R2Pi0

Tosatto Davide Riccardo Grespan

7 marzo 2017

Indice

1 Obiettivi del progetto

L'intera documentazione in formato LaTeXè disponibile nella repo `https://github.com/Tosa95/R2Pi0_Doc.git`.

L'obiettivo del progetto è di creare una riproduzione in scala di R2-D2, noto personaggio robotico della serie di Star Wars.

Figura 1: R2D2

La ricostruzione comprende:

1. movimento sul piano attraverso ruote;
2. riproduzione di suoni simili a quelli originali attraverso un buzzer, per dare quel tocco di retro che non guasta mai;
3. luci led e finto proiettore, comunque emulato da un led;
4. rilevamento degli ostacoli e corrispondente arrabbiatura del robot se viene mandato a tutta velocità verso uno di essi;
5. funzione follow che permette di mantenere una certa distanza dall'oggetto (o dalla persona) che lo precede
6. controllo vocale per funzioni base;
7. esoscheletro in cartone e plastica che riproduce le fattezze del robot;

8. un abbozzo di emozioni per dare un minimo di personalità al robot.

Questa prima versione non avrà la possibilità di ruotare la testa e cambiare inclinazione, questo per mancanza di tempo e di componenti (ulteriori motori, ulteriori driver, contatti girevoli per le luci posizionate sulla testa)

2 Componenti e scelte progettuali

2.1 Scheda di controllo

Iniziamo dal cuore del progetto: *la scheda di controllo.*

Le opzioni erano sostanzialmente due: Raspberry PiZero o Arduino Uno.

Figura 2: Raspberry PiZero

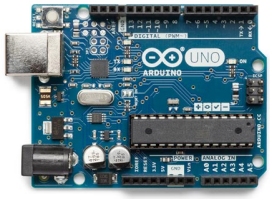

Figura 3: Arduino Uno

Per la maggior parte dei compiti le due schede erano sostanzialmente intercambiabili, in particolare:

1. *GPIO - General Purpose Input Output*: entrambi i dispositivi hanno tutte le interfacce che ci servono, ossia qualche I/O digitale e uscite PWM. Entrambi non possono dare in uscita elevate correnti, ma questo non importa perché il controllo di ogni componente è sempre mediato da transistor esterni;

2. *Alimentazione*: entrambi i dispositivi possono essere alimentati da usb, rendendo quindi possibile l'utilizzo di una batteria esterna per cellulare;

3. *Potenza di calcolo*: entrambi i dispositivi in esame hanno la potenza di calcolo necessaria per svolgere ognuno dei compiti necessari agli scopi del progetto.

Le seguenti motivazioni ci hanno fatto propendere per Raspberry PiZero:

1. *Semplicità di sviluppo*: grazie al fatto di avere a disposizione un sistema operativo completo, possiamo utilizzare linguaggi di più alto livello rispetto al C fornito da Arduino. Questo ci permette di creare codice meglio organizzato e più facilmente espandibile in futuro. La nostra scelta per il linguaggio è ricaduta su *Python*, in quanto semplice, conciso ed efficace;

2. *Memoria a disposizione*: grazie alla memoria disponibile, sensibilmente superiore a quella di Arduino, ci è permesso di integrare più suoni. Inoltre, in futuro, sarà sempre possibile migliorare la qualità degli stessi.

3. *Facilità di connessione*: Raspberry PiZero risulta più facile da connettere al mondo in quanto basta collegare un adattatore USB per WiFi o Bluetooth. Arduino invece richiede specifici shield,.

4. *Prezzo*: Raspberry PiZero costa 5$, molto meno di un Arduino, e offre infinite possibilità grazie ad un sistema operativo completo.

Un notevole problema del Raspberry PiZero e' il consumo.

Arduino senza carico consuma circa 50mA, mentre Raspberry PiZero arriva a 65mA in idle, si alza parecchio se il carico di lavoro aumenta, in quanto il processore di Raspberry PiZero è molto più performante di quello di Arduino, inoltre, montando un sistema operativo completo, Raspberry PiZero ha anche diverso overhead in campo di potenza computazionale e consumo, cosa non trascurabile.

Chiaramente i dispositivi di comunicazione comportano un innalzamento dei consumi in entrambe le schede.

Per ridurre al minimo i consumi abbiamo installato Raspbian in versione lite, ossia senza interfaccia grafica, e disattivato il controllore video HDMI, guadagnando così qualche prezioso mA.

In conclusione, i vantaggi nell'utilizzo di Raspberry PiZero ci sono sembrati schiaccianti.

2.2 Comunicazione

Le due opzioni per la comunicazione sono: Bluetooth o WiFi.

Abbiamo optato per una connessione Bluetooth per i seguenti motivi:

1. *Consumo*: il Bluetooth risulta avere un consumo energetico inferiore rispetto al WiFi, portandoci ad avere una durata della batteria leggermente superiore (non abbiamo dati precisi).

2. *Semplicità d'uso*: il Bluetooth è pensato per connessioni punto a punto, ossia esattamente la funzione che serve a noi. Il Wifi, invece, non è pensato per quello scopo e quindi relegherebbe all'utilizzo nei soli spazi dotati di rete WiFi oppure alla configurazione di una rete ad-hoc che risulta comunque un'operazione non banale e spesso mal funzionante (soprattutto su Linux).

NOTA: sebbene fosse sicuramente preferibile utilizzare un adattatore BLE, meno energivoro, per ragioni di componentistica disponibile abbiamo dovuto optare per un vecchio adattatore Bluetooth 2.0.
Inoltre, data la recente uscita del Raspberry PiZero W, con lo stesso hardware di Pi Zero, ma dotato anche di BLE e WiFi, al costo (teorico) di 10$, avremmo potuto usare tale dispositivo, ma per mancanza di disponibilità e costi di spedizione superiori al suo valore, abbiamo deciso di rimanere sul Pi Zero di prima generazione.

3 Schemi circuitali

Benchè Raspberry Pi sia spesso usato come microcontrollore, in realtà non è propriamente tale, si avvicina molto più ad un PC standard che ad una MCU.

Non è quindi in grado di erogare grandi correnti sui suoi pin di I/O. Non esiste un vero e proprio datasheet che dia dei valori precisi, ma nella community di sviluppatori Raspberry Pi si danno per accettate le seguenti soglie:

- circa 10-15mA per piedino

- circa 50mA di assorbimento complessivo derivante dalla somma dei contibuti di tutti i piedini
- i piedini a 5V sono collegati direttamente all'alimentazione, quindi da essi si può assorbire tutta la corrente erogabile dall'alimentatore meno quella assorbita dal Raspberry Pi stesso

Da questi valori si può capire come le uscite del Raspberry Pi siano state ideate per essere bufferate, e così noi abbiamo fatto.

3.1 Controllo motori

Per il controllo dei motori abbiamo deciso di utilizzare una versione leggermente modificata del ponte H. I nomi dei componenti sono presi da **??**. É necessario uno di questi circuiti per ogni motore installato.

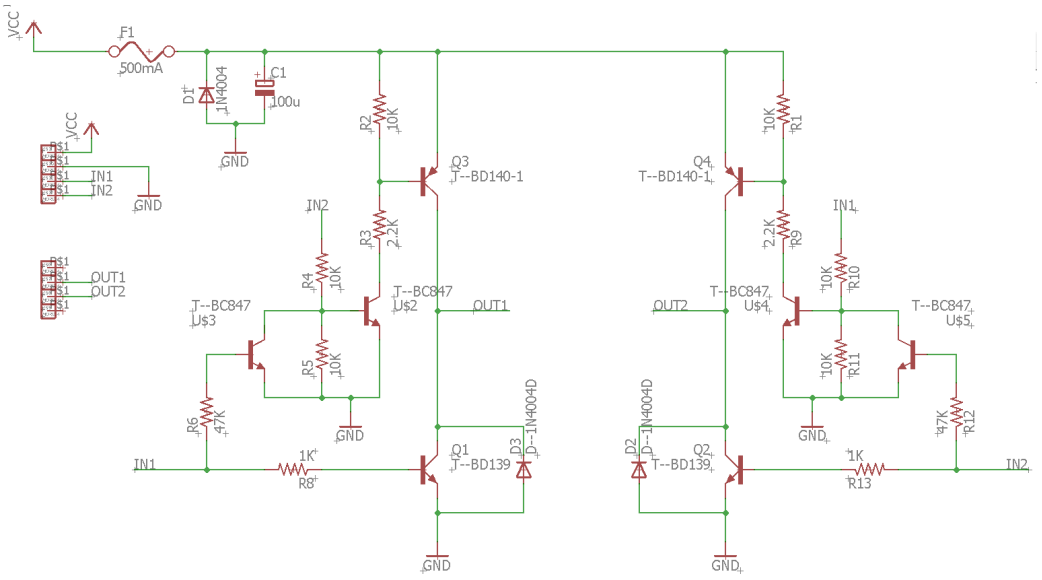

Figura 4: Schema del circuito di controllo dei motori

Passiamo alla rapida spiegazione del funzionamento del circuito. Il circuito ha due input: IN1 e IN2. Il comportamento è il seguente:

1. Se solo uno dei due fra IN1 e IN2 è attivo, nel motore scorre corrente, il verso dipende da quale dei due è alto.

2. Se sia IN1 che IN2 sono a 0, il motore resta completamente scollegato dal circuito, permettendogli di muoversi per inerzia.

3. Se sia IN1 che IN2 sono a 1, il motore ha entrambi i terminali a massa. Questo chiude la bobina del motore creando un cortocircuito. Così facendo se si tenta di muovere il motore si originerà all'interno della bobina una corrente che genererà a sua volta un campo magnetico che si opporrà al movimento del motore. Questo ha come risultato un effetto frenante.

Passiamo alla disamina dei componenti e delle loro funzioni. Tutti i transistori sono BJT, perché permettono di fare passare correnti elevate e soprattutto hanno basse cadute di potenziale in saturazione, ossia circa 0.3V tra collettore ed emettitore. Mantenere cadute di potenziale basse è essenziale perché il tutto è alimentato a soli 5V.

I quattro transistor principali di potenza sono Q1, Q2, Q3 e Q4. La circuiteria è fatta in modo tale che né Q1 e Q3, né Q2 e Q4 conducano *mai* insieme. Questo avvenimento infatti produrrebbe un cortocircuito che danneggerebbe l'hardware.

Da misurazioni, si è visto che il consumo di ogni motore è inferiore ai 200mA, si è dunque pensato di inserire per sicurezza un fusibile da 500mA, F1, sufficientemente permissivo da far funzionare il motore, ma abbastanza restrittivo per evitare di bruciare l'alimentatore in caso di corto.

Abbiamo poi altri quattro transistor che servono ad assicurare l'assenza di corti e ad implementare la funzionalità di cortocircuitare la bobina con entrambi gli ingressi a 1.
La funzione logica realizzata è quella mostrata in tabella 1

IN1	IN2	Q1	Q2	Q3	Q4
off	off	off	off	off	off
off	on	off	on	on	off
on	off	on	off	off	on
on	on	on	on	off	off

Tabella 1: Funzione logica implementata nel circuito

Da notare che U$2 e U$4 hanno un partitore resistivo sulla base. Questo serve a ritardare la loro accensione, infatti gli input non sono realmente squadrati ma aumentano gradualmente con una certa pendenza, seppur elevata.

Questo permette di ritardare l'attivazione dei transistor sopracitati. Infatti se sulla base servono 0.7V per permettere l'accensione, con il partitore è richiesto che IN2 sia già ad almeno 1.4V. Questo accorgimento serve quando attiviamo contemporaneamente IN1 e IN2, per dare il tempo a U$3 e U$1 di inibire l'accensione di Q3 e Q4 impedendo quindi cortocircuiti.

3.2 Buzzer

Abbiamo usato un buzzer piezoelettrico recuperato da un vecchio telefono.

Dal datasheet del buzzer emerge la seguente figura:

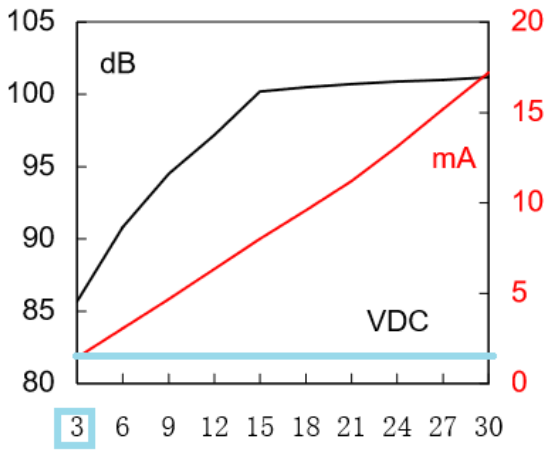

Figura 5: Caratteristica tensione-corrente del buzzer

Considerando che lavoriamo attorno ai 3V, dal grafico possiamo notare come, a tale tensione, la corrente assorbita sia di circa 2mA.
Possiamo quindi lasciare che il buzzer sia pilotato direttamente dal Raspberry Pi in quanto, come detto ad inizio sezione, ogni piedino può erogare fino a 10mA, quindi siamo ampiamente entro la soglia.

4 Software

4.1 Lato Raspberry Pi

Tutto il codice sorgente lato Raspberry Pi è liberamente accessibile attraverso il seguente repository github: `https://github.com/vxrich/Python_`

`R2Pi0.git`.

Il codice è commentato, ma solo nei punti in cui abbiamo ritenuto necessario precisare qualcosa. Per le funzioni o le classi autoesplicative abbiamo evitato di aggiungere commenti, in modo da non appesantire la lettura del codice.

Si è deciso di dare al software un'impronta un po' più professionale rispetto al semplice script dimostrativo autocontenuto.
Abbiamo quindi suddiviso il codice in diversi moduli e usato alcuni design patterns per rendere il software il più possibile espandibile in futuro. Ovviamente resta un progetto di piccole dimensioni, senza particolari complessità.

Vediamo una rapida disamina dei moduli software:

1. *Controllo del movimento*: questo è un macro modulo composto da diversi sottomoduli. Abbiamo deciso di realizzare il controllo dei motori attraverso un decorator pattern: abbiamo uno stack liberamente componibile di controllori che via via aggiungono funzionalità. La classe base *MotionController* astrae il controllo vedendolo come una semplice combinazione di una velocità ed una rotazione. Aggiunge inoltre un *watchdog* per essere certi che, in assenza di comandi per un certo periodo di tempo, il robot venga fermato. Abbiamo poi il layer *DistanceAdapter* che aggiunge la funzionalità di blocco in caso di ostacoli e il layer *MoodedAdapter* che aggiunge alcuni semplici comportamenti "emotivi" del robot.
   Ci sono poi i sottomoduli *Follower* e *Reacher* che si occupano rispettivamente di mantenere una certa distanza dall'ostacolo di fronte e di raggiungerlo (attivati mediante comando vocale dall'app Android).

2. *Server*: composto essenzialmente da due classi: *Server* e *Subserver*. Server rimane in ascolto sul bluetooth e per ogni nuova connessione avvia una nuova istanza di Subserver che si occuperà di soddisfare le richieste del client.
   Ricevono in ingresso un array associativo contenente coppie comando-callback, in modo da poter aggiungere e togliere comandi nel modo più semplice possibile.

3. *Suono*: Si occupa di gestire la riproduzione di suoni monofonici in formato RTTTL. Composto da due classi: *RTTLConverter*, che converte i file RTTTL in un formato più usabile in runtime ed *RTTLPlayer*, che

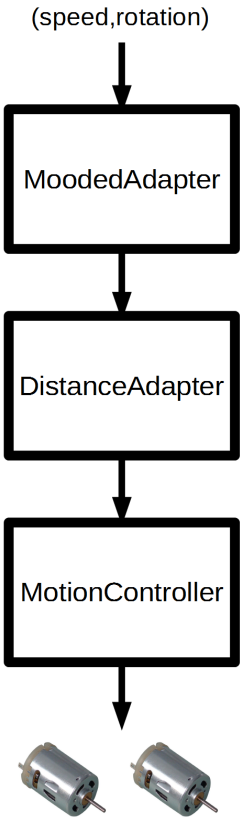

Figura 6: Stack controllo movimento usato nella versione attuale

prende in input i suoni convertiti e li esegue in sequenza (ha una coda di esecuzione).

4. *Sensore di prossimità*: questo modulo è stato il più critico, perchè, per leggere la distanza, è necessario misurare la durata di un impulso su un pin di I/O. Scritto in python, questo risultava troppo CPU intensive, in quanto dovevamo di fatto eseguire un polling sul piedino senza poter mettere in pausa il thread per non avere enormi perdite di precisione. La soluzione è stata di ricorrere alla libreria *wiringPi*, che, nella sua declinazione per il linguaggio C, permette di utilizzare degli interrupt scatenati sui fronti di salita e/o discesa dei pin di I/O.
Abbiamo quindi scritto questo particolare modulo in C, poi un wrapper

per poterlo compilare come modulo Python. Un po' una faticaccia ma il risultato è ottimo!
Il modulo C si chiama *proxsensor*, mentre in Python lo abbiamo incapsulato nella classe *SensorController* (del modulo UltrasonicSensor.py) che legge la distanza ad intervalli regolari, scatenando un evento con la nuova distanza a cui si possono collegare tutti i listener del caso (pattern Observer).

5. *Mood*: piccolo modulo che ha il compito di gestire qualche emozione del robot. Niente di che, andrebbe ampliato.

6. *Pinout*: questo modulo contiene *tutte* le associazioni pin-funzione svolta necessarie per il progetto. Questo ci permette di rimappare tutti gli I/O velocemente attraverso un singolo file.

7. *Blue*: alcune funzioni di utilità per utilizzare la connessione bluetooth.

8. *Misc*: cartella contente note ed appunti su come preparare un Raspberry Pi per fare funzionare questa applicazione.

9. *r2pi0_V2*: punto di ingresso del sistema. É lo script da lanciare (con permessi di amministratore) per fare funzionare il tutto.

   Quasi tutti i moduli sono connessi attraverso eventi e rispettivi callback per ridurre al minimo possibile l'interdipendenza tra gli stessi.

4.2 Lato Smartphone

Tutto il codice sorgente lato Smartphone è liberamente accessibile attraverso il seguente repository github: `https://github.com/vxrich/R2-Pi0.git`.

Il codice è commentato, ma solo nei punti in cui abbiamo ritenuto necessario precisare qualcosa. Per le funzioni o le classi autoesplicative abbiamo evitato di aggiungere commenti, in modo da non appesantire la lettura del codice.

La nostra riproduzione di R2-D2 necessita di un controller remoto per i movimenti e le interazioni base, per questo abbiamo deciso di utilizzare un'applicazione programmata ad hoc per smartphone.

L'applicazione e' scritta in *Java* per sistema operativo *Android*.

Partendo dalla parte visiva, l'applicazione e' costituita da elementi grafici per l'interazione con il robot, che permettono:

- il controllo del movimento tramite un joystick virtuale
- l'attivazione della connessione bluetooth con la macchina
- l'invio di una richiesta per la riproduzione di suoni
- l'attivazione dei comandi vocali
- lo spegnimento completo del robot

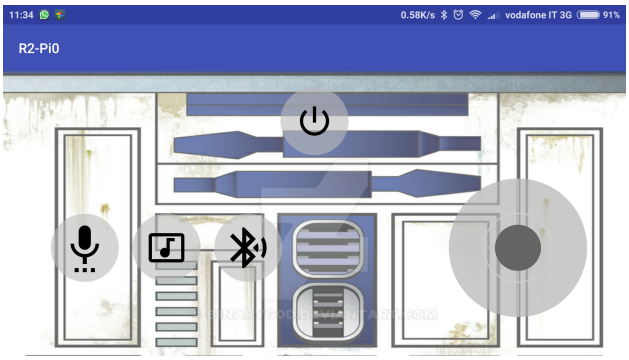

Figura 7: Screen dell'applicazione

4.2.1 Classi

Le classi che compongono l'applicazione e che permettono tutte le interazioni sono:

- MainActivity
- BluetoothComunication
- JoyStickView
- JoyStickTranslator
- JoyStickTrigonometricTranslator
- VocalTranslator

La classe *MainActivity* rappresenta l'activity principale dell'applicazione, l'unica schermata con la quale interagiremo. Il suo compito è quello di instanziare i pulsanti e il joystick, che permettono il movimento e l'utilizzo delle funzioni del robot. In questa classe creaiamo un oggetto *BluetoothConnection* che ci permettera' l'invio dei vari comandi legati ad ogni pulsante.

La gestione dei comandi vocali avviene direttamente nella MainActivity, fatto che ci permette di non dover creare una nuova activity per la gestione di questa funzione. Viene quindi creata una funzione startSpeechToText che avvia un Intent che inzia a catturare l'audio dal microfono per poi convertirlo tramite i servizi google in testo e salvarlo in una variabile dal metodo onActivityResult, da tale messaggio verra' poi estratto il comando tramite l'oggetto translatore della classe *VocalTranslator* e inviato tramite Bluetooth al robot.

La classe *BluetoothComunication* utilizza le classi android per la gestione dei moduli e connessioni bluetooth, in particolare vengono utilizzate le classi BluetoothAdapter, per creare un oggetto adattatore che ci consentirà di connetterci al robot, BluetoothDevice, che ci servirà per elencare tutti i dispositivi ai quali il nostro smartphone è associato, e BluetoothSocket, che sarà necessario per instaurare la connesione.
Affinchè ci possa essere la connesione è necessario effettuare l'associazione tra la parte di programma Python residente su Raspberry Pi e la parte Java su Android manualmente, da riga di comando, successivamente il Raspberry Pi verrà visualizzato nell'elenco dei dispositivi associati allo smartphone e

basterà selezionarlo nella finestra che compare premendo il pulsante per la connessione Bluetooth. La classe inoltre prevede la gestione degli errori dovuti all'utilizzo su dispositivi senza Bluetooth e le eventuali perdite della connessione Bluetooth tra smartphone e Raspberry Pi. Sono poi implementate tutte le funzioni per inviare i comandi al robot.

La classe *VocalTranslator* ha il compito di ricercare, nelle frasi ritornate dal servizio di riconoscimento vocale, le parole chiave che ci permettono di inviare i comandi al robot. I comandi implementati sono:

- seguimi
- rotazione destra e sinistra
- interruzione della musica

Possono essere interpretati sia in lingua italiana che inglese e in diverse forme.

Nel caso dell'inserimento di un comando non riconosciuto il robot emetterà un segnale acustico di errore.

La classe *JoyStickView* permette di disegnare l'elemento JoyStick sulla shermata della nostra applicazione e permettere di ottenere la resistuzione di valori al movimento del cursore.

L'interfaccia *JoyStickTranslator* permette di stabilire i metodi che verranno utilizzati per la restituzione della velocita' e della rotazione.

La classe *JoyStickTrigonometricTranslator* implementa i metodi dell'interfaccia JoyStickTranslator in modo che vengano resituiti dei valori corretti e normalizzati per il successivo invio via Bluetooth al robot.

5 Sviluppi futuri

Abbiamo pensato ad alcuni interessanti sviluppi futuri:

- *Aggancio bersaglio ed inseguimento*: l'idea è di dotare il robot di videocamera e, attraverso Open CV o librerie simili, permettergli di agganciare e seguire un oggetto, in particolare pensavamo ad un simbolo da incollare sui nostri vestiti per farci seguire. Ovviamente questo presuppone di passare ad un modello di Raspberry Pi più performante, come

ad esempio il Raspberry Pi 3, in quanto la computer vision richiede una potenza di calcolo elevata.

- *Puntamento automatico*: dotare il robot di una pistola ad acqua con cui possa "fare il dispettoso" sparando getti d'acqua contro le persone. Ovviamente sempre attraverso Open CV, ma questa volta puntando alle facce.