RELAZIONE PROGETTO

Laboratorio Applicazioni Mobili - A.A. 2017/2018

Berti Matteo
767035

SCOPO DELL'APPLICAZIONE

Il progetto consiste in un'applicazione Android che comunica tramite bluetooth con un dispositivo Arduino dotato di due ruote e un sensore ad ultrasuoni. La comunicazione avviene in due sensi: dall'applicazione verso la piattaforma hardware e viceversa. Quindi sia lo smartphone che il dispositivo Arduino ricevono ed emettono segnali bluetooth.

FUNZIONALITÀ PREVISTE

È costituita da 3 funzionalità base, che compongono il cuore dell'applicazione, più altre 3 funzionalità che arricchiscono in modo significativo il progetto.

- **Controller**: comandare la direzione tramite l'app. L'applicazione Android invia al dispositivo Arduino essenzialmente 4 comandi: procedi *avanti*, torna *indietro*, svolta a *destra* e svolta a *sinistra*. L'utente può scegliere se comunicare con la piattaforma hardware attraverso un controller manuale, composto da 4 semplici bottoni, oppure tramite l'utilizzo dei sensori dello smartphone. In quest'ultimo caso i sensori di default (se disponibili) sono il sensore di *prossimità* (vicino torna indietro, lontano procedi avanti) e il sensore di *game rotation* (ruota verso destra lo smartphone per svoltare a destra, e verso sinistra per svoltare a sinistra).
- **Registrazione**: entrando in modalità registrazione i segnali emessi dal controller per telecomandare il dispositivo vengono registrati; una volta soddisfatti del proprio percorso, è possibile concludere la registrazione e salvarla. In futuro sarà possibile inoltre recuperarla e rieseguirla, in questo modo il dispositivo Arduino completerà il percorso salvato autonomamente senza che vengano emessi comandi dall'utente.
- **Ostacoli**: l'applicazione Android riceve in input dal modulo bluetooth la distanza in centimetri che c'è tra il dispositivo Arduino e un ostacolo posto di fronte a lui. Questa viene processata e mostrata sullo schermo all'utente. Se questa distanza diventa inferiore a 10 centimetri il testo si colora di rosso.

- **Sensori**: invece che fornire una, o comunque solo alcune, impostazioni predefinite per il controllo tramite sensori, è possibile far scegliere all'utente tutte le combinazioni di sensori che preferisce tra quelli supportati dal proprio smartphone. In questo modo ad ognuno dei 4 comandi base può associare un diverso sensore. Ad ogni sensore può associare un range di valori in cui il comando viene innescato e l'asse del sensore valido. Il tutto viene salvato tramite un sistema di preferenze. Esempio: si sceglie il sensore accelerometro per la svolta a destra, si imposta l'ascolto per l'asse X e si mette il range di validità 25%-50%, così facendo quando lo smartphone registra un cambiamento per il sensore accelerometro, per l'asse X, se il valore riportato è compreso tra il 25% e il 50% del range di valori che può assumere tale sensore, allora verrà inviato il comando di svolta a destra al dispositivo Arduino.
- **Calibrazione**: in aggiunta alla feature precedente. Ogni asse di ogni sensore può produrre valori estremamente diversi tra loro, e non sempre i valori indicati come massimo range combaciano con quelli reali. Per questo, e soprattutto, per migliorare l'esperienza utente si fornisce un sistema di calibrazione dei sensori. Il controller tramite sensori ha due stati: in pausa e in esecuzione, quando il controller è in pausa i valori prodotti dai sensori vengono registrati e viene salvato il valore minimo e il valore massimo che raggiungono. Da qui la percentuale del range del sensore che innescherà il comando varierà in base al minimo e massimo valore registrato durante questa fase. Esempio: un utente mette in pausa per calibrare il sensore di rotazione, ruota lo smartphone verso sinistra di 45° e di 45° verso destra, questo produrrà un valore minimo ed uno massimo, creando un range. Quando sarà in esecuzione, il comando partirà se i valori di tale sensore saranno validi sul range precedentemente impostato. Se l'utente lo ricalibra a 25° verso sinistra e 25° verso destra, il range cambierà e il comando sarà innescato quando lo smartphone è in una posizione diversa rispetto a prima. Anche qui si utilizzano preferenze.
- **Notificazione**: per aumentare l'attenzione in caso di ostacoli davanti al dispositivo Arduino è possibile inviare allo smartphone una notifica con un contatore che indica il numero di volte in cui è stato registrato un ostacolo a meno di 10cm di distanza. In questo modo se l'applicazione è attiva in background allerta l'utente anche se sta facendo altro. È presente infine un bottone di reset, che cancella la notifica, se presente, e azzera il contatore.

CARATTERISTICHE E REQUISITI

L'applicazione è formata da una prima Activity iniziale, in cui si mostrano i dispositivi bluetooth accoppiati e l'utente può scegliere quello a cui collegarsi. Successivamente si

entra nell'Activity Controller, che gestisce l'invio di notifiche, la connessione al database, la gestione del servizio bluetooth e un controller base del dispositivo tramite 4 bottoni. Qui è possibile spostarsi in Impostazioni, Registrazioni o Sensori. Il primo permette di cambiare i sensori associati ad ogni comando e i relativi valori. Il secondo di visualizzare i percorsi registrati e rieseguirli. Il terzo infine permette di controllare il dispositivo tramite i sensori dello smartphone (o di default, o quelli impostati). L'unico requisito fondamentale è che lo smartphone supporti il bluetooth.

PROGETTAZIONE E SCELTE IMPLEMENTATIVE

La versione di SDK minima per questa applicazione è la **19**, ovvero **Android 4.4 KitKat**. In questo modo è possibile far girare l'applicativo sul **90.1%** dei dispositivi Android in circolazione (fonte: Google).
I permessi richiesti dall'applicazione sono:
- **BLUETOOTH_ADMIN**: utilizzato per mostrare i dispositivi bluetooth accoppiati allo smartphone e collegarsi al dispositivo selezionato.
- **BLUETOOTH**: è collegato al permesso sopraccitato ed è necessario per la comunicazione bluetooth.
Sfortunatamente non è stato possibile utilizzare l'emulatore di Android Studio per testare l'applicazione, in quanto l'emulatore non supporta l'utilizzo del bluetooth e per questa app è assolutamente fondamentale. Tuttavia sono stati effettuati vari test su smartphone fisici, di seguito l'elenco:

Smartphone	Versione Android	Livello API	Risoluzione
Samsung Galaxy S2	4.0.4	16	480x800px
Samsung Galaxy S3	4.4.2	19	720x1280px
Samsung Galaxy J7	5.1	22	720x1280px
OnePlus 2	6.0.1	23	1080x1920px
Huawei P10 lite	7.0	24	1080x1920px

Per quello sottolineato si è presentato un problema riguardo il menu dell'action bar, non veniva mostrato per intero, ciò ha impedito l'accesso a parte delle funzioni. Per cui è stato necessario spostare l'SDK alla versione 19, tuttavia le funzionalità che sfruttano il

bluetooth e la notifica sono utilizzabili anche in versioni precedenti.
Il bluetooth è stato gestito attraverso un thread interno ad un **Bound Service**. Questa scelta è stata presa in quanto l'intento era di creare un servizio privato per l'applicazione eseguito nello stesso processo del client. Così facendo il servizio bluetooth è sempre attivo in background e le varie Activities accedono ai metodi messi a disposizione dal servizio, tramite il **Binder**.
Il flusso quindi sarà il seguente: l'utente invia un comando (da un'Activity che ha effettuato il binding) al servizio, il quale provvederà tramite la socket bluetooth ad inoltrarlo al dispositivo collegato, che agirà di conseguenza.
È necessario tuttavia che avvenga anche la comunicazione inversa, ovvero che il servizio comunichi con l'Activity in foreground. Per far ciò è stato utilizzato un sistema di **Broadcasting** gestito da un **LocalBroadcastManager**. Questo elemento fornisce molteplici vantaggi:
- I dati che si stanno trasmettendo non lasceranno mai l'applicazione, non verranno quindi persi dati privati.
- Non è possibile che altre applicazioni inviino tali broadcast all'app, ciò non genera buchi di sicurezza.
- Infine è più efficiente dell'invio di un broadcast globale attraverso il sistema.
Il flusso quindi sarà il seguente: Arduino invia tramite bluetooth un segnale, che viene catturato dalla socket bluetooth nel servizio, tramite un **Handler** il messaggio viene processato ed inviato in broadcast alle Activities che provvederanno a mostrarlo sullo schermo.
Tale sistema di broadcasting è utile in due casi:
- Per l'invio della distanza da un oggetto registrata dal dispositivo Arduino, alle Activities
- Per l'invio del segnale di disconnessione dal bluetooth
È stato infatti previsto che nel caso in cui la connessione bluetooth con il dispositivo cada, il servizio invia a tutte le Activity un segnale, esse provvederanno a terminarsi, lasciando solo l'Activity iniziale.
Per gestire questi due "canali" di comunicazione è stato utilizzato un **IntentFilter**.
I sensori vengono gestiti in modo tradizionale, nell'**onSensorChanged** si verifica se si è in fase di *calibrazione* o di *esecuzione*. Nel primo caso il valore viene registrato e aggiornato se supera il minimo o il massimo precedentemente salvato tramite un sistema di **Preferences**. Nel secondo caso, se tutti i parametri combaciano: asse e range di validità, viene inviato il comando ad Arduino. Inoltre se si è in stato di registrazione il comando viene anche salvato in una struttura temporanea, la quale verrà memorizzata in un database quando la registrazione termina.
Dei sensori salvati viene memorizzata la loro posizione all'interno della **lista dei sensori**

disponibili, fornita dal sistema. Viene salvata la posizione in quanto l'ordine di tale lista è sempre lo stesso, ed è improbabile che un sensore nel tempo non venga più riconosciuto, ciò rende questo metodo abbastanza sicuro.
Nell'Activity dedicata alla sensoristica è stata prestata molta attenzione alle fasi di **onPause** e **onStart**:
- Nell'onPause viene fatto l'unbind del servizio, stoppato il dispositivo Arduino ed effettuato l'**unregisterListener** dei sensori, ciò per evitare un consumo eccessivo di risorse da parte dell'applicazione.
- Nell'onStart si chiude un eventuale fase di esecuzione (tornando in modalità calibrazione) e si termina un eventuale registrazione. Inoltre in questa fase se i sensori sono cambiati rispetto a prima (modificando le impostazioni) vengono reimpostati i valori base e fatto l'unregisterListener dei sensori non più utilizzati.
Per la memorizzazione dei sensori scelti dall'utente, e i loro relativi parametri, come range di validità e asse, è stato utilizzato un sistema di **SharedPreferences** private. Questa scelta è stata fatta in quanto:
- Le impostazioni da salvare sono un un numero finito e tuttavia basso (20).
- I valori salvati sono brevi stringhe, interi e valori booleani, il modello chiave-valore è perfetto per lo scopo.
Precisamente per ognuno dei 4 comandi è stato salvato:

- **setting**: la posizione del sensore nella lista di sensori fornita dal sistema
- **range**: un intervallo percentuale sotto forma di stringa [es. "0.25:0.66"]
- **value**: un intero compreso tra 0 e 5
- **minmax**: l'intervallo di valori che vengono registrati durante la calibrazione
- **changedSensor**: un booleano che indica se il sensore è stato modificato

Infine se non vi sono valori già memorizzati vengono automaticamente salvati quelli di default.
La registrazione può avvenire sia all'interno nel controller manuale che del controller con i sensori. Essendo possibile registrare tutti i percorsi che si vuole ho scelto di utilizzare un database **SQLite**. Di seguito alcuni vantaggi:
- Permette di salvare dati strutturati [Nome, ListaDiComandi]
- Ha performance migliori che l'utilizzo del file system ad esempio
- Il database è visibile solo all'applicazione, ciò aumenta la sicurezza
- Può memorizzare ampie moli di dati
È stata implementata inoltre la possibilità di cancellare registrazioni precedentemente salvate. Per quanto riguarda l'esecuzione di un percorso memorizzato, è stato utilizzato un thread che esegue la serie di comandi, grazie a **sleep()** viene messo in pausa per permettere l'esecuzione dei comandi per il tempo registrato.

Si è cercato dove possibile di migliorare l'esperienza utente tramite una corretta gestione delle risorse. Nello specifico sono stati creati file di layout XML diversi per l'orientamento Landscape e Portrait rispettivamente nelle cartelle **res/layout-land** e **res/layout-port**. Tramite **onConfigurationChanged** è stato opportunamente gestito il cambio di configurazione, in quanto l'Activity viene ogni volta riavviata. Sempre parlando di risorse per ogni stringa presente all'interno dell'applicazione è stato utilizzata la referenza al file **res/values/strings.xml** che contiene la versione di default, in inglese. Tramite il file **res/values-it/strings.xml** è stata fornita anche una traduzione in italiano dell'applicazione.
Per completezza è stato aggiunto un **NotificationChannel** per garantire la compatibilità dell'utilizzo di notifiche anche a versioni di Android superiori alla 8.0. È infatti stato annunciato che dal livello API 26 tutte le notifiche devono essere assegnate ad un canale, quindi tramite un controllo nella versione di SDK viene o meno creato tale canale.
Per concludere, a livello concettuale la classe **Controller.java** è centrale, infatti la notificazione e l'accesso al database vengono effettuati qui. Se tale activity viene distrutta anche il servizio bluetooth viene chiuso. Altre activity che hanno bisogno di inviare notifiche, scrivere / leggere dal database o prendere valori statici (come i codici per i 4 comandi) fanno riferimento a lei. Questo evita di creare varie istanze ripetute quando non sono necessarie e fornisce concettualmente centralità a questa Activity.

DIFFICOLTÀ E SOLUZIONI

Le difficoltà principali sono state: permettere ad ogni activity di interagire in modo indipendente con l'interfaccia bluetooth, processare i dati in arrivo dal modulo e memorizzare i percorsi registrati. Il primo problema è stato risolto con un servizio eseguito in background, il secondo tramite un riassemblamento dei byte ricevuti all'interno dell'Handler e l'ultimo tramite query ad un database SQLite. Per quanto riguarda l'utilizzo dei sensori non è stato problematico la ricezione, ma piuttosto il sistema che processa un segnale in arrivo e calcola se è valido in base alle impostazioni scelte dall'utente.

POSSIBILI ESTENSIONI

Le possibili evoluzioni che mi vengono in mente possono essere un'estensione dei componenti nel dispositivo Arduino. In questo modo sarà possibile tramite bluetooth ricevere altri dati ed inviare comandi. Alcuni esempi di ricezione di dati aggiuntivi possono essere sensori come temperatura, umidità e luminosità. Esempi di controlli aggiuntivi possono essere un servo motore, delle ventole invece che delle ruote (tipo

drone). Un altro sviluppo interessante potrebbe essere utilizzare una connessione ad internet per controllarlo invece che il bluetooth, così che non sia necessario esservi vicino fisicamente.

CONCLUSIONE

Per concludere sono state utilizzare 6 principali tecnologie Android (Servizi, Bluetooth, Notifiche, Preferences, Broadcasting, SQLite) e un buon utilizzo del sistema di risorse (stringhe con traduzioni e layout in base all'orientamento). Un semplice progetto che integra elettronica di base con un livello medio di Android. In futuro probabilmente questo progetto verrà esteso ed integrato man mano che le mie conoscenze in entrambi gli ambiti aumenteranno.