza da parte dei vari filtri.

7.4 DeviceConstants

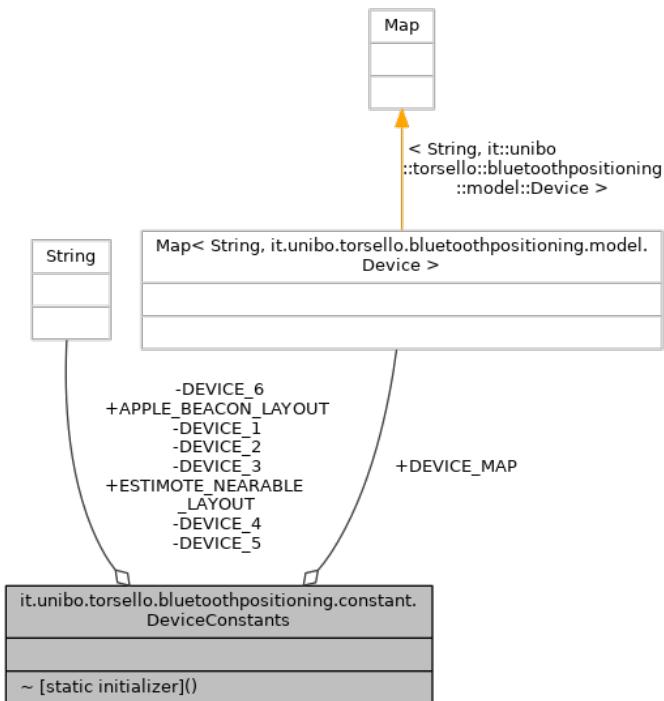


Figura 7.6: Collaborazione - DeviceConstants

Classe che modella ed inizializza i dispositivi sotto forma di oggetto `Device`. Al suo interno è presente una `Map` in cui sono indicati i soli ed unici dispositivi che saranno riconosciuti dal sistema.

7.5 DeviceObservable

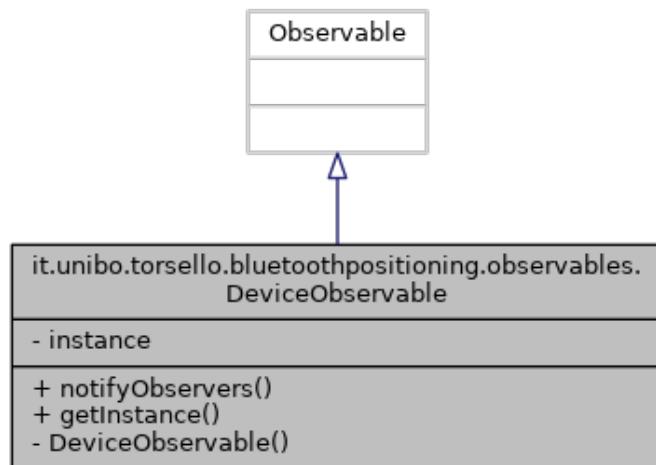


Figura 7.7: Classe - DeviceObservable

Lo scopo di questa classe è creare un Observable che notifichi a tutti gli Observer registrati l'oggetto list con tutti gli update dei dispositivi scansinati. Questa lista viene aggiornata nella classe `ApplicationActivity` nel metodo `onBeaconServiceConnect`

7.6 Estimation

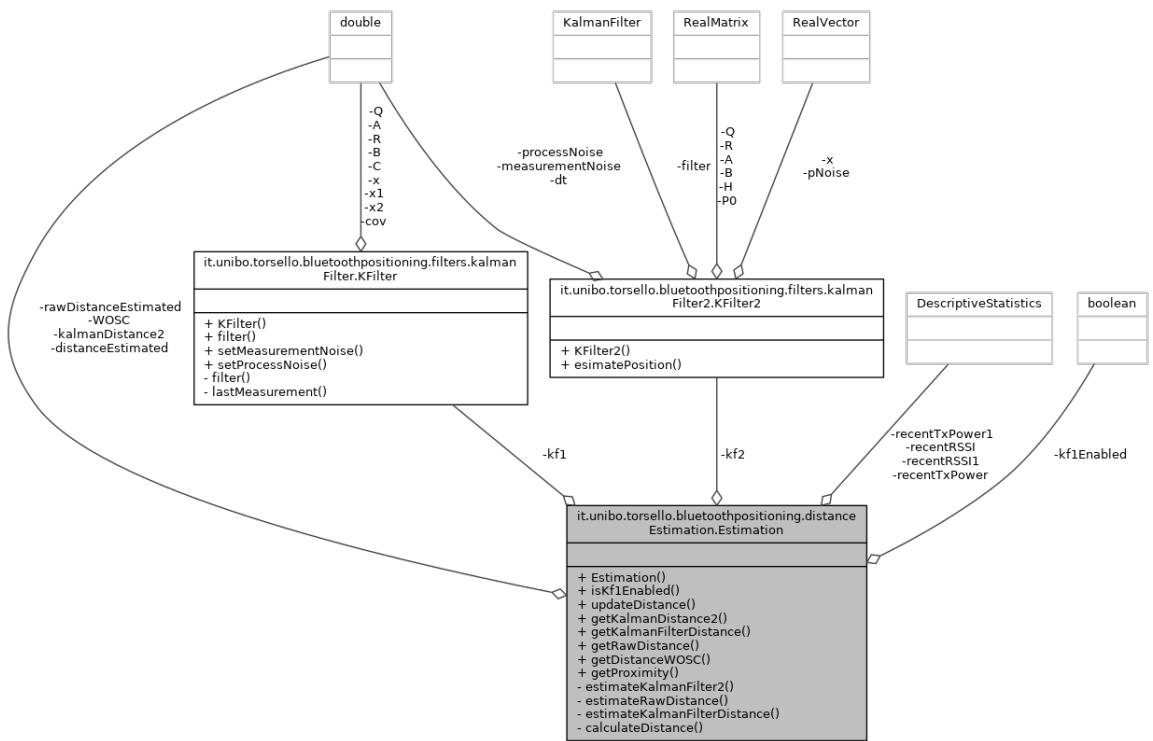


Figura 7.8: Collaborazione - Estimation

7.7 SettingsFragment

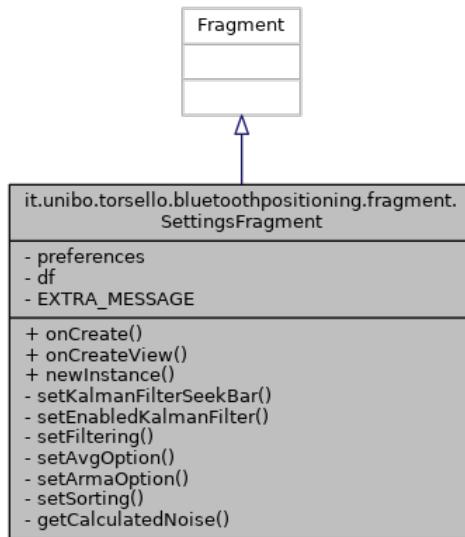


Figura 7.9: Classe - SettingsFragment

Classe che implementa Fragment per creare una vista disposta in un menu a destra. Il suo scopo è permettere all'utente di interagire con l'app e con il sistema fisico in cui ci si esegue la stima modificando dei semplici parametri.

Nel particolare si usa l'oggetto `SharedPreferences` per salvare le impostazioni utilizzate. Nel caso in cui l'app fosse chiusa, le impostazioni salvate precedentemente vengono reimpostate come se l'app non fosse mai stata terminata.

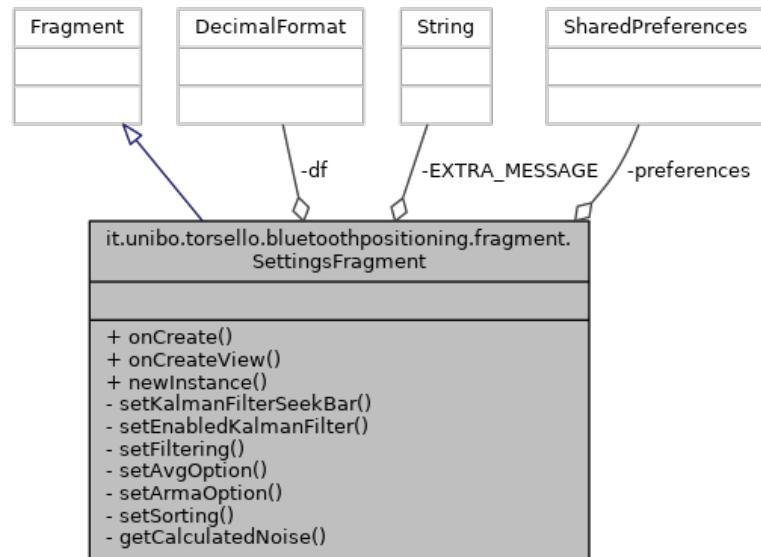


Figura 7.10: Collaborazione - SettingsFragment

7.8 SettingConstants

Classe in cui si settano le costanti relative ai settaggi. Tali costanti sono intese come chiavi per risalire alle impostazioni scelte dall'utente, come ad esempio il filtro o l'ordinamento dei dispositivi trovati.

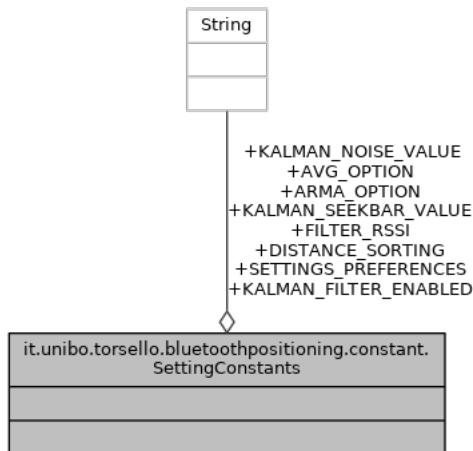


Figura 7.11: Collaborazione - SettingConstants

7.9 DeviceListFragment

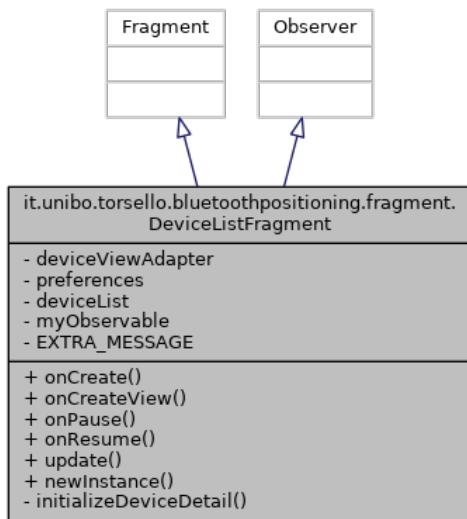


Figura 7.12: Classe - DeviceListFragment

Classe che aggiorna la lista dei dispositivi con le nuove informazioni ricevute dal metodo `update(...)`, *override* della classe `Observer`.

```
@Override
public void update(Observable o, Object arg) {

    if (arg instanceof List) {

        if (!deviceList.isEmpty()) {
            deviceList.clear();
        }

        List<Device> devices = (List<Device>) arg;

        // optional sorting
        Collections.sort(devices, new Comparator<Device>() {
            public int compare(Device b1, Device b2) {
                int sorting = preferences
                    .getInt(SettingConstants.DISTANCE_SORTING,
                            0);
                switch (sorting) {
                    case 0:
                    case R.id.radioButton_default_sorting:
                        return Double.compare(b1.getIndex(),
                                b2.getIndex());
                    case R.id.radioButton_color_sorting:
                        return Double.compare(b1.getColor(),
                                b2.getColor());
                }
            }
        });
    }
}
```

```
        case R.id.radioButton_distance_sorting:
        return
            Double.compare(b1.getKalmanFilterDistance(),
                           b2.getKalmanFilterDistance());
    } // default sorting (a good basic ordering
      for the other options)
    return Double.compare(b1.getIndex(),
                          b2.getIndex());
}
});

deviceList.addAll(devices);
deviceViewAdapter.notifyDataSetChanged();
}
}
```

Nel dettaglio si aggiorna la lista in base alle preferenze impostate in **Settings**.

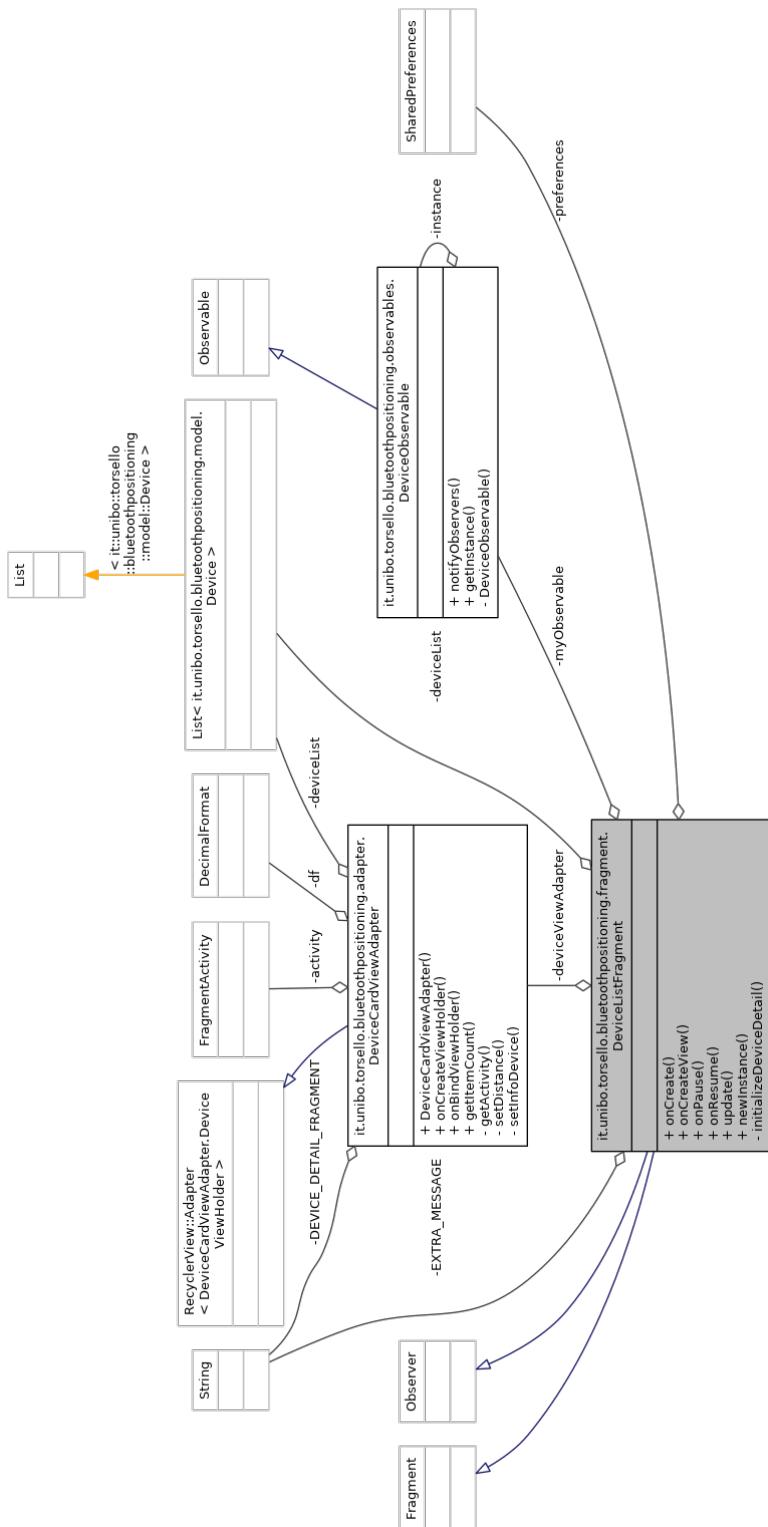


Figura 7.13: Collaborazione - DeviceListFragment

7.10 DeviceCardViewAdapter

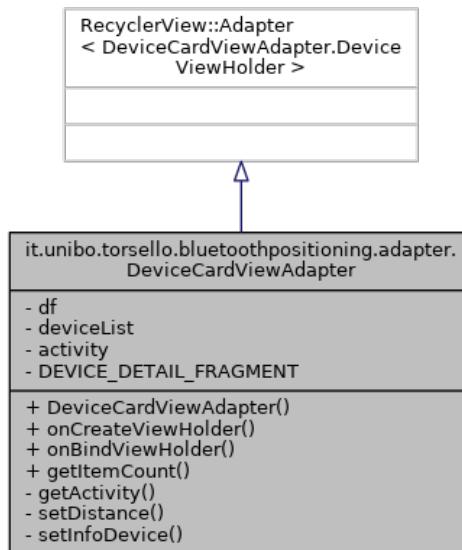


Figura 7.14: Classe - DeviceCardViewAdapter

Classe responsabile della visualizzazione delle nuove informazioni a schermo. Qui si gestiscono gli elementi grafici da fare vedere o nascondere (valori testuali o immagini) nella RecyclerView.

Viene utilizzata nella schermata principale e nel fragment dei dettagli.

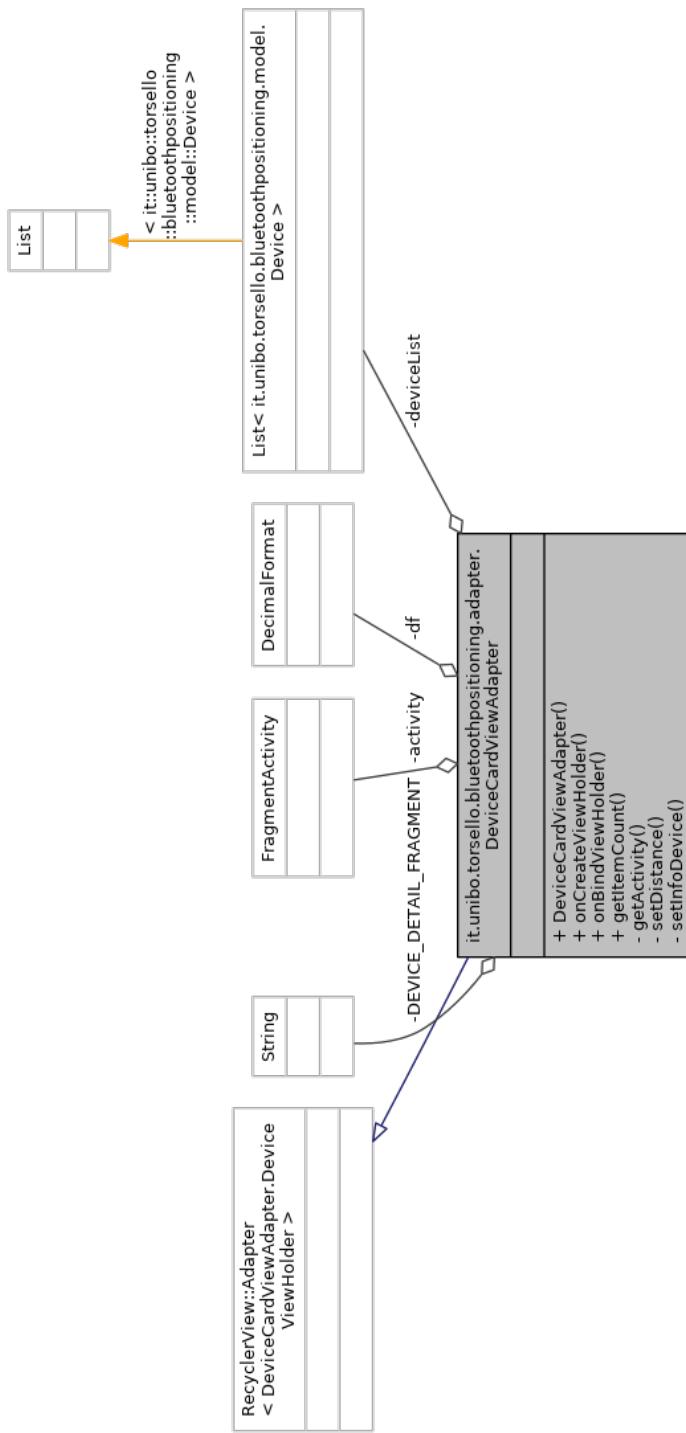


Figura 7.15: Collaborazione - DeviceCardViewAdapter

7.11 DeviceViewHolder

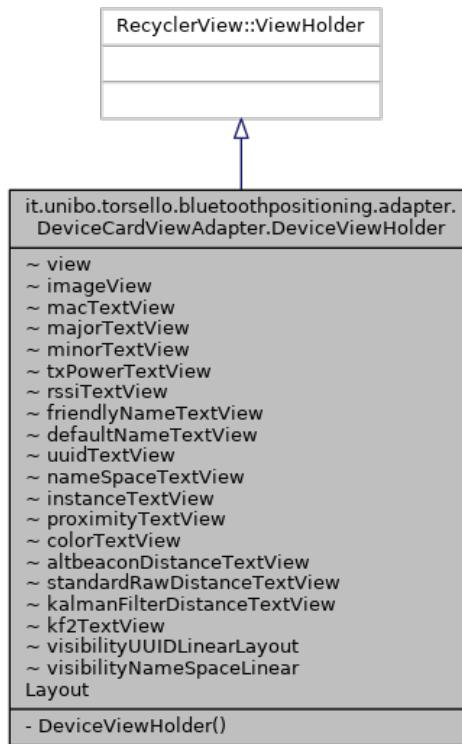


Figura 7.16: Classe - DeviceViewHolder

Classe che serve ad inizializzare i vari oggetti grafici da visualizzare nel RecyclerView, risparmiando memoria.

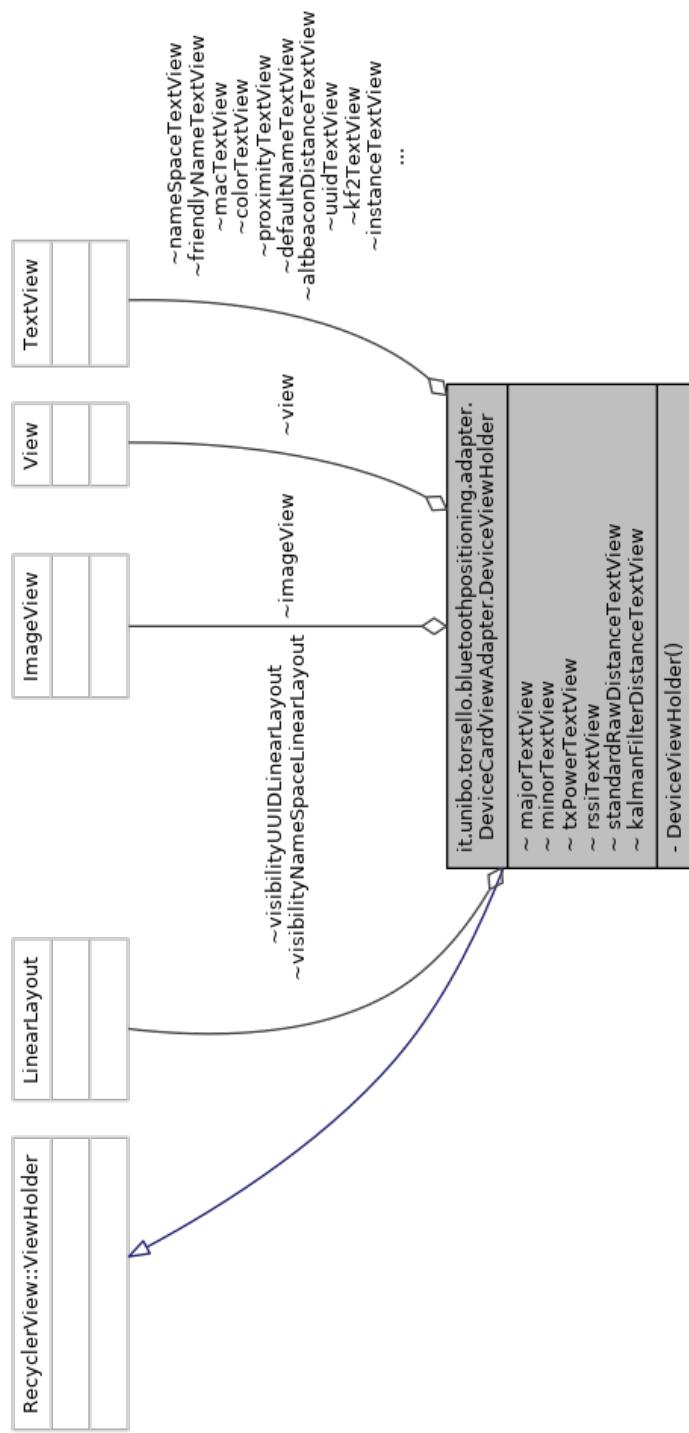


Figura 7.17: Collaborazione - DeviceViewHolder

7.12 MyArmaRssiFilter

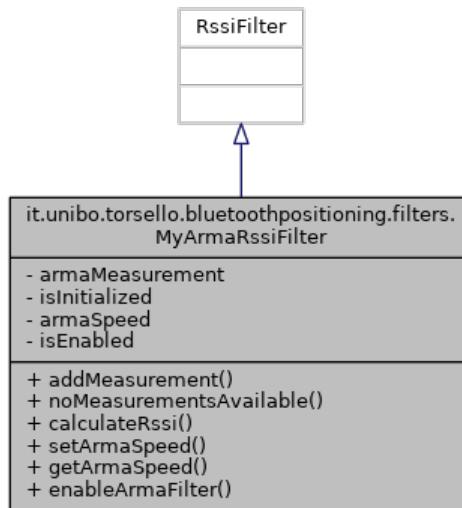


Figura 7.18: Classe - MyArmaRssiFilter

Classe in cui si esegue il filtraggio ARMA spiegato in 5.5.3.

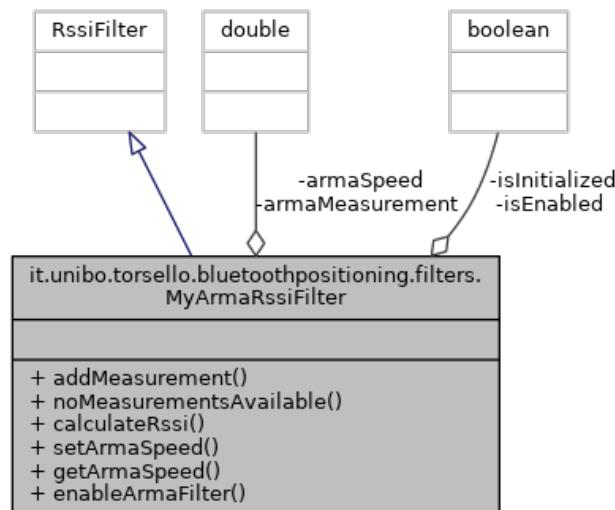


Figura 7.19: Collaborazione - MyArmaRssiFilter

7.13 KFilterBuildertFragment

Classe di supporto per la creazione di un **KFilter**. Il suo scopo è inizializzare tale oggetto con dei parametri passati al costruttore.

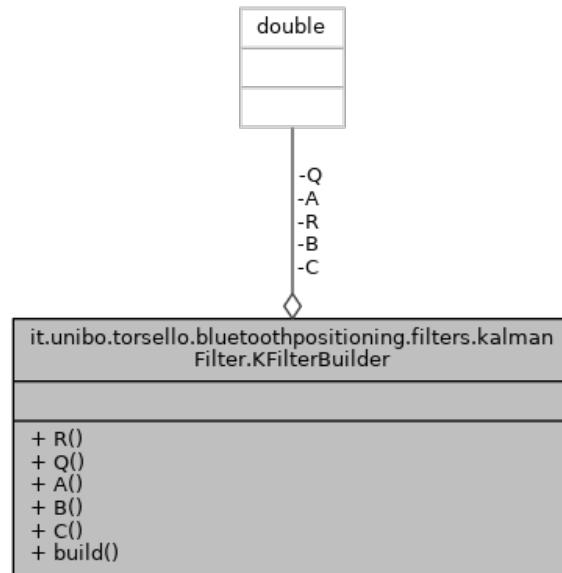


Figura 7.20: Collaborazione - KFilterBuildertFragment

7.14 KFilter

Classe per l'implementazione del filtro di Kalman. Il suo funzionamento viene spiegato in 5.5.2.

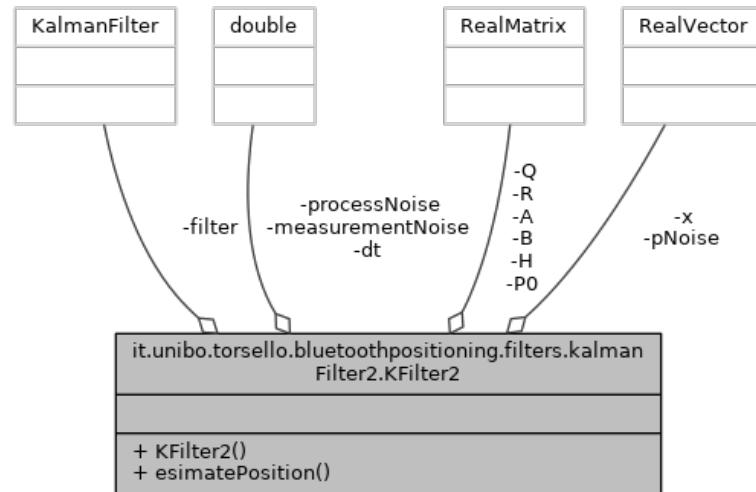


Figura 7.21: Collaborazione - KFilter

7.15 KFilterConstants

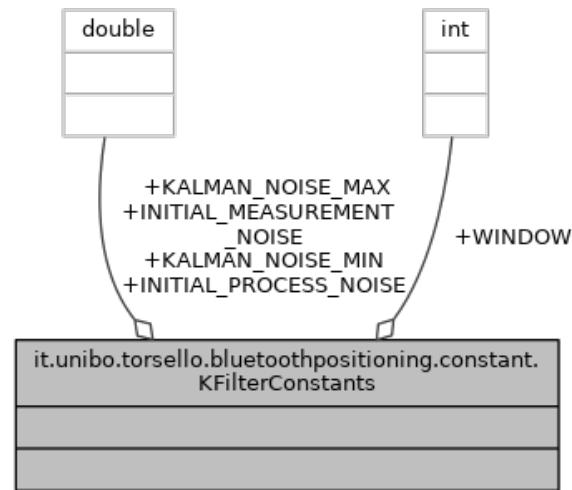


Figura 7.22: Collaborazione - KFilterConstants

7.16 StatePagerAdapter

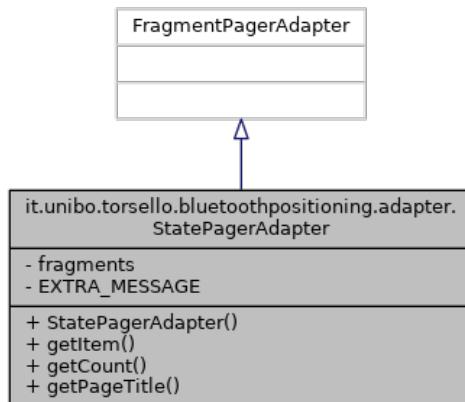


Figura 7.23: Classe - StatePagerAdapter

Questa classe estende `FragmentPagerAdapter` per creare un ambiente in cui aggiungere i fragment dei dettagli riguardanti i beacon.

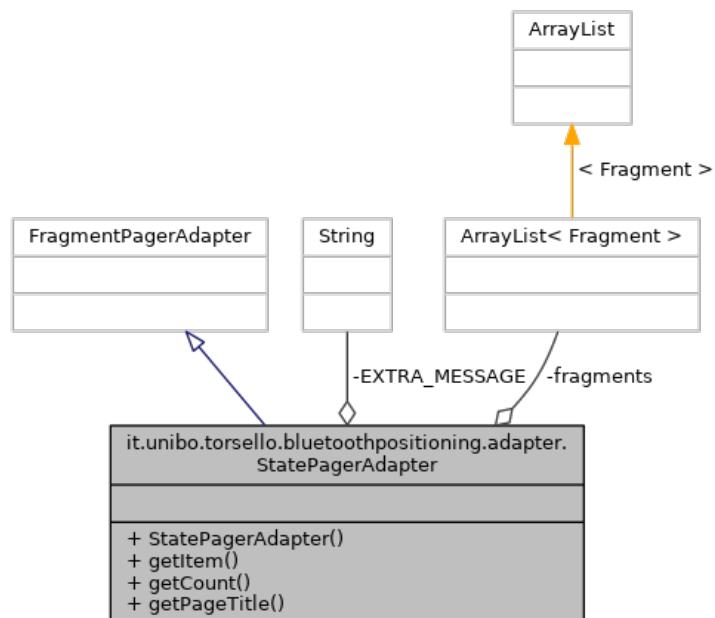


Figura 7.24: Collaborazione - StatePagerAdapter

7.17 DeviceDetailFragment

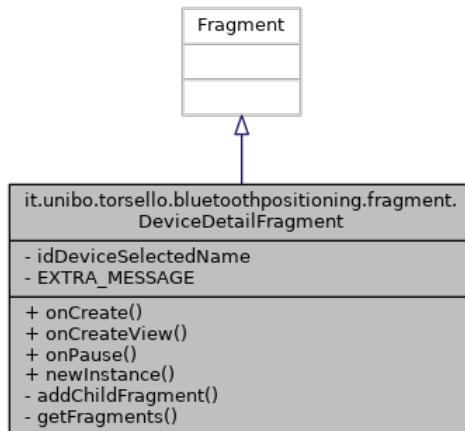


Figura 7.25: Classe - DeviceDetailFragment

Classe che istanzia `StatePagerAdapter` per aggiunge i fragment dei dettagli.

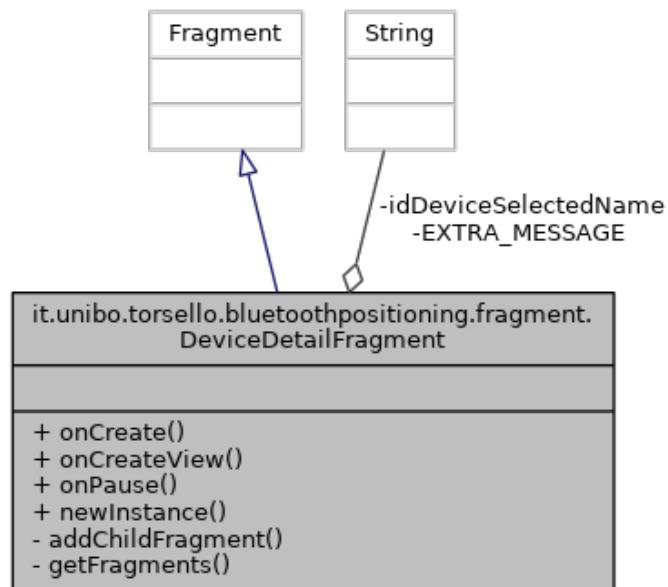


Figura 7.26: Collaborazione - DeviceDetailFragment

7.18 CameraFragment

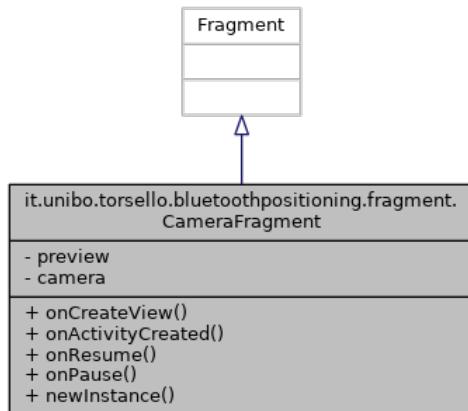


Figura 7.27: Classe - CameraFragment

Classe per implementare una SurfaceView come preview delle fotocamera presente nello smartphone e scattare delle foto (agli iBeacon). Per scattare le foto si è utilizzato un FloatingActionButton, mentre per fare l'autozoom basta cliccare sulla surface stessa.

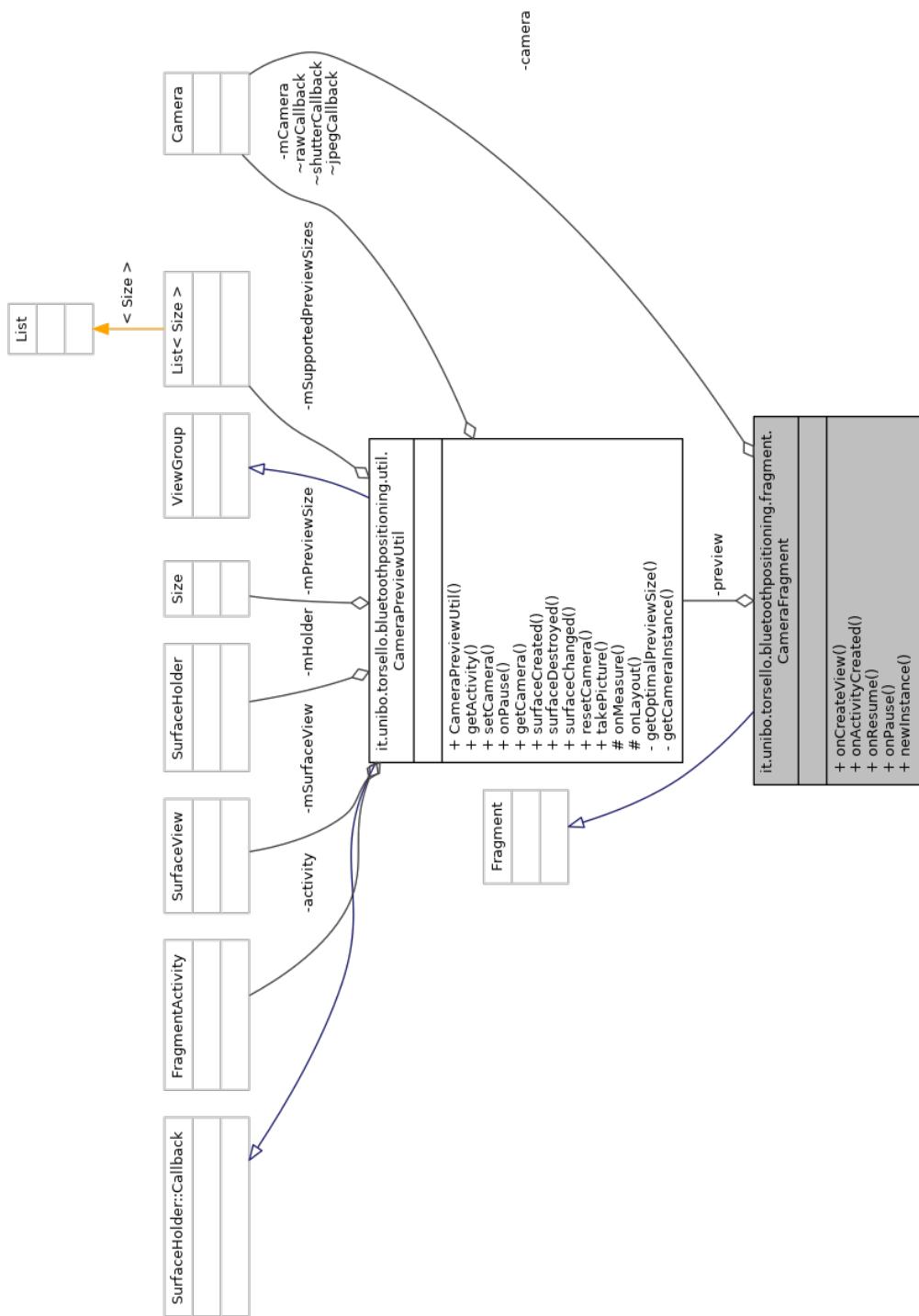


Figura 7.28: Collaborazione - CameraFragment

7.19 CameraPreviewUtil

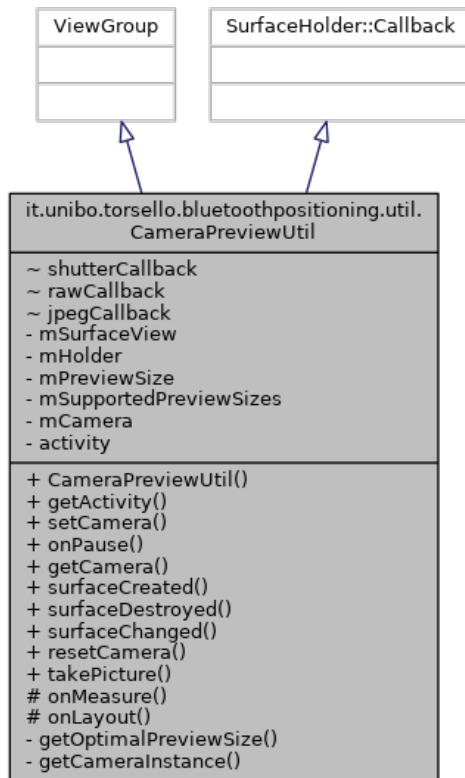


Figura 7.29: Classe - CameraPreviewUtil

Classe che controlla la `SurfaceView` di preview delle fotocamera e la telecamera stessa. Viene istanziata in `CameraFragment`.

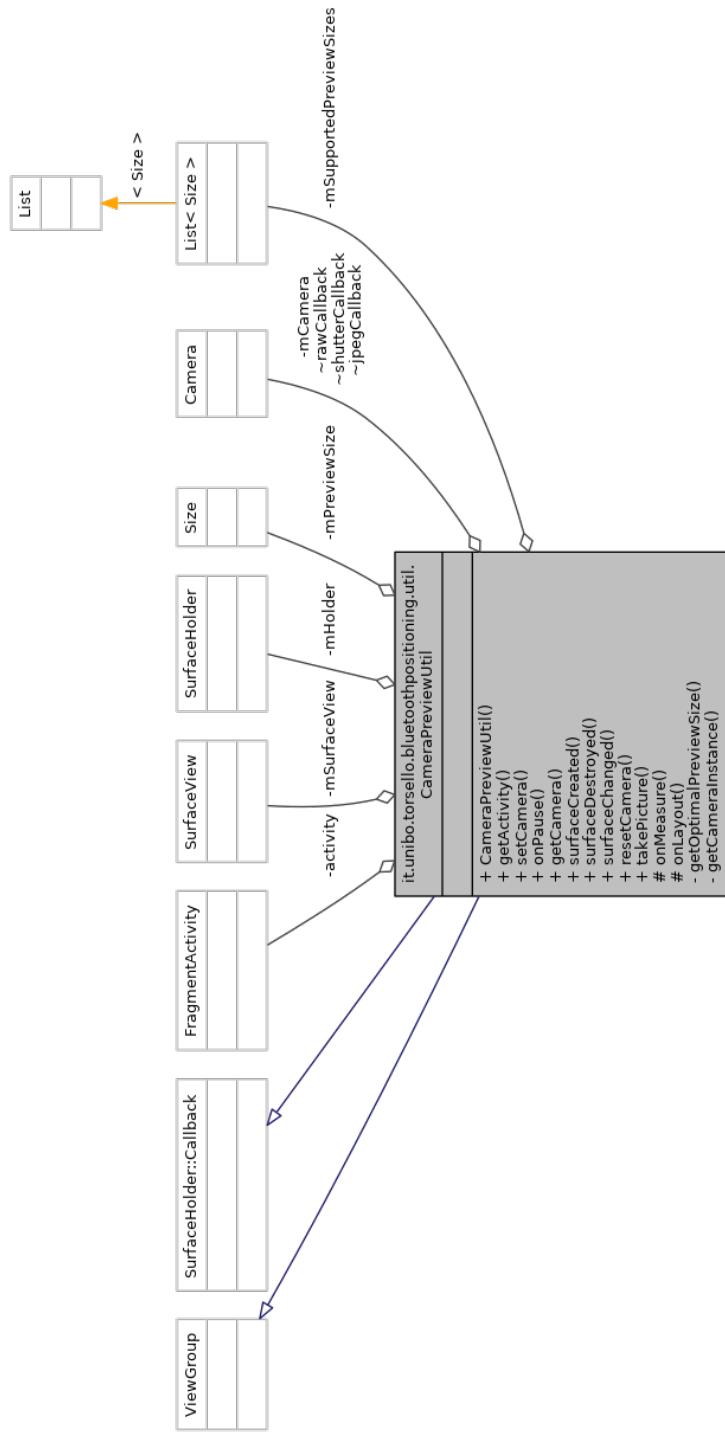


Figura 7.30: Collaborazione - CameraPreviewUtil

7.20 SaveImageTask

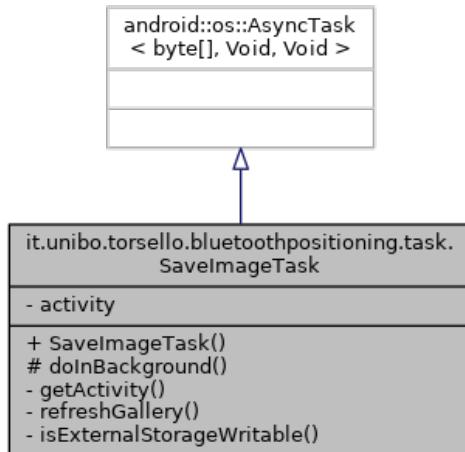


Figura 7.31: Classe - SaveImageTask

Classe che serve a salvare le foto scattate dalla fotocamera. Implementa un AsyncTask per evitare che la GUI di Android si blocchi durante l'elaborazione dell'immagine.

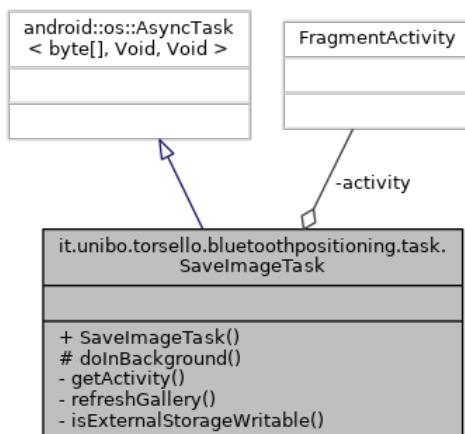


Figura 7.32: Collaborazione - SaveImageTask

7.21 DeviceDetailInner1Fragment

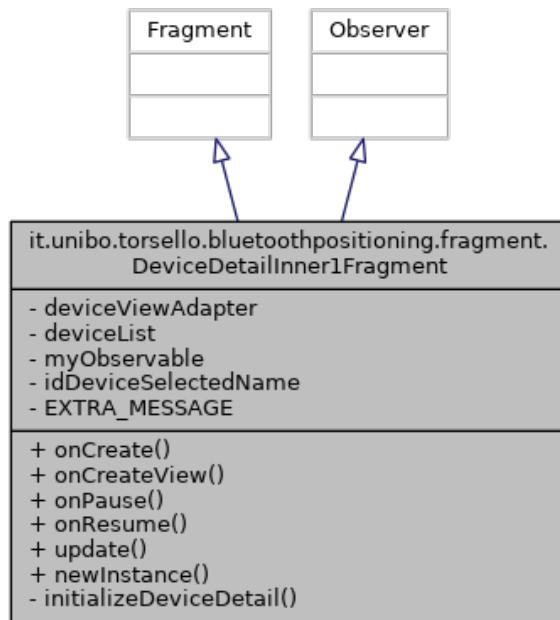


Figura 7.33: Classe - DeviceDetailInner1Fragment

Simile a 7.9, aggiorna i dati di un solo beacon e visualizza la distanza dall'Arduino. Per la distanza stimata da Arduino si utilizza il fragment `UsbMeasurementFragment`, istanziato come fragment statico nel file XML.

7.22 DeviceDetailInner2Fragment

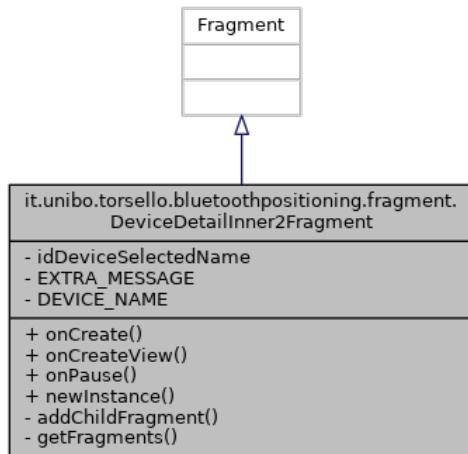


Figura 7.34: Classe - DeviceDetailInner2Fragment

Questa classe estende Fragment per inserire al proprio interno i dettagli dei beacon sotto forma di grafici in real time. L'obiettivo è permettere all'utente di controllare l'evoluzione del sistema e salvare i grafici sotto forma di immagini JPG.

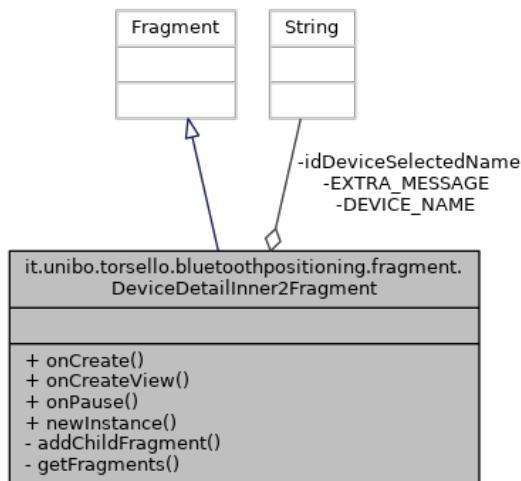


Figura 7.35: Collaborazione -

7.23 DeviceChartFragment

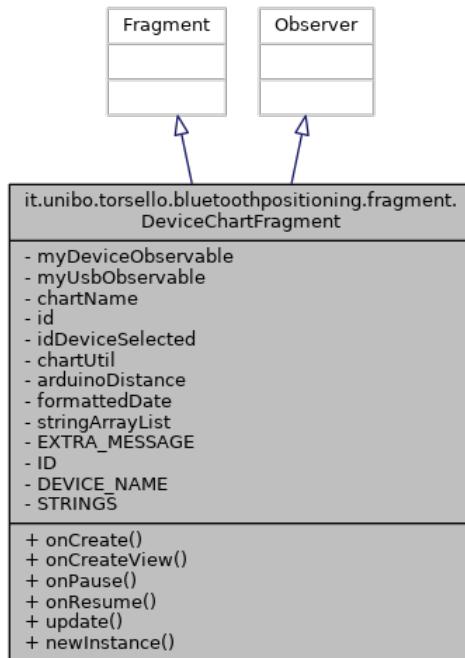


Figura 7.36: Classe - DeviceChartFragment

Classe in cui si generano i grafici in realtime. I valori che vengono plottati sono le stime della distanza secondo Arduino e in base ai vari filtri su RSSI.

7.24 ChartUtil

Classe per la creazione di grafici in real time. Viene sfuttata da `DeviceChartFragment` per istanziare un oggetto `LinearChart` a cui passare le varie distanze stimate via RSSI e USB.

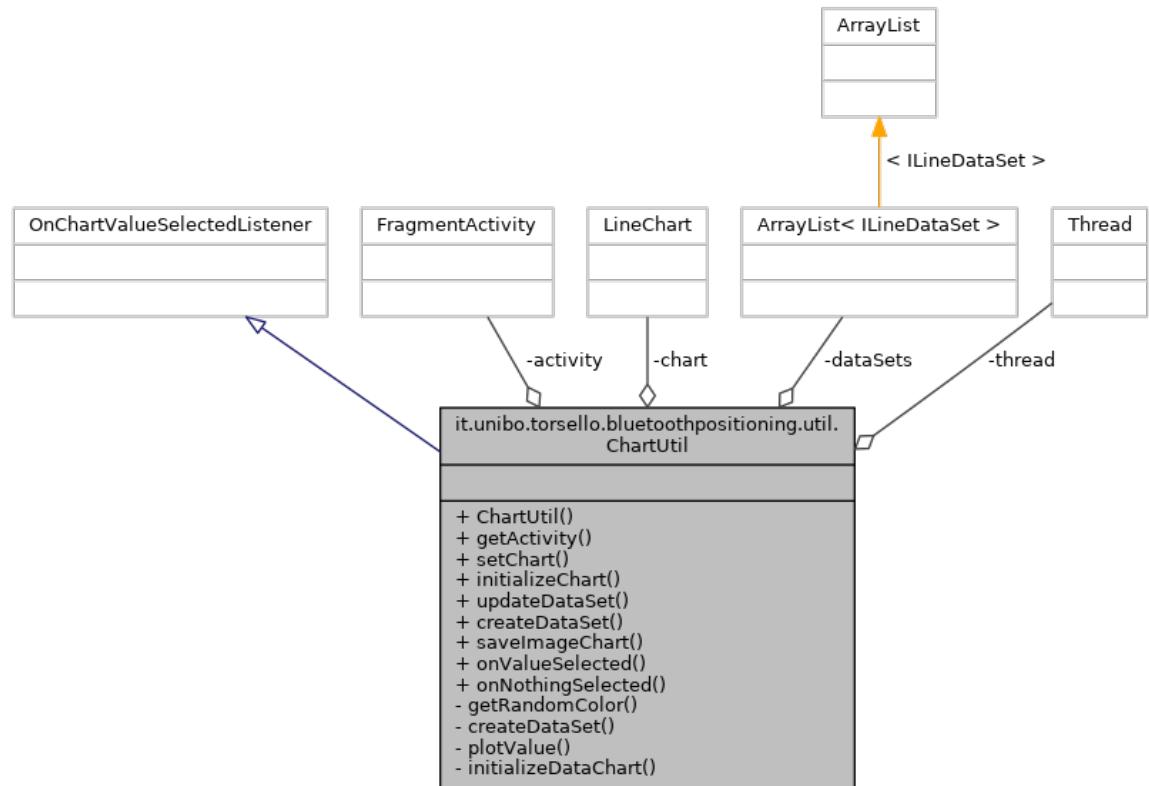


Figura 7.37

7.25 UsbMeasurementFragment

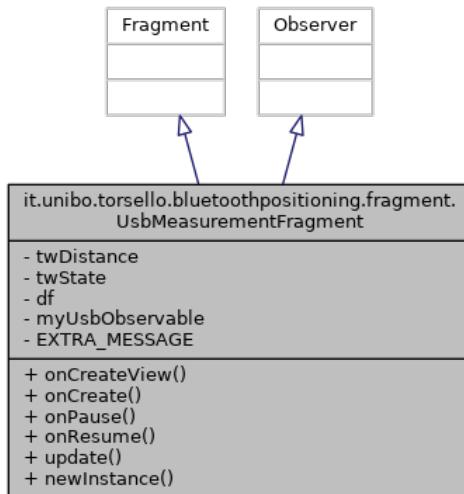


Figura 7.38: Classe - UsbMeasurementFragment

Classe che visualizza su di un fragment i valori di distanza stimati dall'Arduino. Questa visualizzazione viene riutilizzata da sola se si seleziona "Measurement" dal menu a sinistra che nei dettagli. Nel caso dei dettagli il fragment è istanziato come statico dal file XML come segue:

```
<android.support.v7.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/card_margin">

    <fragment android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="it.unibo.torsello.bluetoothpositioning
            .fragment.usbObservers.UsbMeasurementFragment"
        android:id="@+id/usbArduino"
        tools:layout="@layout/fragment_usb_measurement" />
</android.support.v7.widget.CardView>
```

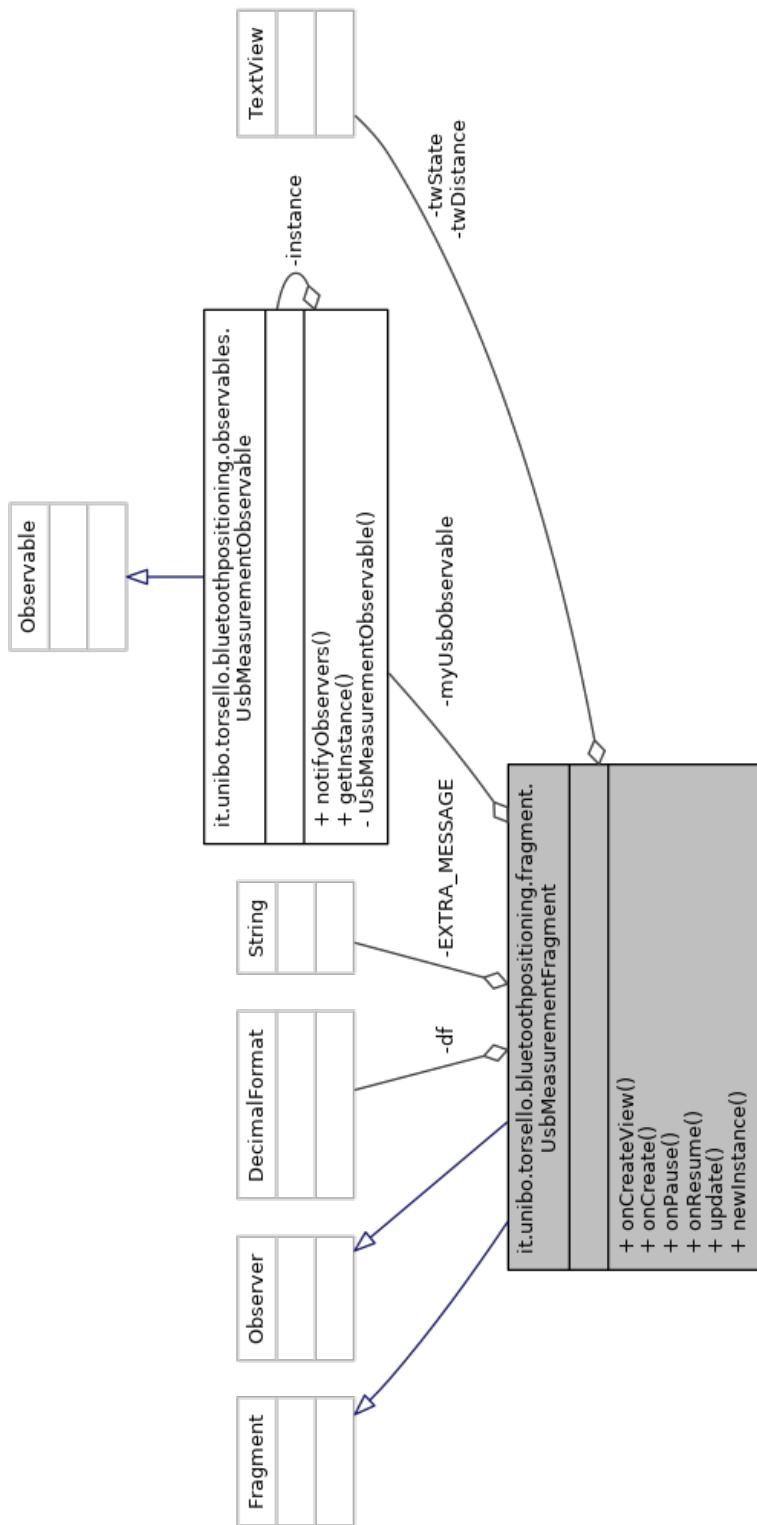


Figura 7.39: Collaborazione - UsbMeasurementFragment

7.26 UsbUtil

Classe che permette di comunicare con la porta USB dello smartphone e in questo caso di ricevere i dati della distanza stimata da Arduino.

Per inizializzare la comunicazione su USB si utilizza il metodo `initializeUsb()`

Metodo `initializeUsb()`

```
private void initializeUsb() {

    // Find all available drivers from attached devices.
    UsbManager usbManager = (UsbManager)
        getActivity().getSystemService(Context.USB_SERVICE);
    List<UsbSerialDriver> availableDrivers =
        UsbSerialProber.getDefaultProber()
            .findAllDrivers(usbManager);

    if (!availableDrivers.isEmpty()) {

        // Open a connection to the first available driver.
        UsbSerialDriver driver = availableDrivers.get(0);

        if (usbManager.hasPermission(driver.getDevice())) {
            if (usbManager.openDevice(driver.getDevice())
                != null) {
                // Read some data! Most have just one port
                // (port 0).
                port = driver.getPorts().get(0);
            }
        } else {
            Intent startIntent = new Intent(getActivity(),
                getClass());
            PendingIntent pendingIntent =
                PendingIntent.getService(getActivity(), 0,
                    startIntent, 0);
            usbManager.requestPermission(driver.getDevice(),
                pendingIntent);
        }
    }

    if (port != null) {

        UsbDeviceConnection connection =
            usbManager.openDevice(port.getDriver()
                .getDevice());

        if (connection != null) {
            try {
                port.open(connection);
            }
        }
    }
}
```

```
        port.setParameters(BOUND_RATE,
                           UsbSerialPort.DATABITS_8,
                           UsbSerialPort.STOPBITS_1,
                           UsbSerialPort.PARITY_NONE);
    } catch (IOException e) {
        myUsbObservable.notifyObservers(getActivity()
            .getString(R.string.error_opening_device)
            + " " + e.getMessage());
        myUsbObservable.notifyObservers(false);
        closePort();
        return;
    }

    stopIoManager();
    startIoManager();
}
}
}
```

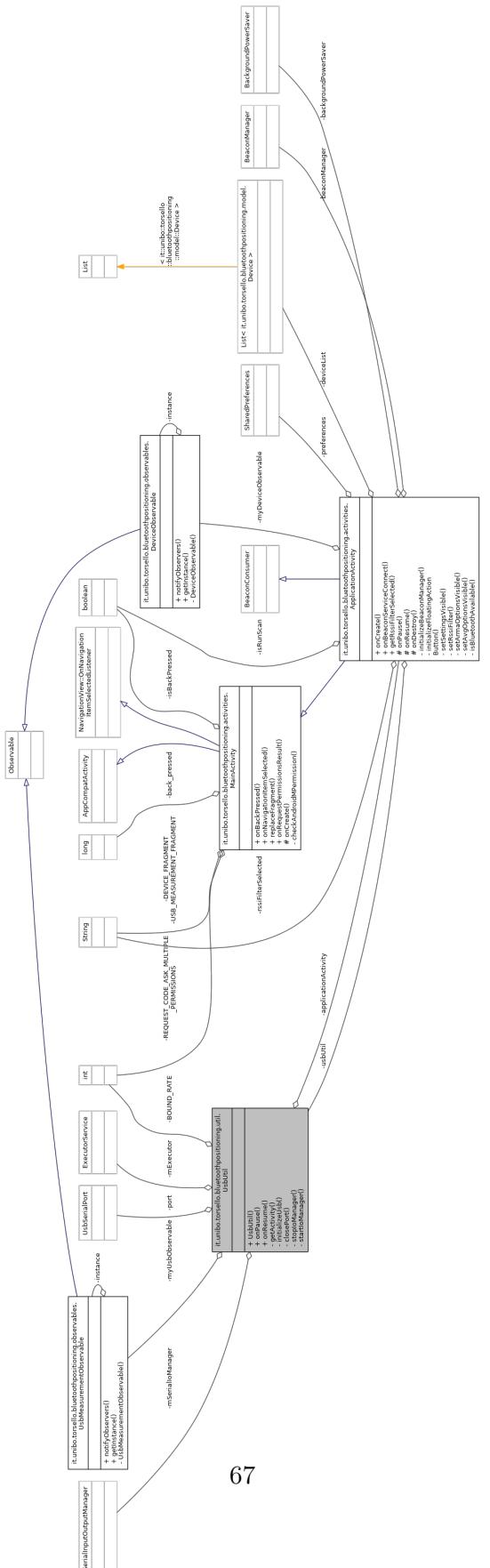


Figura 7.40: Collaborazione - UsbUtil

7.27 UsbMeasurementObservable

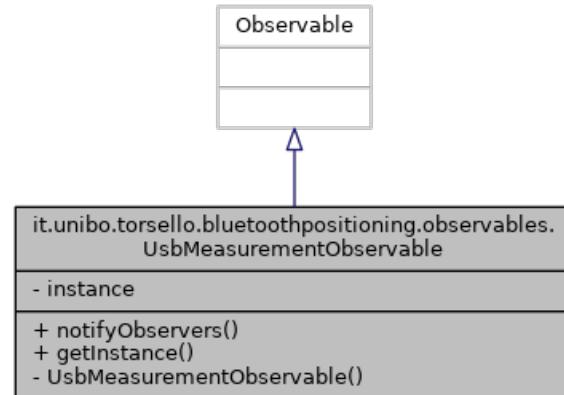


Figura 7.41: Classe - UsbMeasurementObservable

Classe Observable che emette le distanze stimate da Arduino a tutti gli Observer a lui registrati.

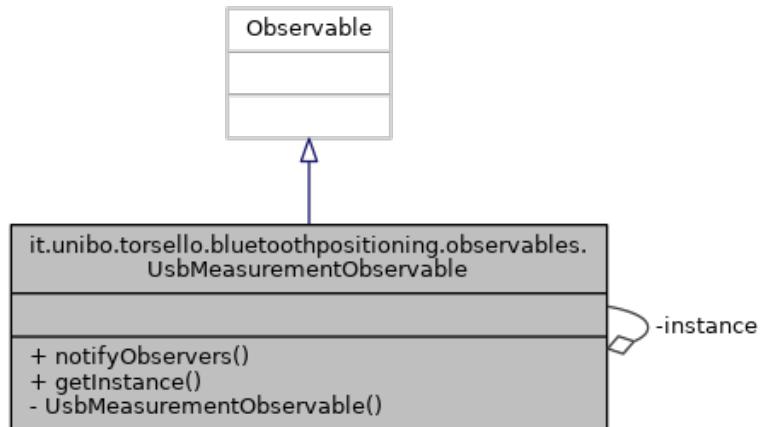


Figura 7.42: Collaborazione - UsbMeasurementObservable

7.28 FABBehavior

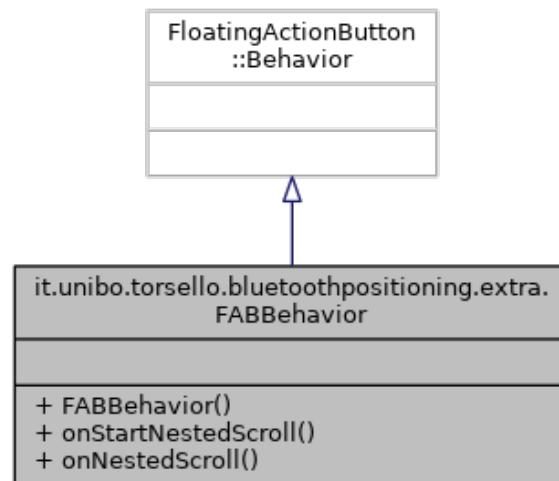


Figura 7.43: Classe - FABBehavior

Classe che amplia il comportamento del FloatingActionButton in basso a destra, facendolo scomparire per un secondo quando una RecyclerView viene scrollata in alto o in basso. Per funzionare si è dovuto modificare il file XML in cui il FAB è inserito con la regola `app:layout_behavior=".extra.FABBehavior"`.

Parte IV

Implementazione

Capitolo 8

GUI dell'app

8.1 Avvio dell'app

Quando si avvia l'app viene visualizzato un avviso che invita l'utente a premere sul bottone in basso a destra per far partire la scansione dei dispositivi BT.

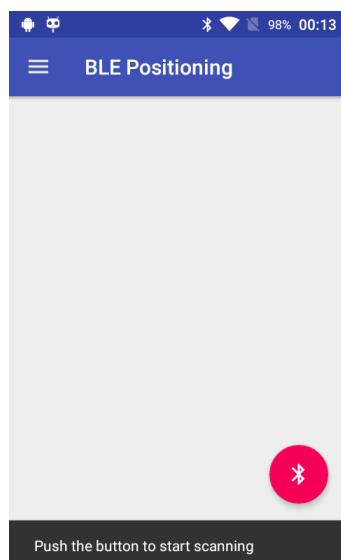


Figura 8.1: Avvio dell'app

8.2 Abilitazione della radio Bluetooth

Questa schermata viene visualizzata se la radio BT è spenta. Appare all'avvio dell'app o di una nuova scansione.

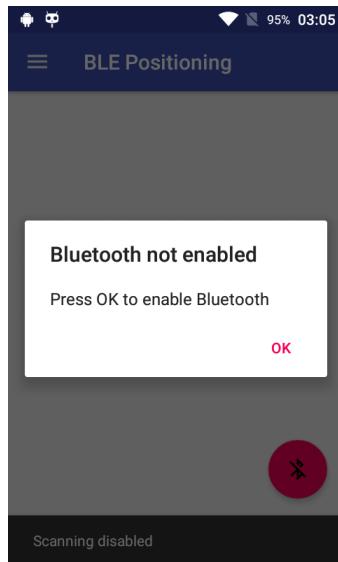


Figura 8.2: Abilitazione della radio Bluetooth

8.3 Scansione dei dispositivi

In questo fragment viene caricata la RecycleView con i dati dei dispositivi. Questa lista può essere ordinata in base ai settaggi scelti. Se si seleziona un dispositivo vengono visualizzati i dettagli e grafici per mettere in relazione le diverse stime sulla distanza.

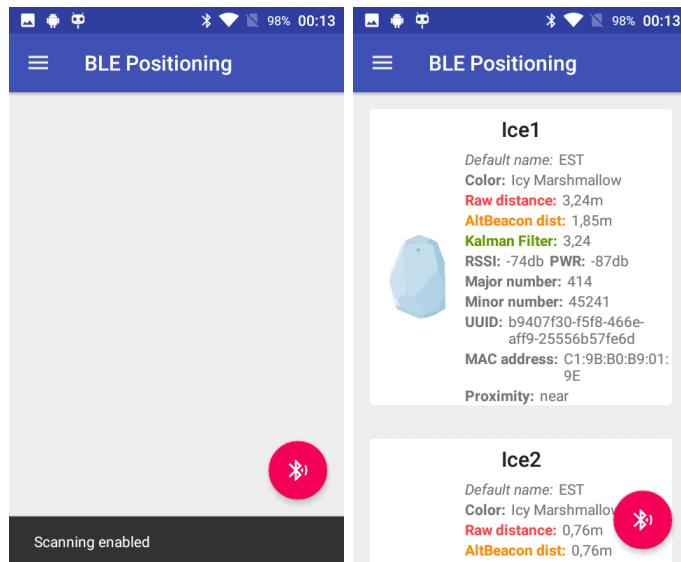


Figura 8.3: Scansione dei dispositivi

8.4 Menù a sinistra

In questo menù viene data la possibilità di tarare la distanza con Arduino (campo Measurement), visualizzare i settings oppure ritornare alla vista principale.

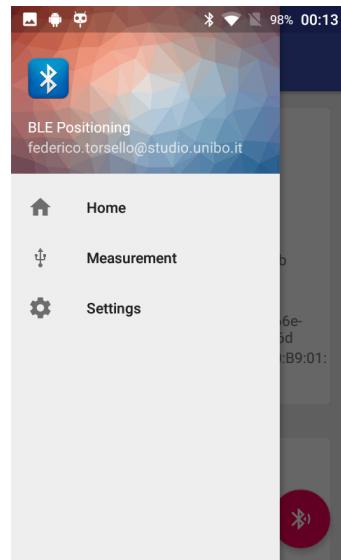


Figura 8.4: Menù a sinistra

8.5 Menù a destra

Il menù a destra consiste in un fragment in cui l'utente può selezionare le proprie preferenze. Nel caso in cui la scansione fosse interrotta, i settaggi risultano non abilitati.

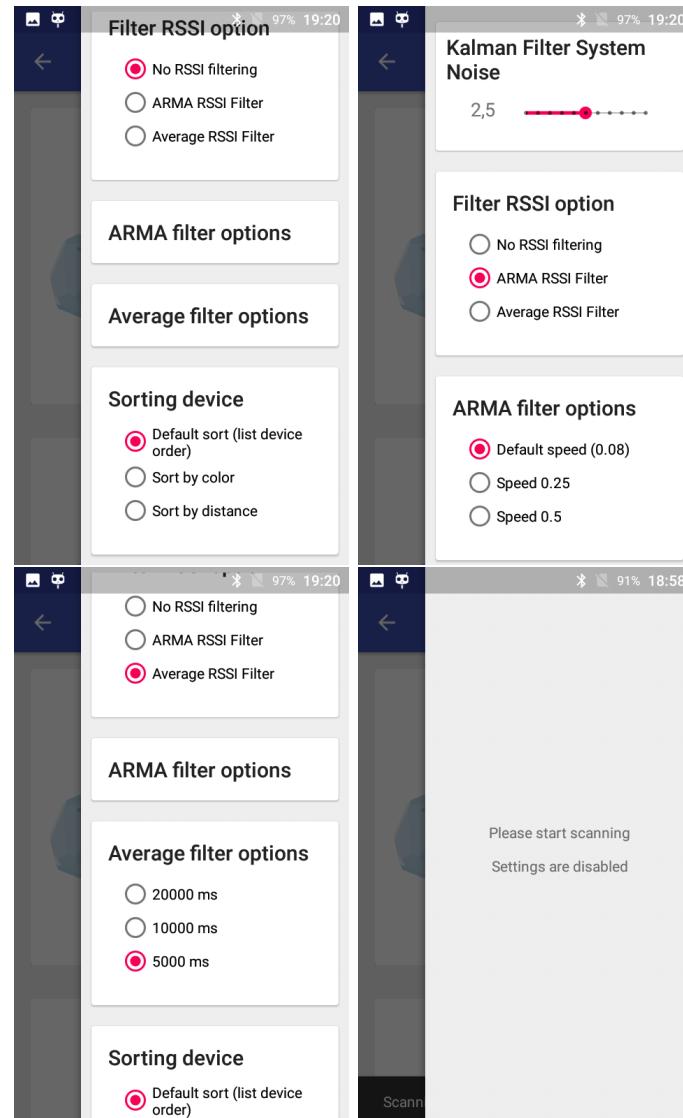


Figura 8.5: Menù a destra

8.6 Foto ad un dispositivo

Possibilità di scattare una foto a un dispositivo BT per ricordare dove è stato posizionato.

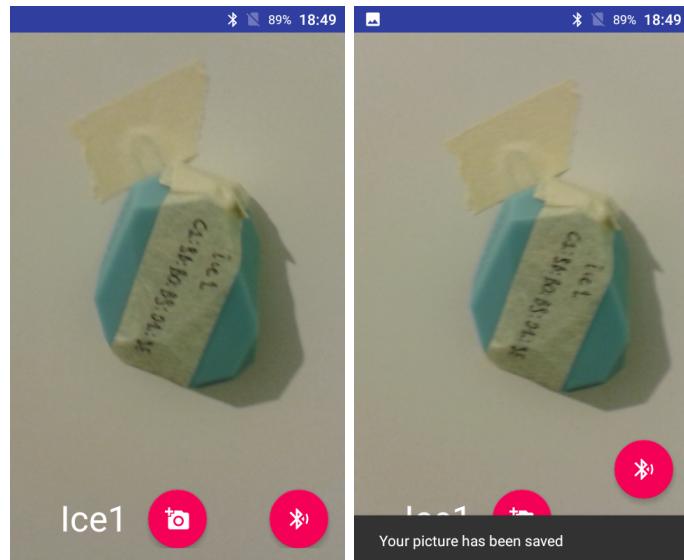


Figura 8.6: Foto ad un dispositivo

8.7 Dettagli e grafici real time

Il fragment dei dettagli è diviso in due parti, la parte a sinistra indica i dettagli del dispositivo selezionato e la distanza stimata dall'Arduino, a destra vi è sempre la stima di Arduino ma sono indicati dei grafici.

Tali grafici sono realizzati in real time. Visualizzano l'andamento della stima delle distanze stimate.

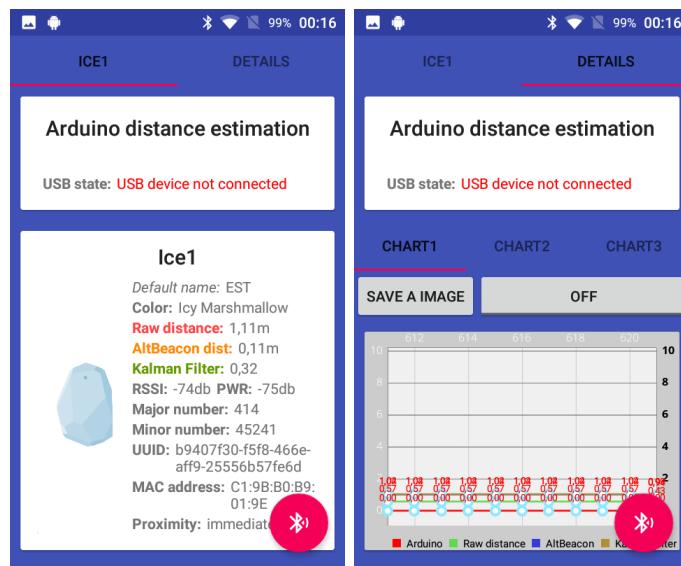


Figura 8.7: Dettagli e grafici real time

8.8 Connessione con Arduino via USB OTG

Con questi messaggi si richiede all'utente di abilitare la comunicazione USB sul dispositivo. Senza questo abilitazione la comunicazione con l'Arduino non può avvenire.

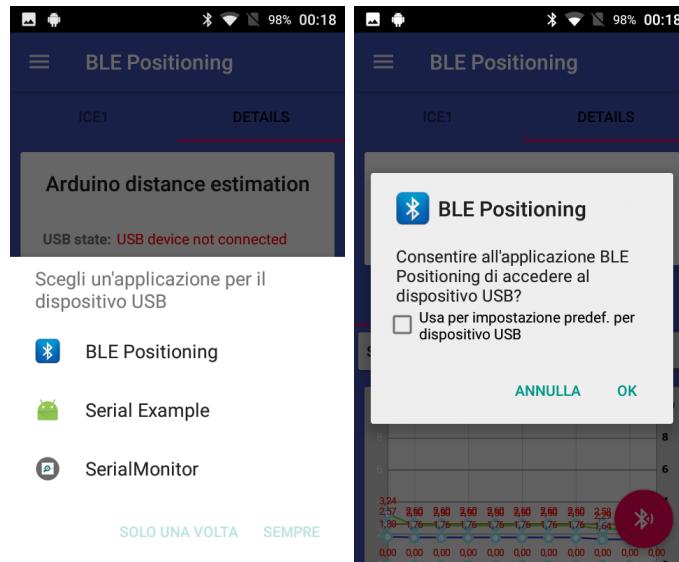


Figura 8.8: Connessione con Arduino via USB OTG

8.9 Feedback della stima della distanza con Arduino

Qui si fa vedere come appare la stima della distanza con Arduino quando il cavo è connesso e la comunicazione USB con OTG è abilitata.

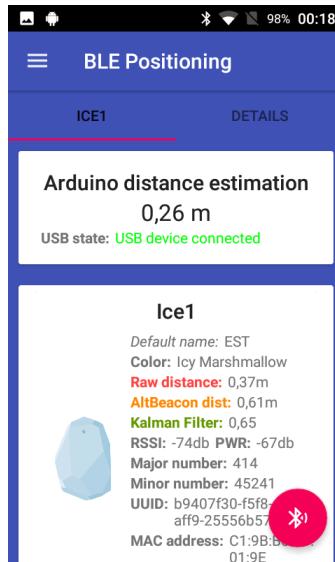


Figura 8.9: Feedback della stima della distanza con Arduino

Capitolo 9

Testing

9.1 Configurazione dell'ambiente indoor

La configurazione dell'ambiente consiste nella disposizione dei dispositivi Bluetooth nel modo corretto. Per poter avere un ricezione dei segnali inviati dagli iBeacon senza troppi disturbi, questi devono essere così 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.

9.2 Collegamento Arduino-smartphone

Come si può vedere nella Fig. 9.1 viene indicato il collegamento diretto Arduino-smartphone

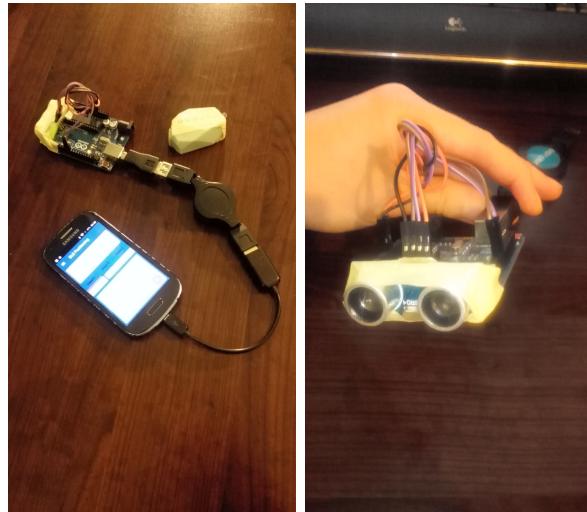


Figura 9.1: Collegamento Arduino-smartphone

Questo approccio permette di avere un sensore di prossimità che restituisce una stima della distanza confrontabile con la stima eseguita con la tecnica RSSI.

9.2.1 Materiale utilizzato

- Samsung GT-I9190, [CyanogenMod 12.1-20160718-UNOFFICIAL-golden Android 5.1.1](#)¹.
- Sensore di prossimità ultrasonico HC-SR04.
- Cavo USB OTG (~ 2€).



Figura 9.2: Materiale utilizzato

¹[Novafusion](http://novafusion.pl/s3-mini/) - <http://novafusion.pl/s3-mini/>

9.2.2 Dati raccolti

	Stima (m)	Deviazione (m)
Min	4,54	+1.2m
Max	2	1
Avg	2	3

Bibliografia

- [1] Matthew S. Gast, *Building Applications with iBeacon.*
- [2] Stephen Statler, *Beacon Technologies.*
- [3] Frode Eika Sndnes, Yan Zhang, Chunming Rong, Laurence T. Yang, Jianhua Ma, *Ubiquitous Intelligence and Computing - 5th International Conference, UIC 2008, Spring.*
- [4] Ugur Bekcibasi, *Increasing RSSI Localization Accuracy with Distance Reference Anchor in Wireless Sensor Networks.*
- [5] Luca Pappalardo, *Localizzazione - Problema, Tecniche, Algoritmi,*
Reti mobili: Ad Hoc e di sensori,
Slide, 2011.
- [6] Cuccado, De Franceschi, Fauri, Sartor, *Analisi di algoritmi di autolocalizzazione per reti di sensori wireless,*
Tesi di laurea, 2011.
- [7] Paolo Sperandio, *Algoritmi di localizzazione per reti di sensori wireless*
Tesi di laurea, 2007.
- [8] Xuchen Yao, *An Introduction to the Kalman Filter.*
http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf
- [9] Ilenia Tinnirello, *Un Esempio di Applicazione del Filtro di Kalman alle reti WiFi.*
- [10] Angelo Nunzio La Bruna, *Applicazioni del filtro di Kalman su accelerometri.*
- [11] Philipp Jahoda, *MPAndroidChart* <https://github.com/PhilJay/MPAndroidChart>