

SUPSI

Real Time Object Identification - Computer Vision

Studente/i

Claudio Bonesana

Relatore

Dipl. Ing. Giacomo Poretti

Correlatore

Marco Bulani

Committente

Stouch SA

Corso di laurea

C08865

Modulo

Lavoro di Diploma

Anno

2013

Data

2 settembre 2013

STUDENTSUPSI

Indice

Abstract	1
1 Introduzione	3
1.1 Il commitente	3
1.2 L'idea del dispositivo	3
2 Obiettivi	5
3 Stato dell'arte	7
3.1 Raspberry Pi modello B	8
3.2 Beaglebone Black	9
3.3 Android PC - Rock	10
3.4 Mini-ITX computer	11
3.5 Scelta della piattaforma hardware	12
4 Sviluppo	13
4.1 Target scelto	13
4.1.1 Fotocamera	14
4.1.2 RFID - Radio Frequency IDentification	15
4.1.3 Connessione WiFi	16
4.1.4 Bluetooth	17
4.1.5 Periferiche aggiuntive	17
4.1.5.1 Bottoni	17
4.1.5.2 Speaker e LED	18
4.1.5.3 Alimentazione e Batteria	18
4.1.6 Hardware implementato	19
4.2 Ambiente target e ambiente di sviluppo	21
4.2.1 Sistema operativo target	21
4.2.2 Linguaggio e librerie di sviluppo	22
4.2.3 Ambiente di sviluppo	23
4.3 Software sviluppato	24

4.3.1	Librerie RFID	24
4.3.1.1	Protocollo easy2read	25
4.3.1.2	Comunicazione	26
4.3.1.3	Esempio di comandi	26
4.3.1.4	Librerie implementate	28
4.3.1.5	Pubblicazione	30
4.3.2	Demone di sistema	30
4.3.2.1	Ciclo di vita	30
4.3.2.2	Ciclo principale del demone	30
4.3.3	Software applicativo	33
4.3.3.1	Idea di base	34
4.3.3.2	Interazione tra i moduli	35
4.3.3.3	Sistema di log	35
4.3.3.4	Ciclo di vita	36
4.3.3.5	Invio di immagini al server	36
4.3.3.6	Modalità online/offline	36
4.3.3.7	Comunicazione con il demone	37
4.3.3.8	Variabili globali e configurazione	37
4.3.3.9	Software esterni al dispositivo	38
4.3.4	OpenCV: programma cropper	38
5	Test e risultati	43
5.1	Test compiuti	43
5.1.1	Test sull'hardware	43
5.2	Funzionalità	46
5.2.1	Test funzionali	46
5.2.2	Test sul campo	47
5.2.2.1	Connessione alla rete Wireless di Lugano Cornaredo	47
5.2.2.2	Connessione alla rete Wireless di WiFi Lugano	47
5.2.3	Risposta dell'utente al dispositivo	49
5.3	Difetti e limitazioni	49
5.3.1	Parte software	49
5.3.1.1	Passaggio in modalità online	49
5.3.1.2	Rallentamenti e crash di sistema	49
5.3.1.3	Gestione di rete	51
5.3.2	Parte hardware	51
5.3.2.1	Bluetooth	51
5.3.2.2	Interferenze	51
5.3.2.3	Alimentazione	52

6 Conclusioni	53
6.1 Raggiungimento degli obiettivi	53
6.2 Sviluppi futuri	54
6.2.1 Comunicazione bidirezionale con server	54
6.2.2 Aggiornamenti via server	54
6.2.3 Bluetooth	54
6.2.4 Comunicazioni di rete	55
6.2.4.1 Voce	55
6.2.5 Hardware e LED	55
6.2.6 OpenCV	55
6.2.7 Interfaccia grafica	56
6.3 Passo di ingegnerizzazione per il prototipo 2	56
6.4 Conclusioni personali	56
Elenco degli allegati	59
Appendice A Configurazione di un cross-compiler per il Raspberry Pi	61
A.1 Definizioni	61
A.2 Preparazione	62
A.3 Installazione del cross-compiler	62
A.4 Verifica del funzionamento corretto dei compilatori	63
A.4.1 Compilare il linguaggio C	63
A.4.2 Compilare il linguaggio C++	63
A.4.3 Deployment	64
Appendice B Compilare Qt5 per Raspberry Pi	65
B.1 Preparazione	65
B.1.1 Directory di lavoro	65
B.1.2 Immagine di sistema locale	65
B.2 Installazione di Qt5 per Raspberry Pi	66
B.2.1 Compilazione di Qt base	67
B.2.2 Compilare e installare i moduli aggiuntivi	67
B.3 Completamento dell'installazione	68
Appendice C Configurare Qt Creator per l'utilizzo con Qt5 su Raspberry Pi	69
C.1 Preparazione	69
C.2 Configurazione di QtCreator	69
C.2.1 Installazione delle librerie locali per il sistema	69
C.2.2 Impostazione della Tool Chain	69
C.2.3 Impostazione delle librerie Qt	70

C.2.4 Configurazione del dispositivo Linux	70
C.2.5 Configurazione dei Kit di sviluppo	70
C.2.6 Creazione e configurazione di un progetto	71
Appendice D OpenCV per Raspberry Pi	73
D.1 Preparazione delle librerie	73
D.2 Cross-compilazione delle librerie	73
D.3 Configurazione del progetto in QtCreator	75
Appendice E WiringPi per Raspberry Pi	77
E.1 Installazione	77
E.2 Sistemazione dei link	77
E.3 Configurazione di un progetto in QtCreator	78
E.4 Mappatura dei GPIO	78
Appendice F Struttura della directory dell'applicazione	81
Appendice G Raccolta di comandi utili	83
G.1 Connessione remota	83
G.2 Installazione	83
G.3 Avviare e fermare il demone	84
Allegati	89

Elenco delle figure

2.1	Diagramma di lavoro.	6
3.1	Il Raspberry Pi modello B [1].	8
3.2	La BeagleBone Black [2].	9
3.3	La scheda Android PC [3].	10
3.4	Una scheda Mini-ITX PC [4].	11
4.1	Schema a blocchi dei componenti hardware del dispositivo.	14
4.2	Camera per RaspberryPi [1].	14
4.3	Modulo RFID prodotta da CAEN SpA.	15
4.4	Antenna WiFi.	16
4.5	Dongle Bluetooth.	17
4.6	Bottoni utilizzati.	18
4.7	Schema a blocchi dei componenti hardware utilizzati per il prototipo.	19
4.8	L'interno del dispositivo con i componenti collegati.	20
4.9	Il logo di Raspbian.	22
4.10	Kit di sviluppo RFID CAEN SpA. in dotazione per il progetto.	24
4.11	Sequenza di avvio del demone.	31
4.12	Ciclo principale del demone.	32
4.13	Diagramma di sequenza per un braccialetto trovato.	39
4.14	Diagramma di sequenza per un braccialetto non trovato.	40
4.15	Diagramma di sequenza del passaggio a modalità offline.	41
4.16	Diagramma di sequenza del ritorno in modalità online.	42
5.1	Il dispositivo in posizione nella zona Park & Read del Parco Ciani a Lugano.	48
5.2	Utilizzo del dispositivo da parte di un utente.	50
E.1	Come i pin sono segnati nella libreria WiringPi.	79

Elenco delle tabelle

3.1 Scheda tecnica di un Raspberry Pi [1]	8
3.2 Scheda tecnica della Beaglebone Black [2]	9
3.3 Scheda tecnica di un APC Rock [3]	10
3.4 Scheda tecnica di una scheda madre MiniITX [4]	11
4.1 Parametri utilizzati per la comunicazione sulla porta seriale.	29
5.1 Test eseguiti sull'hardware.	43
5.1 Test eseguiti sull'hardware.	44
5.1 Test eseguiti sull'hardware.	45
5.1 Test eseguiti sull'hardware.	46

Elenco dei listati

4.1	Struttura del header di un pacchetto easy2read.	25
4.2	Struttura di un frame di un pacchetto easy2read.	26
4.3	Coppia AVP per il nome del comando.	27
4.4	Coppia AVP per un parametro relativo al comando.	27
4.5	Messaggio inviato dal host al lettore.	27
4.6	Messaggio in risposta proveniente dal lettore.	28
4.7	File di configurazione per l'applicativo (stouch.conf).	37
4.8	File di configurazione per il comparto suoni (sound.conf).	38

Abstract

Il marketing online è uno strumento efficace a livello di promozione turistica. Questo progetto prevede la progettazione e la realizzazione di un prototipo per sistema autonomo che permetta ad utenti, identificati da un braccialetto dotato di tag RFID, di scattare fotografie in luoghi strategici dal profilo turistico, a pubblicarle e condividerle attraverso i nuovi canali di comunicazione interattivi quali smartphone, tablet, blog e social networks. Lo scopo del committente è quello di sfruttare le nuove tecnologie nel marketing turistico.

Dal profilo tecnico il sistema realizzato è incentrato su un Raspberry Pi, con core ARM11, collegato con periferiche interattive e multimediali. Il sistema software scelto è basato su un sistema Linux derivato da Debian con librerie Qt e OpenCV per la manipolazione delle immagini. Valore aggiunto è l'utilizzo di un sistema di cross-compilazione e debugging remoto dell'applicativo software.

Il progetto ha portato alla realizzazione di un prototipo funzionante che è stato testato sul campo.

Nowadays, online marketing is a powerful tool for tourists advertising around the world. This project focuses on the design and the implementation of a prototype of an autonomous system able to interact with registered users, identified by RFID bracelet, taking pictures of themselves into awesome touristic places . Pictures are then shared through new media as smartphone, tablets, blogs and social networks. The aim of the customer is to exploit this technology in touristic marketing and promotion.

From the technical point of view, the system is based on a Raspberry Pi (core ARM11) connected with several interactive and multimedia devices. The software chosen to run on this system is a Debian-like Linux distribution with Qt libraries and OpenCV toolkit to handle pictures. An additional peculiarity of the project is the usage of a cross-compiler,a remote debugging and development system for the software application.

The project has produced a working prototype tested in real conditions.

Capitolo 1

Introduzione

1.1 Il commitente

Il richiedente principale di questo progetto è la società Stouch SA, iscritta al registro di commercio Svizzero e basata a Lugano (CH-501.3.017.954-0).

Al progetto partecipa come partner commerciale la Città di Lugano (Dicastero Turismo e Eventi) che con Stouch SA ha firmato una lettera di intenti relativa al proseguimento di un primo progetto pilota realizzato nel corso del 2013 e con l'obiettivo di estendere sul proprio territorio l'installazione degli apparecchi prodotti e commercializzati dalla società Stouch SA.

1.2 L'idea del dispositivo

Il marketing online diventa uno strumento sempre più importante e strategico a livello di promozione turistica. Con questo strumento la città si dota di una tecnologia innovativa dal potenziale unico e molto interessante.

Trattandosi di un progetto imprenditoriale con importanti prospettive, per la Città di Lugano si tratta di una vetrina importante che sarà portata come esempio e applicazione reale del prodotto sia a livello nazionale che a livello internazionale.

Il progetto prevede la produzione e installazione di apparati per il promozione turistico basati su una tecnologia che invita il visitatore (dotato di appositi braccialetti identificativi) a scattare fotografie, quindi recapitarle e condividerle tramite telefono, posta elettronica, social networks e blogs in modo completamente automatico, arricchite con informazioni aggiuntive e propagandistiche e sfruttare nuovi modelli di social marketing e marketing virale.

Il progetto è inteso a fianco di un mandato diretto della ditta alla SUPSI che coinvolge sia l'istituto ISIN per l'architettura del SW che il DACD per la produzione fisica dell'oggetto tramite stampate 3D.

Capitolo 2

Obiettivi

L'obiettivo principale di questo lavoro consiste nel creare l'architettura software necessaria al funzionamento dell'idea implementandola su hardware a basso costo partendo dalle richieste del committente e dall'analisi iniziale fatta dal DACD.

L'architettura software dovrà essere in grado di:

- funzionare 24 ore su 24, 7 giorni su 7;
- acquisire immagini da una fotocamera;
- interagire con un dispositivo RFID;
- scambiare dati con un server remoto;
- interagire con una serie di pulsanti;
- essere completamente autonoma senza avere la possibilità di avere uno schermo o comunicazioni attraverso periferiche non precedentemente citate.

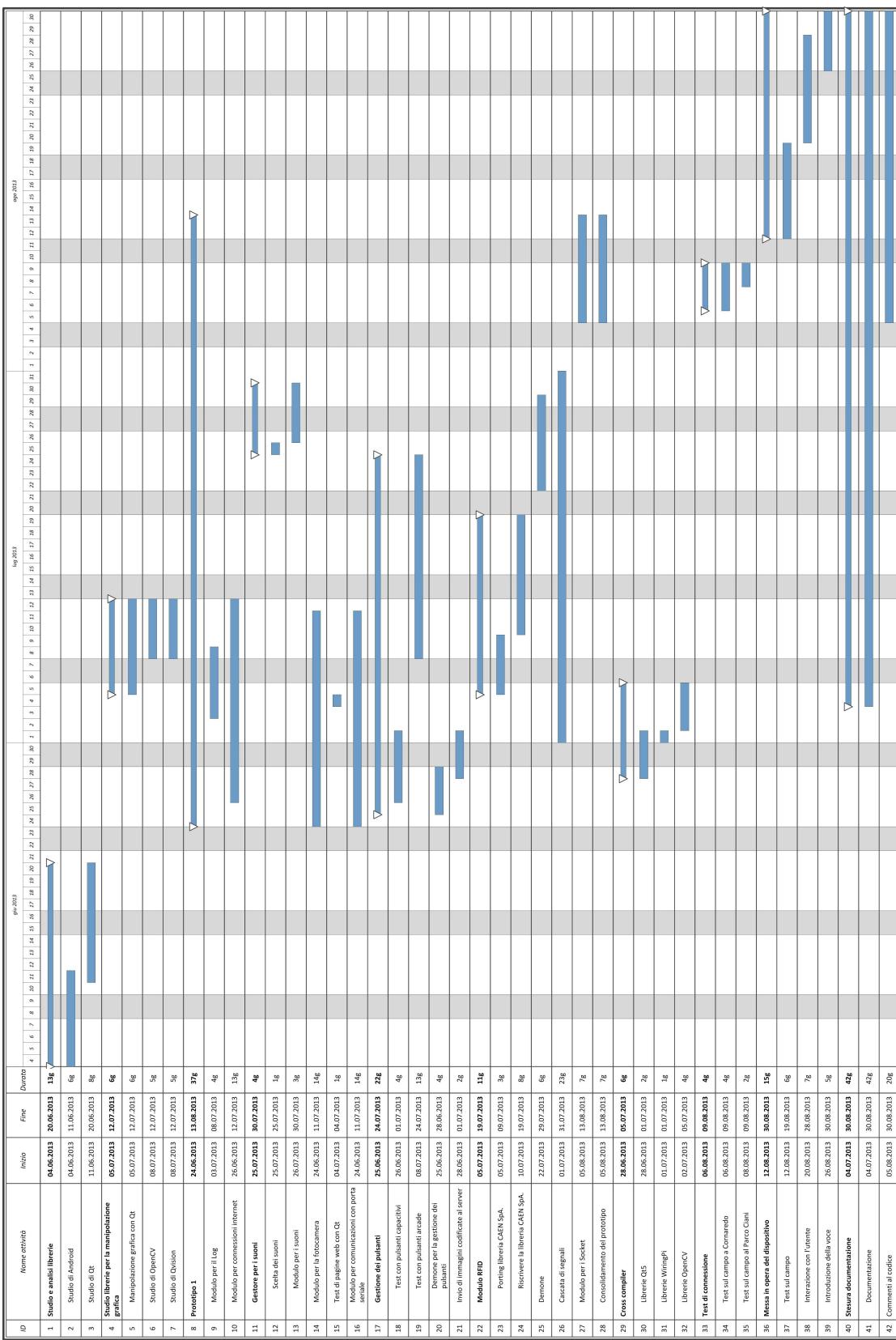


Figura 2.1: Diagramma di lavoro.

Capitolo 3

Stato dell'arte

I requisiti del sistema Stouch necessitano una configurazione hardware con una potenza di calcolo adatta a gestire in modo affidabile diverse funzionalità, come ad esempio i protocolli di comunicazione o l'acquisizione e l'elaborazione di immagini da una fotocamera. Nel contempo il sistema deve essere di dimensioni contenute e in grado di funzionare in luoghi in cui l'uso di un normale computer risulterebbe inadatto. Essenziale è anche la possibilità di interazione attraverso porte di I/O multifunzionali e protocolli di comunicazione quali UART, I2C e SPI non sempre disponibili nei sistemi convenzionali.

Trattandosi di un dispositivo fisso, la minimizzazione del consumo energetico non è un obiettivo primario dello sviluppo, sebbene vada comunque tenuto in considerazione.

Con queste prerogative, si è quindi scelto di puntare su di un dispositivo di piccole dimensioni, ma abbastanza potente da poter supportare un sistema operativo per poter gestire la cattura e manipolazioni di immagini e i necessari servizi di comunicazione via USB e rete. Inoltre, in previsione di una produzione in serie del sistema, si è cercato di contenere i costi del prototipo scegliendo il più possibile componenti standard.

Grazie all'enorme spinta data negli ultimi anni dall'evoluzione di smartphone e tablet e dallo sviluppo delle architetture ARM v6 e v7, i cui capostipite sono stati gli ARM11 e la famiglia Cortex, il mercato offre attualmente molte soluzioni su schede dalle dimensioni ridotte e ad un prezzo contenuto. Punto di forza di queste soluzioni risiede nella capacità di supportare appieno un sistema operativo completo, spesso basato su Linux, ed essere molto versatili, facilmente programmabili e adattabili agli scopi più disparati.

In una fase di ricerca iniziale si sono prese in considerazione quattro tipologie di microprocessori, contemplando soluzioni che comprendono sia le nuove generazioni ARM che le più tradizionali basate su Intel x64. In questo capitolo verranno esposti a titolo di confronto alcune soluzioni che sono state prese in considerazione nella fase preliminare del progetto.

3.1 Raspberry Pi modello B

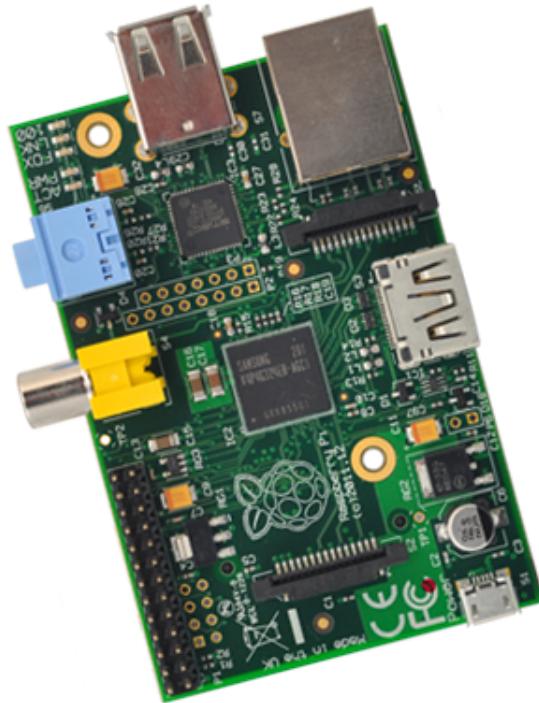


Figura 3.1: Il Raspberry Pi modello B [1].

Tabella 3.1: Scheda tecnica di un Raspberry Pi [1].

Core	Broadcom BCM2835 SoC full HD multimedia applications processor
CPU	700 MHz Low Power ARM1176JZ-F applications Processor
GPU	Dual Core VideoCore IV®Multimedia Co-Processor
Memoria	512MB SDRAM
Ethernet	10/100 Ethernet RJ45
USB 2.0	doppio connettore
Video Output	HDMI (rev 1.3 & 1.4) RCA video composito (PAL e NTSC)
Audio Output	3.5mm jack, HDMI
Archiviazione	SD, MMC, SDIO card slot
Sistema operativo	Linux, Android ¹
Dimensioni	8.6cm x 5.4cm x 1.7cm
Costo	55 Fr.

3.2 Beaglebone Black

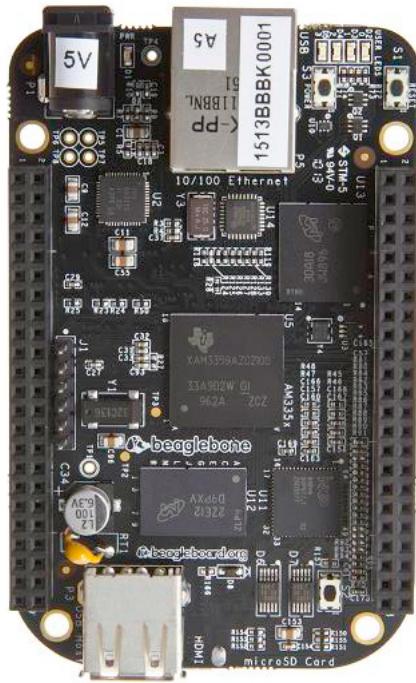


Figura 3.2: La BeagleBone Black [2].

Tabella 3.2: Scheda tecnica della Beaglebone Black [2].

Core	TI Sitara AM3359AZCZ100
CPU	AM335x 1GHz ARM® Cortex-A8
GPU	3D graphics accelerator
Memoria	512MB DDR3 RAM
Ethernet	10/100 Ethernet RJ45
USB 2.0	connettore singolo
Video Output	mini HDMI
Audio Output	mini HDMI
Archiviazione	2GB 8-bit eMMC on-board flash storage, SD
Sistema operativo	Ångström Linux, Android, Ubuntu
Dimensioni	8.64cm x 5.33cm
Costo	57.40 Fr.

3.3 Android PC - Rock

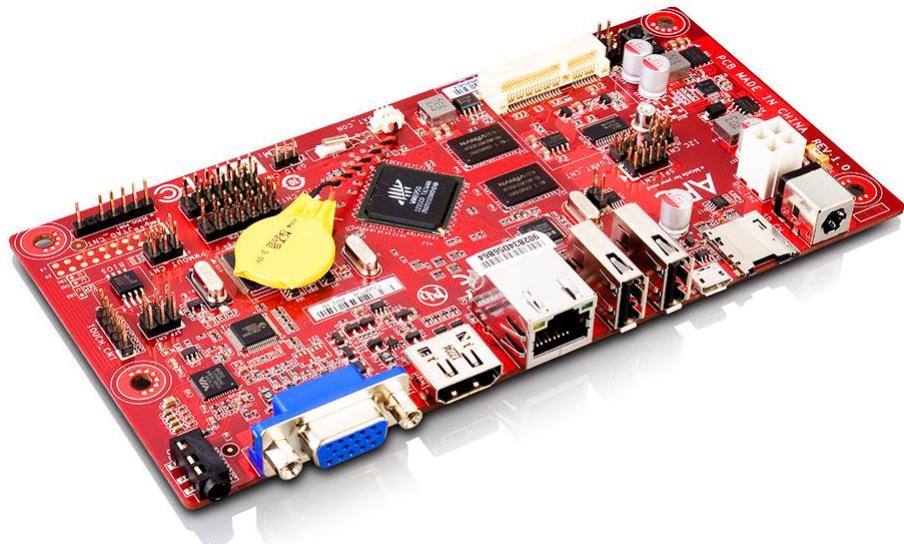


Figura 3.3: La scheda Android PC [3].

Tabella 3.3: Scheda tecnica di un APC Rock [3].

Core	VIA WonderMedia WM8950
CPU	VIA ARM Cortex-A9 @800Mhz
GPU	Built-in 2D/3D risoluzione massima 1080p
Memoria	512MB DDR3 RAM, 4GB NAND Flash
Ethernet	10/100 Ethernet RJ45
USB 2.0	doppio connettore
Video Output	HDMI, VGA
Audio Output	3.5mm jack
Archiviazione	microSD
Sistema operativo	Android 4.0 (PC System)
Dimensioni	17.0cm x 8.5cm (Neo-ITX Standard)
Costo	ca. 100 Fr.

3.4 Mini-ITX computer



Figura 3.4: Una scheda Mini-ITX PC [4].

Tabella 3.4: Scheda tecnica di una scheda madre MiniITX [4].

Chipset	ASRock B85M-ITX Mini-ITX
CPU	Intel LGA1150 compatible 4 th gen Haswell (non incluso)
GPU	Integrato nel processore o PCI Express 16x (non incluso)
Memoria	max 8GB DDR3 1600 DIMM
Ethernet	Gigabit Ethernet RJ45
USB 2.0	2 x USB 2.0, 2 x USB 3.0
Video Output	HDMI, DVI, VGA
Audio Output	5.1
Archiviazione	4x SATA
Sistema operativo	Linux, Windows
Dimensioni	standard Mini-ITX
Costo	100 Fr. + componenti

3.5 Scelta della piattaforma hardware

La soluzione che offre le migliori prestazioni in assoluto è una Mini-ITX con un classico processore Intel di ultima generazione. Tuttavia questo comporta un costo molto più alto rispetto alle altre soluzioni analizzate. I rimanenti tre dispositivi sono tutti basati su architetture ARM. Tra questi, il **Raspberry Pi** risulta essere la scelta dal rapporto qualità-prezzo migliore.

Sono state inoltre considerate eventuali espansioni future relative al progetto. Ad esempio il Raspberry Pi offre la possibilità di visualizzare immagini e video su uno schermo di dimensioni variabili e possiede una GPU integrata. Come si può notare dal sito del distributore svizzero [5], il Raspberry Pi offre anche un'ampia espandibilità con numerosi dispositivi e accessori.

Capitolo 4

Sviluppo

4.1 Target scelto

La piattaforma hardware Raspberry Pi, scelta come indicato in sezione 3.5, da sola non è sufficiente a soddisfare tutti i requisiti funzionali richiesti dal progetto. Si sono dovuti infatti aggiungere alcuni moduli supplementari che sono stati scelti in funzione delle necessità del progetto ed in seguito sottoposti a test per verificarne l'effettiva utilizzabilità entro i parametri richiesti. L'insieme completo dei componenti che compongono il sistema è mostrato in forma schematica nella figura 4.1.

Il primo elemento necessario è la fotocamera, indispensabile per acquisire le immagini da elaborare e inviare al server. Si è optato per la soluzione presentata in sezione 4.1.1: una piccola telecamera CMOS integrata montata su di un circuito stampato realizzata appositamente per essere utilizzata con un Raspberry Pi. Dato che ogni utente verrà identificato da un id univoco al momento dell'interazione con il dispositivo, occorre un sistema di identificazione univoco ed affidabile. A questo scopo si è pensato di utilizzare un modulo RFID associato a dei braccialetti da fornire in dotazione agli utilizzatori. La tecnologia RFID è ampiamente diffusa ed affermata nel campo dell'identificazione di oggetti, animali e persone applicata a svariati ambiti. Si basa su di una breve trasmissione radio emessa da un ricetrasmettitore e captata da un piccolo circuito (denominato *tag*), il quale invia al ricetrasmettitore una risposta contenente un codice identificativo univoco di vario genere. Il modulo RFID utilizzato è presentato nella sezione 4.1.2.

Per le comunicazioni di rete si è deciso di utilizzare moduli per reti Wireless e Bluetooth in quanto l'interfaccia Ethernet presente sul Raspberry Pi, poco si addice ad un sistema che con ogni probabilità si troverà in luoghi sprovvisti della possibilità di utilizzare una rete cablate. Le soluzioni analizzate sono presentate nelle sezioni 4.1.3 e 4.1.4. L'interazione con l'utente, dat anche l'assenza di uno schermo, sarà affidata a pulsanti, LED e suoni, come descritto in sezione 4.1.5. Infine, si è approcciato il problema della sopravvivenza del sistema ad una mancata alimentazione proponendo l'uso di una batteria di emergenza per

l'arresto pulito del sistema.

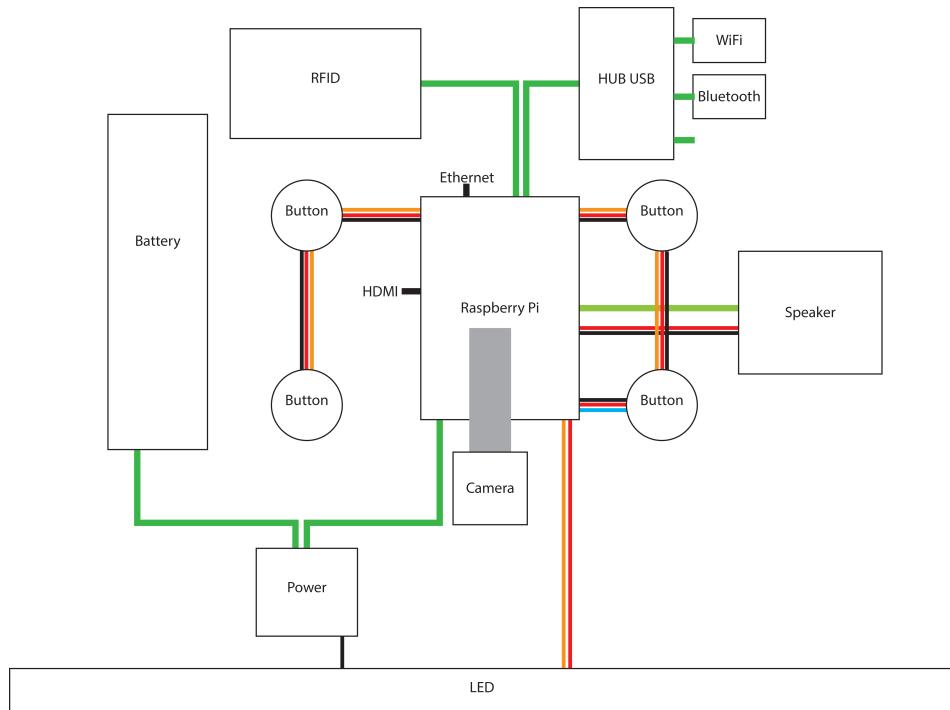


Figura 4.1: Schema a blocchi dei componenti hardware del dispositivo.

4.1.1 Fotocamera

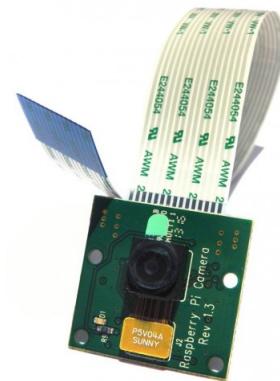


Figura 4.2: Camera per RaspberryPi [1].

Collegata direttamente con il dispositivo, la fotocamera è indispensabile per la cattura di immagini e video di qualità. Si tratta di una **Raspberry Pi Camera Board** [6], basata su di un chip CMOS, con le seguenti caratteristiche:

- OmniVision 5647 Camera Module;
- Fotografie: risoluzione 5MP, 2592 x 1944 pixel;
- Video: Supports 1080p @ 30fps, 720p @ 60fps e 640x480p 60/90 Recording;
- 15-pin MIPI¹ Camera Serial Interface;
- dimensioni: 20 x 25 x 9mm.

4.1.2 RFID - Radio Frequency IDentification



Figura 4.3: Modulo RFID prodotta da CAEN SpA.

Per il progetto è stato proposto un modulo RFID **Quark R1230C** prodotto da **CAEN SpA**. [7]. Il modulo comunica attraverso una scheda collegata tramite USB al Raspberry Pi. Una volta collegata ad un dispositivo o ad un computer di un sistema operativo Windows o Linux, l'antenna viene vista come una porta seriale standard virtualizzata sull'interfaccia USB da un chip FTDI² presente sul modulo. Il modulo viene quindi riconosciuto automaticamente con gli appositi driver ed è utilizzabile semplicemente come una porta seriale tradizionale. Il modulo CAEN supporta il protocollo do comunicazione a pacchetti **easy2read**, che funziona sia su collegamenti seriali che su collegamenti di rete. La potenza di trasmissione è programmabile via software e arriva ad un massimo di 200mW (variabile da 10dBm a 23dBm). A piena potenza, il modulo è capace di leggere un tag RFID alla distanza di 2 metri. Purtroppo questo raggio si riduce notevolmente se il tag RFID è tenuto in mano da una

¹Mobile Industry Processor Interface, interfaccia di comunicazione comunemente utilizzata in dispositivi mobili.

²FTDI - Future Technology Devices International.

persona, a causa delle interferenze che il corpo umano crea al momento della detezione del segnale tra l'antenna del modulo e quella del tag.

Come tag RFID di riferimento sono stati scelti dei braccialetti in gomma dotati di tag passivi che reagiscono alle frequenze **UHF**³ e seguono lo standard EPC C1 G2, ISO 18000-6C. Data che i braccialetti verranno indossati dagli utenti, si deve prevedere di posizionare l'antenna all'interno del dispositivo in modo tale da massimizzare l'efficacia di ricezione e trasmissione del segnale da e verso i tag, così da avere un'identificazione rapida e precisa.

4.1.3 Connessione WiFi



Figura 4.4: Antenna WiFi.

Il Raspberry Pi non dispone di connessioni di rete Wireless, ma il prototipo iniziale non può avere una connessione di rete cablata. Per sopperire a questa mancanza è stata aggiunta un *dongle* WiFi utilizzato dal sistema operativo sul Raspberry Pi come una normale scheda di rete. Le caratteristiche del *dongle* sono le seguenti:

- supporto WLAN B/G/N;
- supporto a WEP, WPA, WPA2, WPS/QSS;
- velocità di trasferimento: 150 Mbps;
- interfaccia USB 2.0;
- dimensioni 36 x 18 x 6mm.

³UHF - Ultra High Frequency, i tag scelti lavorano nelle frequenze da 860 MHz a 960 MHz.

4.1.4 Bluetooth



Figura 4.5: Dongle Bluetooth.

Per permettere una comunicazione tra il dispositivo e un sistema di controllo e amministrazione (ad esempio un'applicazione su smartphone o tablet) si è pensato di utilizzare un *dongle* Bluetooth. È stata utilizzata una **Nano USB to Bluetooth Dongle V2.0** con le seguenti caratteristiche:

- compatibile con Bluetooth v2.0 e v1.2;
- supporto per Networking, Dial-up, Fax, LAN Access e Headsets;
- velocità di trasferimento: 3 Mbps;
- interfaccia USB 2.0;
- Portata: 20m.

4.1.5 Periferiche aggiuntive

In aggiunta alle periferiche connesse tramite USB, ci sono anche alcuni componenti hardware che andranno collegati tramite i GPIO⁴ del Raspberry Pi. Assieme ai dati verranno utilizzati anche dei PIN di alimentazione presenti sulla scheda per fornire corrente ai componenti.

4.1.5.1 Bottoni

L'interazione tra l'utente e il dispositivo idealmente inizia con una pressione di una serie di pulsanti di tipo arcade collegati in serie. Per questa operazione si è scelto di utilizzare dei pulsanti del tipo **Sanwa OBSF 30mm Pushbuttons** appositamente modificati per essere collegati in serie e avere tre fili da collegare ai GPIO del Raspberry Pi.

⁴General Purpose Input/Output, 26 pin di collegamento per periferiche non standard.



Figura 4.6: Bottoni utilizzati.

4.1.5.2 Speaker e LED

Non avendo inizialmente a disposizione uno schermo per mostrare dei dati, il dispositivo comunicherà il suo stato all'utente attraverso informazioni sonore e visive. A questo scopo si è scelto di utilizzare uno speaker con amplificatore per inviare dei messaggi audio e una striscia di LED per informare l'utente sullo stato del dispositivo.

In una versione successiva del dispositivo si pensa ad una variante dotata di un piccolo schermo o di posizionare la porta HDMI del Raspberry in modo che sia possibile collegarla ad uno schermo esterno e mostrare alcune informazioni.

4.1.5.3 Alimentazione e Batteria

Dal punto di vista dell'alimentazione il Raspberry Pi è molto semplice. L'idea del modulo è quella di essere acceso un'unica volta e funzionare in continuazione. Nel caso in cui occorra spegnerlo, deve essere il software a bordo a disattivare in sicurezza il dispositivo. La scheda offre un circuito di reset hardware che però va cablato ad un pulsante.

Per funzionare il Raspberry Pi e le sue componenti necessitano di un alimentatore capace di erogare **5V** di tensione e almeno **1200mA** di corrente.

Si è scelto quindi di aggiungere una batteria ricaricabile all'interno del dispositivo per il mantenimento temporaneo del suo funzionamento in caso di assenza dell'alimentazione principale. In questo modo, oltre a garantire uno spegnimento pulito del sistema, sarebbe inoltre possibile spostare in sicurezza il dispositivo anche senza alimentazione ed effettuare presentazioni o dimostrazioni. Il circuito di alimentazione è stato adattato di conseguenza per poter gestire il cambiamento dell'alimentazione da batteria a corrente diretta in modo da alternare la carica e scarica della batteria stessa.

L'idea di base è un accorgimento che permetta alla batteria di subentrare all'alimentazione principale quando questa viene a mancare. La batteria fornisce corrente per circa un'ora a tutto il sistema, inclusi i moduli di comunicazione (RFID, WiFi, . . .).

Per gestire questo cambiamento di stato, un filo collega un pin GPIO del Raspberry con il circuito di alimentazione in modo da avvisare il dispositivo dell'avvenuto passaggio da un tipo di alimentazione ad un altro. Il software sul Raspberry Pi è responsabile delle operazioni successive a questo evento, che teoricamente portano alla disattivazione del sistema.

4.1.6 Hardware implementato

In seguito a scelte effettuate in fase di implementazione, spiegate nel capitolo 5, alcuni componenti previsti non sono stati collegati o sono stati rimossi. Dato che il dispositivo realizzato in questo progetto è un primo prototipo funzionale, le componenti hardware sono state collegate in maniera estremamente semplice, spesso sfruttando le connessioni dei cavi USB.

Allo stato finale dei lavori, solamente il modulo WiFi, la camera e il modulo RFID sono pienamente funzionanti ed utilizzati, come mostrato dallo schema in figura 4.7. L'implementazione dei pulsanti, sebbene ad uno stadio avanzato, non ha raggiunto uno livello di stabilità soddisfacente: spesso i pulsanti creavano disturbi ed interferenze e si sono dimostrati difficilmente compatibili con l'uso dei braccialetti RFID.

Per quanto riguarda la striscia di LED di segnalazione, per funzionare necessita di un'alimentazione a 12V, ma il sistema attuale si basa un'alimentazione a 5V per tutti i componenti, che è risultata non adatta ad accendere la striscia di LED. Per questi motivi, pulsanti e LED non sono utilizzati.

Il circuito di alimentazione pensato inizialmente non è stato sfruttato. Il prototipo può essere alimentato direttamente dalla corrente oppure da una batteria USB e può sfruttare la corrente per caricare la batteria, ma in caso di assenza di corrente (cavo scollegato, batteria scarica) il dispositivo si spegne in maniera non sicura.

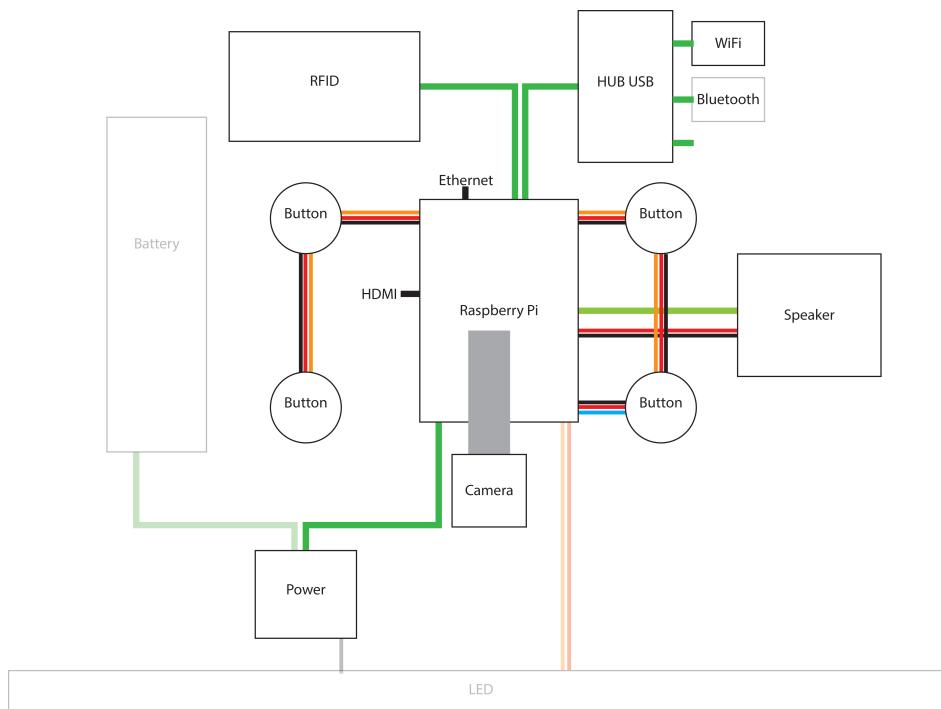


Figura 4.7: Schema a blocchi dei componenti hardware utilizzati per il prototipo.

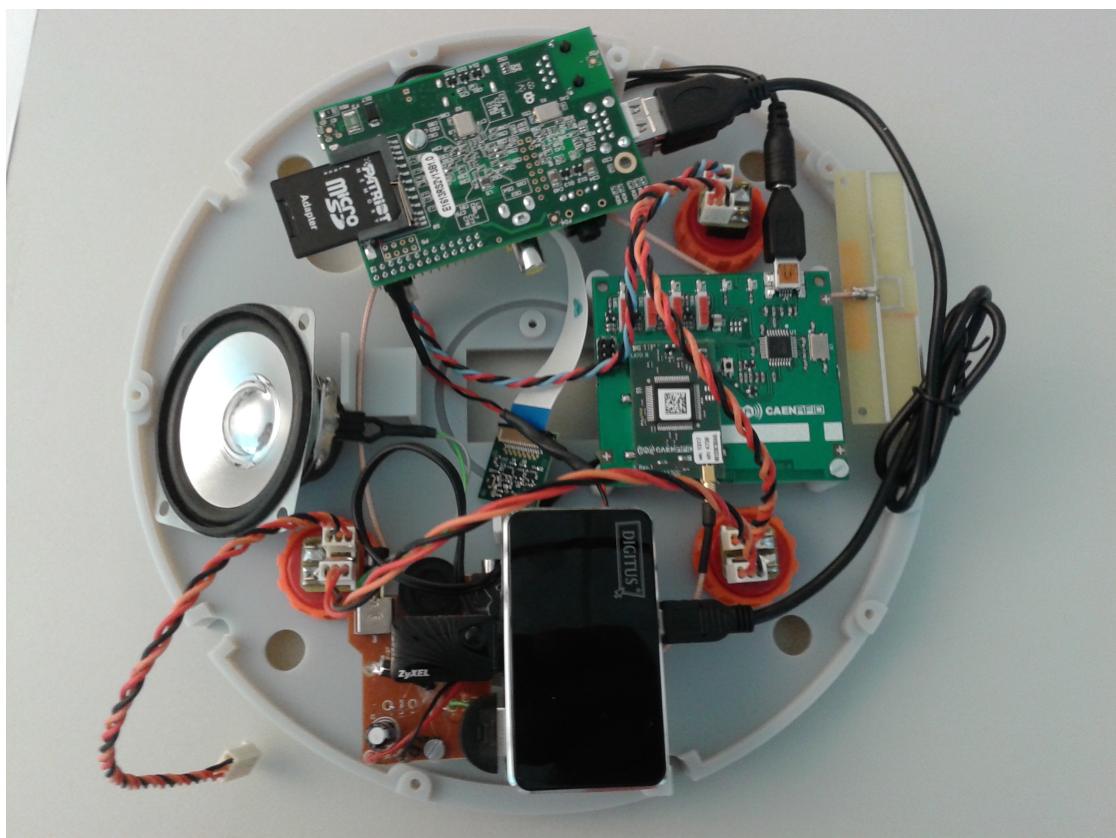


Figura 4.8: L'interno del dispositivo con i componenti collegati.

4.2 Ambiente target e ambiente di sviluppo

4.2.1 Sistema operativo target

Appurato che il target è un Raspberry Pi con processore ARM 1176JZ-F 3.1, occorre scegliere quale sistema operativo utilizzare per lo sviluppo di tutta l'architettura software necessaria. Le scelte disponibili all'inizio del progetto erano le seguenti:

- una distribuzione Linux,
- una distribuzione Android,
- nessun sistema operativo.

Considerando la complessità dell'architettura di rete necessaria alla comunicazione con il server aziendale, basti pensare allo *stack* ISO/OSI da implementare da zero, l'opzione di non utilizzare un sistema operativo non è stata presa in considerazione.

Le versioni di Android disponibili per Raspberry Pi, purtroppo, sono instabili e spesso di difficile utilizzo, frutto sovente di *porting* approssimativi. Data la necessità di una certa stabilità del dispositivo, si è quindi optato per l'unica scelta rimasta: **una distribuzione Linux**.

L'utilizzo di un sistema operativo completo e facilmente modificabile come Linux è un ottimo punto di partenza per la costruzione di un'architettura software. I creatori di Raspberry Pi consigliano l'utilizzo di una distribuzione basata su Debian chiamata *Raspbian* [8], figura 4.9. Dato che Debian viene sempre considerato come una roccia in ambiente Linux, sono infatti numerosi i riferimenti alla sua stabilità anche durante i corsi svolti durante i tre anni, il consiglio è stato accettato senza alcuna opposizione.

È rimarchevole il fatto che un dispositivo compatto come un Raspberry Pi, così come molti altri moduli basati su architettura ARM (specialmente dalla v6/v7 in poi), permetta l'utilizzo di un sistema operativo completo con un ambiente grafico e un ampio parco software con programmi già esistenti su repository dedicati. Gli unici grossi problemi che potrebbero emergere utilizzando un sistema operativo del genere, sono legati ad eventuali incompatibilità provocate da librerie mancanti che andranno, per forza di cose, compilate partendo dai sorgenti. Dato che le distribuzioni derivate da Debian aderiscono alle direttive del movimento *Open Source*, i codici sorgente sono a disposizione per poter essere ricompilati su necessità. A questo proposito è interessante far notare che le attuali versioni del kernel Linux (successive alla release 3.0) sono concepite per poter essere facilmente compilate per processori ARM.

La possibilità di **cross-compilare** i sorgenti disponibili è qualcosa che non va sottovalutato. Sovente la potenza di calcolo offerta da moduli come il Raspberry Pi è notevolmente minore

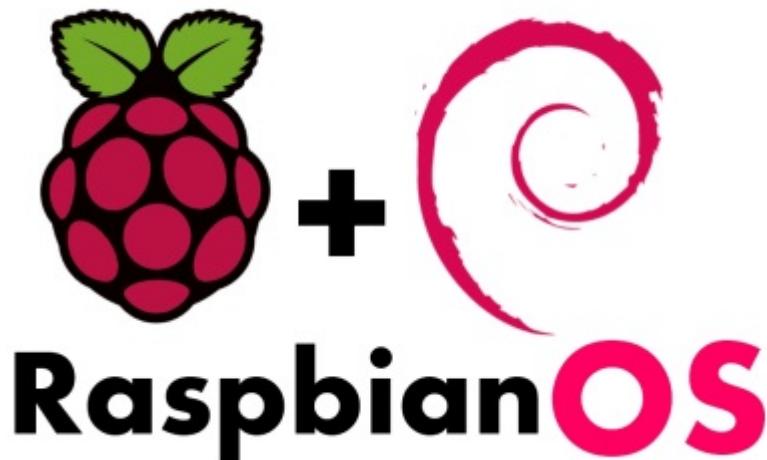


Figura 4.9: Il logo di Raspbian.

a quella di un comune computer o comunque poco adatta alla compilazione di codice. Questo perché i questi sistemi, definiti *embedded*, sono spesso concepiti per lavorare in condizioni particolari, limitare i consumi o per essere incorporate in apparecchi portabili. Tutte situazioni che non prevedono grande potenza computazionale. Per evitare di passare giorni a compilare codice anche semplice, viene utilizzato un cosiddetto *cross-compiler*; cioè un compilatore capace di produrre codice eseguibile su architetture differenti da quella in cui funziona. Nel caso del Raspberry Pi, un computer con architettura x64 può tranquillamente compilare codice eseguibile su ARM11 a 32bit.

Anche la compatibilità con altre componenti hardware è garantita. Infatti, ARM impone l'utilizzo di uno standard di sviluppo software a basso livello, l'interfaccia CMSIS⁵ [9], grazie alla quale viene garantita la portabilità del sistema a basso livello. Questo facilita il *porting* e il mantenimento dei driver e di tutte le altri applicazioni alla base del sistema, contribuendo a migliorarne l'efficacia.

In un sistema Debian la maggior parte dei componenti hardware possiede dei moduli generici funzionanti e subito utilizzabili, oltre ad altre API specifiche per la facile creazione di nuovi moduli e l'interfacciamento con connessioni hardware generiche, come i GPIO di un Raspberry Pi.

4.2.2 Linguaggio e librerie di sviluppo

Utilizzando un sistema operativo Linux, i linguaggi disponibili per la programmazione del software sono numerosi. Le analisi e gli studi compiuti dal DACD, vedi Allegato 2, fanno riferimento ad un sistema espandibile in futuro con parti grafiche. Il sistema software deve

⁵CMSIS - Cortex Microcontroller Software Interface Standard.

inoltre poter controllare in maniera efficiente il modulo RFID e avere una gestione della rete autonoma. Inoltre, occorre una libreria per la manipolazione di immagini. Le manipolazioni vanno dall'applicare semplici filtri e ritagliare immagini al riconoscimento di caratteristiche e oggetti.

Prima del linguaggio, sono state scelte le librerie da utilizzare. Grazie all'evoluzione avuta negli ultimi tempi, le **librerie Qt** sono giunte ad un grado di completezza tale da offrire tutto il supporto necessario alla costruzione non solo di un software completo, ma anche capace di sfruttare interfacce e comunicazioni di rete. Grazie anche al nuovo linguaggio per interfacce grafiche **QML**, è possibile creare una rappresentazione grafica univoca per più piattaforme distinte. Il **Qt Project** ha fatto un enorme lavoro per portare le librerie Qt 5 su Raspberry Pi [10] con ottime performance [11].

Per quanto riguarda la manipolazione di immagini, è stata scelta la libreria **OpenCV** per la sua vasta offerta di algoritmi per la manipolazione grafica e la sua portabilità su svariate architetture, grazie al supporto del *cross-compiling*.

Alla luce delle librerie utilizzate, si è scelto di scrivere l'intera architettura software utilizzando prevalentemente **C/C++**. Al momento dell'inizio dei lavori, la versione per sistemi *embedded* di Java non era ottimizzata e non offriva performance adatte allo scopo. Solamente di recente Java è stato ottimizzato anche per Raspberry Pi [12].

Data la potenza di calcolo inferiore di un Raspberry Pi rispetto ad un computer comune, si è reso necessario fin da subito di utilizzare un **cross-compiler A**.

4.2.3 Ambiente di sviluppo

Il software è stato sviluppato su un computer con **Kubuntu 13.04** in cui è stato installato il cross-compiler **gcc-linaro-arm-linux-gnueabihf-raspbian**, compilatore consigliato dal Raspberry Pi group. Nell'appendice A verrà spiegato come installare ed utilizzare questo compilatore.

Per eseguire il deployment e il debugging remoto del software, il dispositivo è stato dapprima collegato ad una rete LAN e in seguito collegato ad una rete Wireless dalle caratteristiche simili alla rete finale dove sono stati compiti i test.

Sul computer sono state inoltre installate le librerie **Qt 4.7**, **Qt 5.0** e **OpenCV 2.4.0**, per lo sviluppo locale di applicazioni, e si è provveduto a cross-compilare le librerie Qt 5.0 e OpenCV 2.4.0 per il funzionamento su Raspberry Pi. Cross-compilare le librerie è risultato notevolmente più veloce di compilare nativamente sul Raspberry Pi: l'installazione completa di OpenCV, ad esempio, ha richiesto poco meno di un ora, mentre l'installazione direttamente su Raspberry Pi avrebbe richiesto più di otto ore. Nelle appendici B e D verrà spiegato come installare correttamente le librerie per Raspberry Pi.

Come IDE di sviluppo si è scelto di utilizzare **Qt Creator 2.6.1**. Questo IDE, dopo essere stato configurato correttamente, fornisce già le funzioni per il deployment e il debugging

remoto delle applicazioni sul Raspberry Pi. Nell'appendice C verrà spiegato come è stato configurato.

Per la configurazione e l'installazione di pacchetti aggiuntivi del Raspberry Pi si è passati da un controllo diretto con uno schermo e tastiera collegati al dispositivo ad un controllo remoto dopo aver configurato il dispositivo per connettersi in maniera automatica ad alcune reti, cablate o senza fili. La connessione e lo scambio di file tra dispositivo e computer sono avvenuti mediante l'utilizzo dei comandi **ssh** e **scp**.

4.3 Software sviluppato

Il software si compone di due parti separate ma comunicanti: un **demone** di sistema sempre attivo e un software **applicativo**.

4.3.1 Librerie RFID

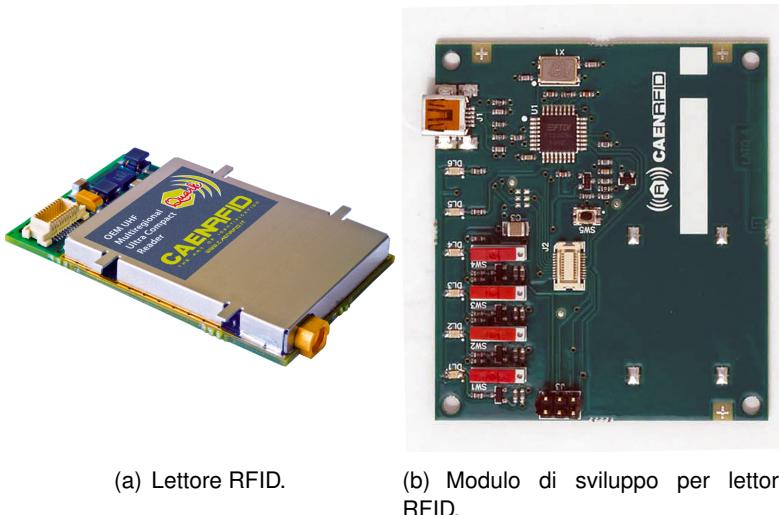


Figura 4.10: Kit di sviluppo RFID CAEN SpA. in dotazione per il progetto.

Il kit di sviluppo della CAEN SpA. in dotazione per questo progetto è mostrato in figura 4.10 e consiste del lettore RFID Quark R1230C (a) montato su di una scheda di sviluppo (b). Il lettore RFID è praticamente autonomo e in grado di operare direttamente con i tag RFID una volta collegato ad un'antenna. La scheda di sviluppo serve unicamente come supporto per l'alimentazione e l'interfacciamento tra la seriale UART standard di cui il modulo Quark R1230C è dotato per comunicare e l'USB. Il componente che permette questo interfacciamento è un chip FTDI, in grado di trasferire i dati provenienti dalla UART del lettore verso l'USB e viceversa, virtualizzando una porta seriale mediante il protocollo standard VCOM. Insieme questi due elementi costituiscono il modulo RFID collegato al Raspberry Pi, essenziale al funzionamento del dispositivo.

La CAEN SpA. ha fornito una libreria con codice sorgente disponibili unicamente per Windows e scritte in C, C# e Java. In un primo momento si è pensato di eseguire il *porting* delle librerie scritte in C per Windows, l'operazione però non è riuscita a causa dell'utilizzo di librerie di sistema di Windows, non disponibili su Linux. È stato provato anche un artificio per aggirare il problema ed utilizzare le librerie di rete per comunicare con il dispositivo, senza successo. Fortunatamente la documentazione allegata al modulo descriveva in maniera esaurente il protocollo *easy2read*, utilizzato per comunicare con il dispositivo.

4.3.1.1 Protocollo easy2read

Il modulo RFID utilizza il protocollo di comunicazione proprietario di CAEN SpA **easy2read** (Allegato 3), basato sullo schema *Attribute Value Pair* (AVP), ovvero uno scambio di pacchetti formati da piccoli coppie attributo-valore tra host e lettore RFID.

Un messaggio di questo protocollo si compone di un *header* di lunghezza fissa e di una lista di frame indicanti gli attributi con il loro relativo valore.

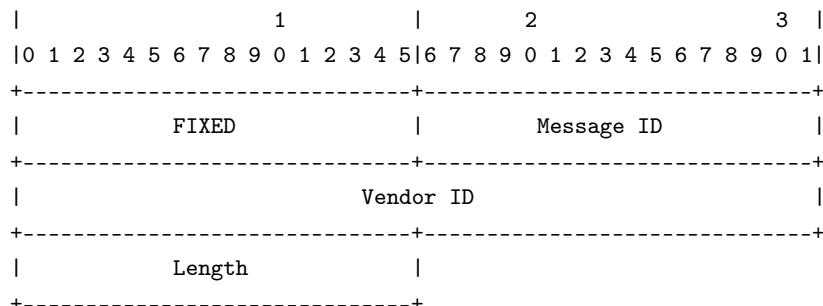
Gli *header*, come mostrato nel listato 4.1, sono una struttura dati di 80 bit suddivisi nei quattro campi elencati di seguito:

FIXED : (16 bit) valore fisso a 0x8001 per i comandi e a 0x0001 per le risposte.

Message ID : (16 bit) identificativo del messaggio. Ogni messaggio contenente un comando genera una risposta. Per identificare a quale comando corrisponde quale risposta viene utilizzato questo campo.

Vendor ID : (32 bit) per i dispositivi di CAEN SpA. il valore deve essere 0x00005358, corrispondente al codice assegnato all'azienda dal consorzio IANA⁶.

Length : (16 bit) indica la lunghezza totale del messaggio espressa in byte, *header* incluso.



Listato 4.1: Struttura del header di un pacchetto easy2read.

Dopo l'header viene accodata una lista di frame. I frame che compongono la lista di comandi sono composti secondo la struttura indicata nel listato 4.2. Le sue parti sono indicate qui di seguito:

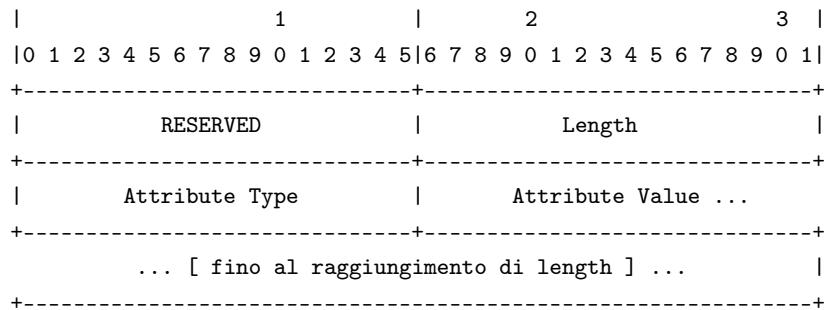
⁶IANA - Internet Assigned Numbers Authority.

RESERVED : (16 bit) 2 byte riservati che vanno impostati a 0x0000.

Length : (16 bit) lunghezza complessiva dell'intero frame in byte.

Attribute Type : (16 bit) codice di 2 byte corrispondente al tipo di attributo.

Attribute Value : parte a lunghezza variabile che contiene il valore dell'attributo indicato dal frame. La lunghezza di questo campo dipende dal tipo di valore ed calcolata come $Length - 6$.



Listato 4.2: Struttura di un frame di un pacchetto easy2read.

La lista di comandi e attributi è disponibile nella descrizione del protocollo dell'Allegato 3.

4.3.1.2 Comunicazione

La comunicazione tra lettore e *host* può avvenire su diversi canali di comunicazione: rete, seriale, Bluetooth. Le librerie sviluppate da CAEN SpA., scritte in Java, C++ e C#, sono in stretta relazione con le interfacce di rete di Windows per cui si dimostrano inutilizzabili direttamente in ambiente Linux.

Dopo alcuni tentativi infruttuosi per tentare un *porting* completo delle librerie, si è deciso di riscriverle da capo in C++, concentrandosi sulla connessione seriale come unico canale di interscambio dati tra il sistema operativo sul Raspberry Pi ed il modulo RFID. Il motivo di questa scelta è duplice: fondamentalmente è il sistema che il Raspberry Pi utilizza per comunicare fisicamente con il modulo RFID (tramite l'interfaccia USB), ma è anche il sistema più elementare per trasmettere i frame in modo controllato. Eventuali estensioni per supportare la rete o applicazioni di alto livello saranno facilmente implementabili modificando le classi che gestiscono la comunicazione seriale.

4.3.1.3 Esempio di comandi

Il protocollo easy2read contiene numerosi comandi compatibili con diversi dispositivi, moduli e lettori RFID. Anche se qui è presentato unicamente un esempio, la documentazione (Allegato 3) offre ulteriori esempi di facile comprensione.

Come detto, la comunicazione tra lettore e *host* avviene tramite lo scambio di pacchetti. Ogni pacchetto contiene, oltre al *header*, un comando con i relativi parametri. Il nome del comando e il valore dei parametri occupano tutti un frame.

Se ad esempio si vuole impostare la potenza del segnale, occorre inviare un messaggio composto da un *header* e due frame: nel primo frame verrà indicato il comando utilizzando la coppia attributo-valore seguente:

```
Attribute Type = CommandName      (0x0001)
Attribute Value = Set Power       (0x0064)
```

Listato 4.3: Coppia AVP per il nome del comando.

Mentre nel secondo frame verrà indicato il valore con un'altra coppia. Il valore è il massimo valore raggiungibile dal modulo RFID utilizzato.

```
Attribute Type = Power Set        (0x0001)
Attribute Value = 199             (0x000000C7)
```

Listato 4.4: Coppia AVP per un parametro relativo al comando.

Il messaggio completo si compone quindi della seguente struttura:

HEADER	Fixed	= 32769	(0x8001)
	Message ID	= 0	(0x0042)
	Vendor ID	= 21336	(0x00005358)
	Message Length	= 28	(0x001C)

FRAME 1	Reserved	= 0	(0x0000)
	Length	= 8	(0x0008)
	Attribute Type	= CommandName	(0x0001)
	Attribute Value	= Set Power	(0x0064)

FRAME 2	Reserved	= 0	(0x0000)
	Length	= 10	(0x000A)
	Attribute Type	= Power Set	(0x0096)
	Attribute Value	= 199	(0x000000C7)

Listato 4.5: Messaggio inviato dal host al lettore.

In risposta ai messaggi inviati dal *host*, il lettore RFID ne invierà uno simile contenente l'esito del comando ed eventualmente i parametri richiesti con il primo comando. Ovviamente, anche questa risposta sarà suddivisa in un *header* seguito da una lista di frame, contenenti questa volta una risposta. Ad esempio, la risposta al comando precedente è la seguente:

HEADER	Fixed	= 1	(0x0001)
	Message ID	= 0	(0x0042)
	Vendor ID	= 21336	(0x00005358)
	Message Length	= 26	(0x001A)
FRAME 1	Reserved	= 0	(0x0000)
	Length	= 8	(0x0008)
	Attribute Type	= CommandName	(0x0001)
	Attribute Value	= Set Power	(0x0064)
FRAME 2	Reserved	= 0	(0x0000)
	Length	= 8	(0x0008)
	Attribute Type	= Result Code	(0x0001)
	Attribute Value	= Success	(0x0000)

Listato 4.6: Messaggio in risposta proveniente dal lettore.

Analizzando il protocollo ad alto livello, i due messaggi nell'esempio del listato 4.6 non sono altro che l'equivalente di una chiamata ad un'ipotetica funzione di questo tipo:

```
bool setPower(int value)
```

4.3.1.4 Librerie implementate

Partendo dall'analisi compiuta sul protocollo, è stata scritta una libreria in C++ capace di gestire in maniera automatica i messaggi. Questa è strutturata in tre livelli distinti, gestiti da tre classi:

SerialModule classe utilizzata per comunicare con il lettore RFID attraverso un canale seriale. I parametri per stabilire la comunicazione sono indicati nella tabella 4.1.

RFIDMessage classe utilizzata per descrivere un messaggio. Questa classe viene utilizzata sia per costruire un messaggio da inviare al lettore, sia per decifrare una risposta proveniente dal lettore. La prima implementazione di questa classe fornisce unicamente i metodi per comporre il messaggio, con la possibilità di aggiungere frame utilizzando la serie di metodi `addCommand()`, di ricavare gli RFID letti con una richiesta di inventario con il metodo `getRFIDs()`, verificare il parametro `Result Code` della risposta con il metodo `success()`, stampare a schermo il messaggio.

RFIDModule classe utilizzata per esporre dei semplici metodi relativi ad ogni comando disponibile, nascondendo l'implementazione dello scambio dei messaggi e dell'invio attraverso la seriale. La classe offre un metodo base per l'invio e la ricezione di un messaggio qualsiasi (`sendAndReceive()`) e una serie di comandi già implementati

nella forma vista nella sottosezione precedente. Ogni metodo ritorna il messaggio ricevuto dal lettore.

Tabella 4.1: Parametri utilizzati per la comunicazione sulla porta seriale.

Baudrate	115200 kbps
Bit dati	8
Stop bit	1
Parità	nessuna
Controllo di flusso	nessuno

L'utilizzo della libreria è pensato per semplificare la gestione e composizione dei messaggi. I passi da compiere per utilizzarla sono i seguenti:

1. Creare un oggetto di tipo `SerialModule` e configurarlo con i parametri corretti (si veda la tabella 4.1).
2. Creare un oggetto di tipo `RFIDModule` a cui verrà passato per riferimento l'oggetto di tipo `SerialModule`.
3. Utilizzare i comandi già pronti per comunicare con il lettore oppure comporre un proprio messaggio partendo da un oggetto della classe `RFIDMessage`.
4. Lavorare con l'oggetto di tipo `RFIDMessage` in risposta utilizzando i metodi della classe `RFIDMessage`.

Nella libreria sono inoltre disponibili due classi di supporto: `RFIDAttributeTypes` e `RFIDCommandsCode`. Queste classi non sono altro che la lista di tutti i comandi e i parametri disponibili spiegati nella documentazione del protocollo RFID. Utilizzando queste tre classi si può alleggerire il lavoro di composizione dei messaggi, passando da questa sintassi:

```
RFIDMessage *message = new RFIDMessage(0x1234);
message->addCommand(0x0001, 0x0064);
message->addCommand(0x0096, 0x000000C7);
```

a questa, più prolissa ma anche più esplicita:

```
RFIDMessage *message = new RFIDMessage(0x1234);
message->addCommand(RFIDAttributeTypes::COMMAND_NAME, RFIDCommandsCodes::SET_POWER);
message->addCommand(RFIDAttributeTypes::POWER_SET, 0x000000C7);
```

4.3.1.5 Pubblicazione

Si confida nella possibilità di fornire direttamente al produttore del modulo, CAEN SpA., queste API in modo da poter supportare appieno anche i sistemi Linux. In alternativa si spera di poter pubblicare queste API come open source.

4.3.2 Demone di sistema

Un **demone di sistema** è un processo che opera in background nel sistema operativo ed esegue determinate operazioni di supporto all'applicazione principale che ne fa uso. Nel caso specifico, l'utilizzo di un demone di sistema si è reso necessario in quanto il controllo dei GPIO del Raspberry Pi è gestito da *interrupt* utilizzabili solo da un utente con accesso privilegiato. La creazione e la gestione degli *interrupt* è stata delegata all'ottima libreria **WiringPi** [13]. Nell'appendice E sarà spiegato come viene installata.

Il demone è stato scritto interamente in C++ seguendo le linee guida per la scrittura di demoni per sistemi operativi Linux [14] [15].

Nonostante i GPIO non siano correntemente utilizzati, il demone si è rivelato utilissimo anche come controllore della vita dell'applicativo e per la gestione delle connessioni di rete. I compiti del demone sono dunque i seguenti:

- interrogare la rete a intervalli regolari per assicurarsi che vi sia una connessione valida;
- avviare l'applicativo e verificarne lo stato;
- modificare la modalità di funzionamento dell'applicativo;
- informare l'applicativo di eventi hardware sui componenti collegati ai GPIO.

4.3.2.1 Ciclo di vita

In figura 4.11 è mostrato il ciclo di vita del demone a partire dall'avvio del sistema al suo spegnimento. Dopo aver eseguito l'inizializzazione esegue fino alla sua terminazione le operazioni che costituiscono il *ciclo principale*.

4.3.2.2 Ciclo principale del demone

In figura 4.12 è mostrato lo schema del ciclo principale del demone. Sostanzialmente il demone rimane in attesa per un periodo che può variare da quindici secondi ad un minuto. Quando l'attesa termina, viene eseguito un controllo della stabilità della rete e, in base al risultato di questa valutazione, vengono prese delle decisioni. Nel caso in cui il Raspberry Pi sia online, ma non viene rilevato l'applicativo, il demone si incarica di avviare l'applicativo stesso. Nel caso invece in cui la connessione di rete cambia (passando da online a offline e



Figura 4.11: Sequenza di avvio del demone.

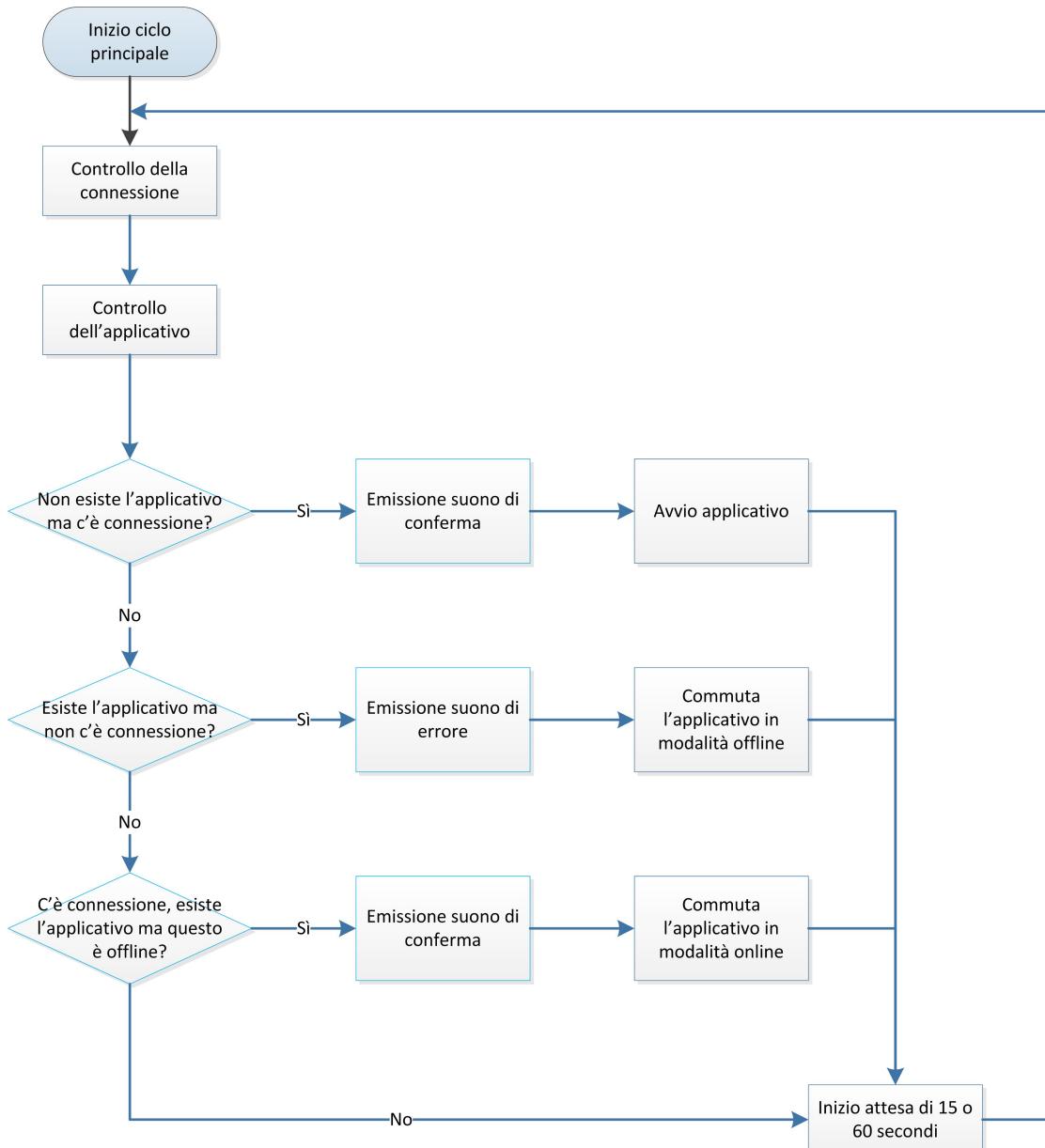


Figura 4.12: Ciclo principale del demone.

viceversa) il demone comunica all'applicativo di modificare la sua modalità operativa. Una volta completato un ciclo, il demone torna in uno stato di attesa.

L'intercomunicazione tra il demone e l'applicativo è basata interamente su segnali POSIX, gestiti dal sistema operativo Linux. È interessante notare che il demone deve sempre accertarsi che l'applicativo stia funzionando prima di lanciare eventuali segnali in quanto se il demone (privilegiato) lanciasse un segnale ad un processo inesistente causerebbe problemi all'intero sistema operativo. Per ovviare a questo problema il demone si assicura che l'applicativo sia effettivamente attivo mediante il segnale 0, che non provoca danni se il processo verso il quale si lancia il segnale non esiste.

4.3.3 Software applicativo

Il software applicativo si preoccupa del funzionamento del dispositivo ed è stato scritto utilizzando le librerie Qt. I suoi compiti principali sono i seguenti:

- scansione degli RFID presenti,
- controllo della validità degli RFID scansionati,
- acquisizione, manipolazione e invio delle immagini al server.

In supporto all'applicativo sono stati creati degli script in **bash** e utilizzati dei programmi già esistenti per compiere determinate azioni. I programmi utilizzati già esistente sono i seguenti:

- **raspistill** software già incluso in Raspbian per l'acquisizione di immagini con la camera,
- **mpg123** software per la riproduzione di file mp3,
- utilizzo di **crontab** per l'eliminazione di immagini e la gestione della rete wireless.

Invece, i programmi e gli script creati per il funzionamento sono i riportati qui di seguito:

- **cropper** software per il ritaglio delle immagini utilizzante la libreria OpenCV,
- **checkip** semplice script per il controllo dell'acquisizione di un indirizzo IP via rete WiFi,
- **crontab** aggiunti dei comandi di manutenzione e per le richieste DHCP,
- **stouchd** script per l'esecuzione del demone di sistema, creato sulla base dello schema di esempio.

4.3.3.1 Idea di base

La pianificazione del software è partita con l'idea di sfruttare al massimo il sistema di **segnali e slot** di Qt. Data la presenza di numerosi componenti con cui comunicare, si è deciso di sfruttare un'architettura composta di moduli dedicati a compiti specifici.

CameraModule Questo modulo è dedicato all'acquisizione di immagini con la fotocamera e alla loro manipolazione. Per il prototipo costruito, la manipolazione si compone di un ritaglio eseguito con il programma di supporto **cropper** basato su OpenCV. Per compiere l'operazione, il modulo immagazzina la posizione di due punti necessari al ritaglio. Modificando o sostituendo questo programma è possibile aggiungere filtri speciali alle immagini come effetti di colore, sfocature o anche scritte. Questo garantisce già da ora una semplice espandibilità.

HTTPModule Per comunicare con il server vengono utilizzati due moduli distinti. Il primo dei quali è questo, il cui compito consiste nel richiamare delle *servlet* sul server di supporto. Per contattare il server viene eseguita una richiesta HTTP GET ad un URL specifico. In base ai parametri inviati il server risponde con un oggetto JSON che viene gestito dall'applicativo. La chiamata si compone di cinque parametri in base ai quali il server decide le operazioni da compiere. I parametri utilizzati sono:

idUser può essere un qualunque id numerico, anche un MAC Address,

comando comando da eseguire sul server,

mac simile a idUser,

param utilizzabile per inviare una serie di parametri concatenati al server,

risposta nome dell'oggetto JSON ricevuto in risposta.

Il sistema gestisce le risposte in base al parametro **risposta**.

Queste chiamate servono principalmente per la gestione degli utenti e la verifica dei tag RFID acquisiti.

SendModule È grazie a questo secondo modulo per la comunicazione che è possibile inviare immagini al server. Le immagini inviate sono codificate in base64. Esistono due tipi di immagini che vengono inviate al server: le **VIEWIMG**, utilizzate come riferimento per il ritaglio, e le **LOADIMG**, immagini scattate dagli utenti con tag RFID registrati.

La stringa da inviare al server si compone come i parametri utilizzati dal modulo **HttpModule** per compiere le richieste HTTP GET, con la differenza che il campo **param** contiene l'intera immagine codificata.

LogModule Ogni azione compiuta dal software e ritenuta importante viene registrata su un file di log. Alcune informazioni ritenute ancora più importanti sono inviate al server attraverso il modulo HTTPModule.

RFIDModule Questo software modulo è l'unico sistema per interagire con il modulo RFID. Il suo scopo primario è effettuare scansioni ad intervalli regolari e segnalare al modulo corretto il rilevamento di almeno un tag RFID.

Per comunicare con il modulo RFID attraverso la porta seriale, utilizza un oggetto di tipo SerialModule correttamente configurato.

SerialModule Una volta collegato al Raspberry Pi, il modulo RFID viene visto come dispositivo seriale. Questo modulo software va quindi ad interfacciarsi con una seriale generica e apre i canali di comunicazione e ricezione con la stessa.

SignalsModule Per inviare e ricevere segnali di sistema POSIX si è reso necessaria la creazione di questo modulo. Il suo scopo è informare il software dei segnali provenienti dal sistema operativo e di inviarne a sua volta in base ai comandi ricevuti dal software. È l'unico modulo in contatto diretto con il demone.

SoundModule Per la riproduzione di file audio viene utilizzato il programma **mpg123**. La gestione di questo programma è affidata a questo modulo.

RaspberryPi Questo è il modulo centrale che coordina il funzionamento dell'intera architettura software. Si preoccupa di gestire le risposte delle varie chiamate al server, di riportare o risolvere errori. Ogni azione del flusso e del ciclo di vita dell'applicazione viene gestita in questo modulo.

4.3.3.2 Interazione tra i moduli

Il software si compone di sei QThread, più la thread principale (`main`). Ogni thread sfrutta il ciclo degli eventi delle librerie Qt per attendere i segnali provenienti dai vari moduli che compongono il software. I thread sono di conseguenza dei contenitori di oggetti e non esistono oggetti derivati da QThread [16].

4.3.3.3 Sistema di log

Tutti i moduli derivano da una classe `Module` capace di emettere il segnale `log`. Tale segnale viene intercettato da un oggetto di tipo `LogModule` che scrive il messaggio su un file. In aggiunta, questo oggetto emette a sua volta un segnale intercettato da un oggetto di tipo `HTTPModule` che si occuperà di scrivere sul server. Solamente i messaggi più importanti

vengono scritti anche sul server. I messaggi indicanti il tipo INFO non vengono scritti sul server ma solo su log locale.

4.3.3.4 Ciclo di vita

All'avvio, l'applicazione si auto configura richiedendo ai vari componenti eventuali informazioni e contattando il server per la registrazione del dispositivo. Ad inizializzazione completa, l'applicazione inizia a richiedere ad intervalli regolari delle richieste di inventario al lettore RFID. Nonostante il modulo hardware del RFID fornisca un'opzione di inventario continuo, si è scelto di controllare questo evento in maniera software in modo da poter scegliere l'intervallo ogni quanto effettuare la scansione.

Quando l'utente avvicina il proprio tag RFID al dispositivo, la scansione acquisisce il tag e fa partire una cascata di segnali. Nel caso in cui il tag RFID non è stato precedentemente registrato, viene eseguita la sequenza descritta nel diagramma di figura 4.13. In caso contrario, l'operazione viene completata e viene scattata una foto, seguendo il diagramma di figura 4.14.

4.3.3.5 Invio di immagini al server

In figura 4.14 viene anche mostrata la sequenza compiuta dal modulo SendModule per inviare le immagini al server.

Solitamente, la comunicazione con il server e altri client avviene attraverso la serializzazione di oggetti Java. Dato che l'applicativo è stato sviluppato in C++ non è possibile serializzare nel medesimo modo gli oggetti. Come *workaround* è stata modificata il funzionamento di una servlet per adattarsi ad un socket. L'applicativo, prima di inviarla, codifica l'immagine in base64 e incapsula il risultato in una stringa simile all'indirizzo URL utilizzato per contattare il server. Il server effettua quindi un parsing dell'enorme stringa ricevuta e ricompone l'immagine, inviandola come cartolina pubblicitaria all'indirizzo email associato.

4.3.3.6 Modalità online/offline

In caso di assenza di rete, il dispositivo può funzionare in **modalità offline**. In questa modalità i tag RFID vengono acquisiti e ad ogni tag viene assegnata una foto, indipendentemente dal fatto che il codice RFID sia registrato o meno. La modalità offline può essere attivata solamente dal demone, ma l'applicativo può segnalare l'assenza di rete al demone che provvederà a verificarne la veridicità e ad attivare la modalità offline.

Il passaggio alla modalità offline e il funzionamento della cattura delle immagini in questa modalità è mostrato in figura 4.15. In modalità offline il demone interroga più rapidamente la rete. Quando la connessione viene ristabilita, il demone riattiva la modalità online dell'applicativo.

Il ritorno in modalità online e la spedizione di un'immagine al server è spiegato in figura 4.16. Al ritorno in modalità online, l'applicativo come prima cosa vuota la coda di immagini compiendo quelle azioni che prima non poteva compiere: verificare la validità del codice RFID, recuperare i dati di crop, inviare l'immagine tagliata al server.

4.3.3.7 Comunicazione con il demone

La comunicazione con il demone di sistema avviene tramite un *signal handler* legato ad un oggetto di tipo `SignalModule`. Ogni segnale proveniente dal demone viene intercettato e gestito dal *signal handler* che richiama dei metodi dell'oggetto di tipo `SignalModule` che controllano il sistema. Viceversa, quando un evento arriva dal sistema, la cascata di segnali relativa arriva sino a questo oggetto che si preoccupa di convertire i segnali e propagarli verso il demone.

Allo stato attuale del prototipo, i segnali gestiti dal *signal handler* sono limitati ad eventi relativi ai GPIO (bottoni), commutazione alla modalità online o offline e alla ricezione di richieste di `acknowledge`, usate per controllare lo stato relativo dei due software.

4.3.3.8 Variabili globali e configurazione

L'applicativo sviluppato utilizza due file di configurazione: uno per l'applicativo, listato 4.7, e uno per i suoni, listato 4.8. Nei due listati sono impostati i valori di difetto.

Per cambiare un parametro nel file di configurazione basta modificare la stringa alla destra dell'uguale `=`. In caso di assenza di un parametro, verrà utilizzato quello impostato di difetto.

```
# file and directory configuration
log_name=          /opt/stouch/log/STOUCH_LOG
sound_dir=          /opt/stouch/sound/
crop_dir=           /opt/stouch/crop/
img_dir=            /opt/stouch/img/
ocv_prog=           /opt/stouch/cropper

# server informations
server_name=        http://www.tv-surf.com/3D-Enter/servlet/P13_19_CommandXML
server_ip=           62.2.177.143
server_port=         62001

# time in milliseconds
rfid_not_found_time= 3000
# time in milliseconds
inventory_time=     1200
# how many inventory scans before log to serve
scan_to_log=         300
# take photo time
take_photo=          8000
```

Listato 4.7: File di configurazione per l'applicativo (stouch.conf).

```
# sounds configuration file
powerup=          /opt/stouch/sound/powerup.mp3
powerdown=         /opt/stouch/sound/powerdown.mp3
stouch=           /opt/stouch/sound/stouch.mp3
error=            /opt/stouch/sound/error.mp3
connectionerror=  /opt/stouch/sound/connectionerror.mp3
send=              /opt/stouch/sound/send.mp3
timer=             /opt/stouch/sound/timer.mp3
timerclick=        /opt/stouch/sound/timerclick.mp3
wait=              /opt/stouch/sound/wait.mp3
voice_active=      /opt/stouch/sound/voice_active.mp3
voice_countdown=   /opt/stouch/sound/voice_countdown.mp3
voice_err_bracc=   /opt/stouch/sound/voice_err_bracc.mp3
voice_err_connection= /opt/stouch/sound/voice_err_connection.mp3
voice_offline=     /opt/stouch/sound/voice_offline.mp3
voice_online=      /opt/stouch/sound/voice_online.mp3
voice_prepare=     /opt/stouch/sound/voice_prepare.mp3
```

Listato 4.8: File di configurazione per il comparto suoni (sound.conf).

4.3.3.9 Software esterni al dispositivo

Il software comunica con una serie di *servlet* sul server aziendale per la gestione dei tag RFID registrati e l'invio di immagini. Inoltre al sistema è stata aggiunta una semplice **applicazione Android** per l'amministrazione del dispositivo e la registrazione di nuovi tag RFID.

4.3.4 OpenCV: programma cropper

La libreria OpenCV è stata utilizzata per la necessità futura di compiere il riconoscimento di oggetti e caratteristiche nelle immagini o nei video acquisiti. Nell'appendice D viene spiegato come cross-compilare questa libreria.

Il programma sviluppato consente di ritagliare l'immagine basandosi su due punti: i due angoli opposti di un rettangolo. Le immagini tagliate sono poi inviate al server. Sebbene molto semplice questa applicazione si basa interamente sulle librerie OpenCV e consente un ulteriore sviluppo sfruttando appieno la ben nota potenza di queste librerie.

Dato che questo programma è esterno all'applicativo sviluppato può essere facilmente sostituito mantenendo il numero di parametri utilizzati, ovvero quattro stringhe. Ad esempio è possibile caricare un programma che aggiunga l'effetto seppia alle immagini scattate o che aggiunga del testo o effetti di sfocatura. Sostituendo il programma **cropper** e adattando il suo valore nel file di configurazione, il software applicativo caricherà un nuovo tipo di immagine sul server.

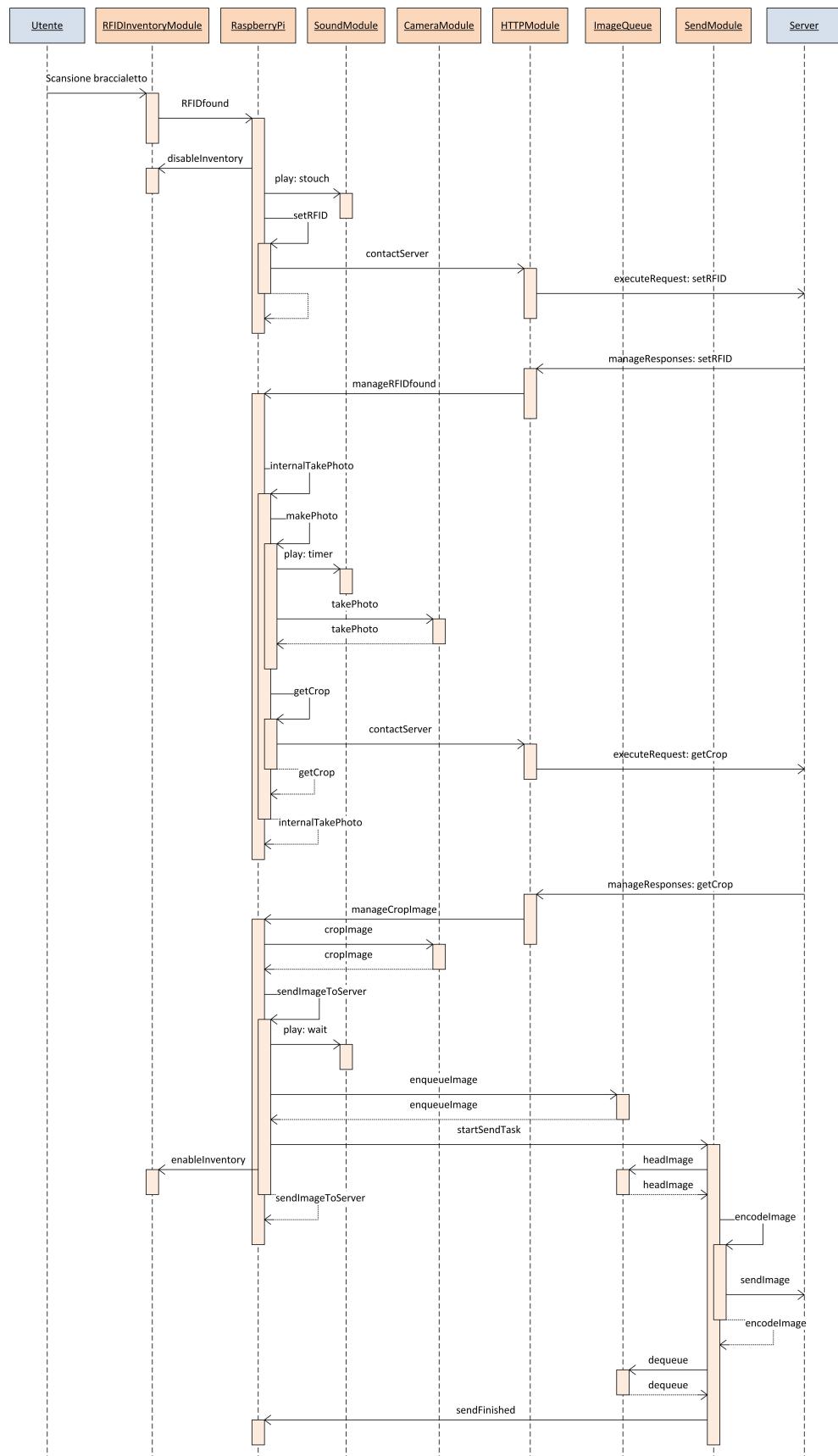


Figura 4.13: Diagramma di sequenza per un braccialetto trovato.

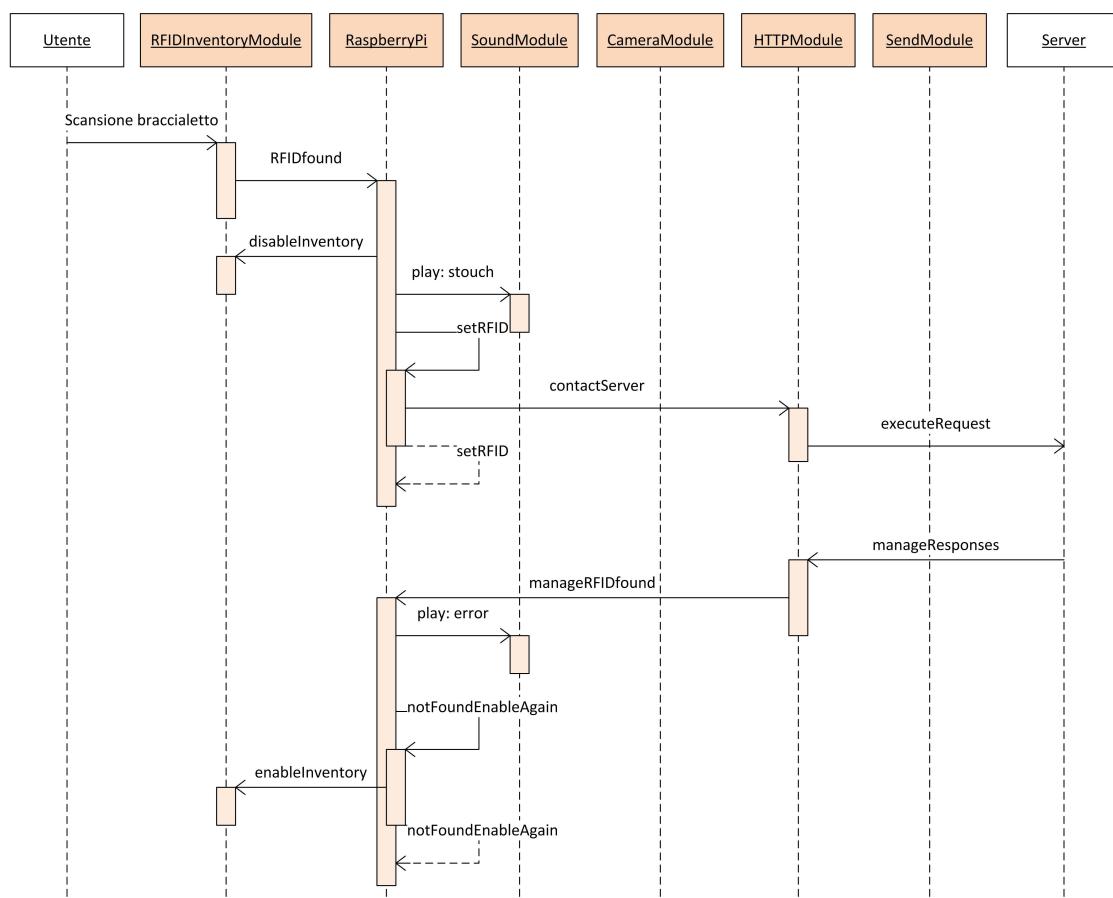


Figura 4.14: Diagramma di sequenza per un braccialetto non trovato.

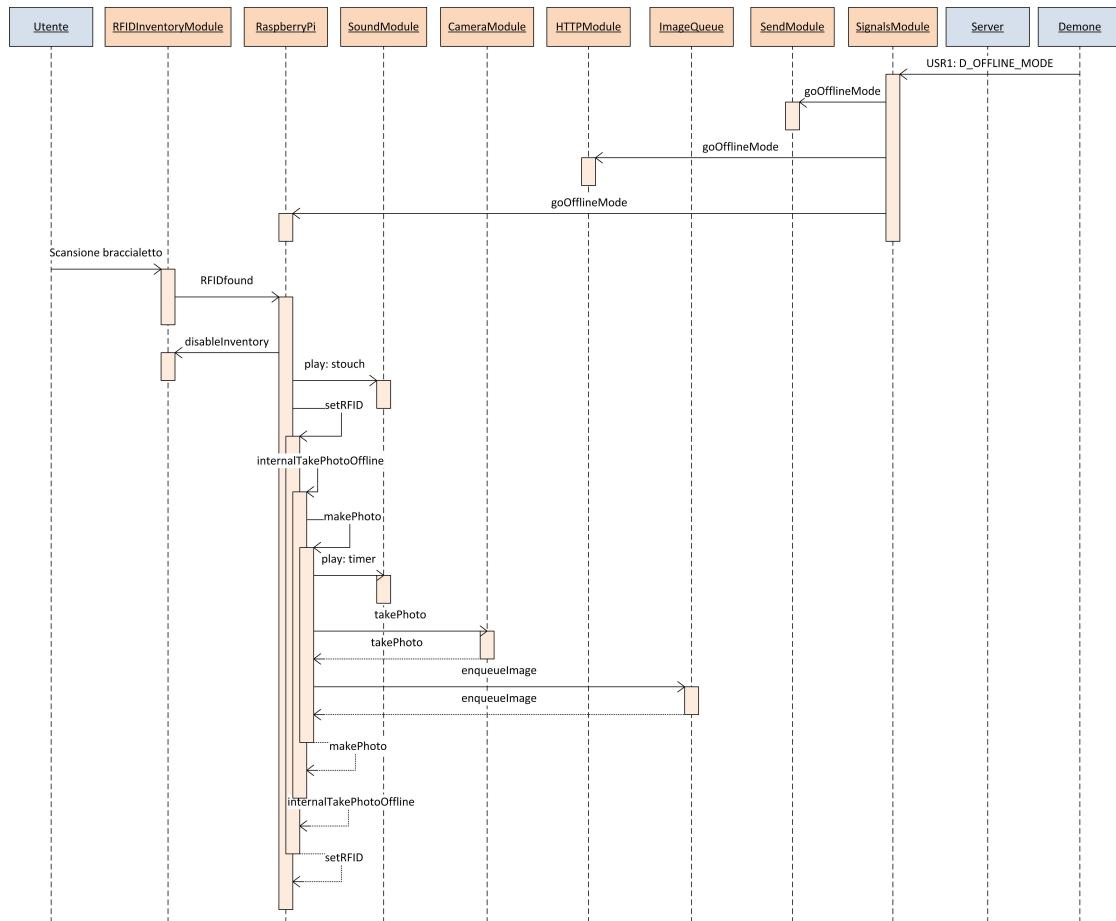


Figura 4.15: Diagramma di sequenza del passaggio a modalità offline.

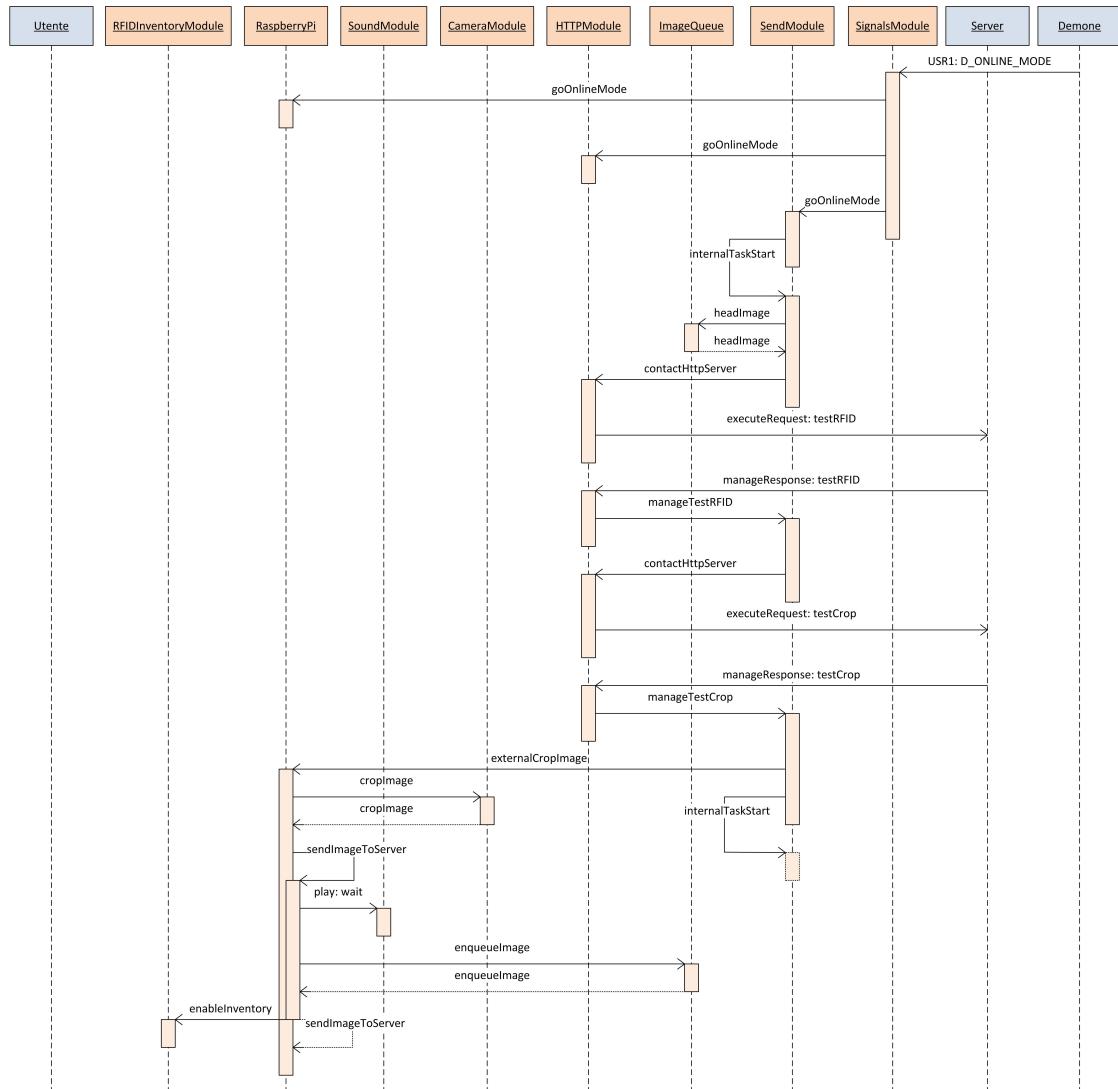


Figura 4.16: Diagramma di sequenza del ritorno in modalità online.

Capitolo 5

Test e risultati

5.1 Test compiuti

La serie di test eseguita è stata compiuta per verificare la funzionalità delle varie componenti hardware con il software. I test iniziali sono stati pensati per studiare il funzionamento dell'hardware, imparare ad interfacciarsi e controllarlo e scovare i primi problemi da risolvere durante il lavoro di sviluppo.

Nella sezione 5.1 sono elencati e descritti i test effettuati sull'hardware.

5.1.1 Test sull'hardware

Tabella 5.1: Test eseguiti sull'hardware.

Hardware	Descrizione test	Risultati
Dongle Bluetooth	Collegato il dongle bluetooth direttamente al Raspberry Pi via USB. È stata testata la scansione di dispositivi, l'accoppiamento manuale e il trasferimento di file.	Inizialmente è risultato impossibile utilizzare il dongle. Dopo alcune ricerche, sono stati aggiunti dei pacchetti necessari alla comunicazione (bluez, blueman) ed è stato rimosso il client grafico per la gestione del bluetooth dall'avvio automatico.

Tabella 5.1: Test eseguiti sull'hardware.

Hardware	Descrizione test	Risultati
Fotocamera	Uso della fotocamera tramite i programmi di sistema. Cattura di immagini, video e test sulla qualità.	Utilizzando il comando <code>raspistill</code> è stato possibile scattare delle foto. Le dimensioni sono di 2592 x 1944 pixel. La camera impiega un po' ad adattarsi alla luce prima di mettere a fuoco. Durante l'acquisizione l'angolo di visuale è di circa 45° in 4:3, al momento dello scatto 60° 16:9. Non sono stati catturati dei video.
Pulsanti arcade	Collegare i pulsanti in serie e collegarli ai GPIO del Raspberry Pi. Eseguire un comando alla pressione.	I pulsanti per funzionare devono avere un collegamento a tre pin distinti: un pin a <i>ground</i> , un pin ad un polo positivo, e un pin in input. I test hanno evidenziato come i pulsanti siano sia molto reattivi ma anche molto imprecisi. La linea di ingresso dei GPIO risulta essere soggetta a disturbi molto forti che provocano delle transizioni di livello logico in realtà inesistenti. Nonostante i tentativi software per ridurre il numero di transizioni acquisite, il problema non è stato del tutto risolto ed è auspicabile l'applicazione di un filtro elettronico sul circuito e l'adozione di altre tecniche software per migliorare l'acquisizione. I test sono stati compiuti con un solo pulsante collegato e con quattro in serie. Non sono state notate differenze.

Tabella 5.1: Test eseguiti sull'hardware.

Hardware	Descrizione test	Risultati
Audio	Riprodurre un file audio MP3.	Prima di riprodurre i suoni occorre caricare il modulo kernel che li gestisce con il comando <code>sudo modprobe snd_bcm2835</code> . In seguito occorre commutare l'uscita audio da HDMI al jack mediante l'utility amixer : <code>amixer cset numid=3 1</code> . Infine utilizzando i programmi <code>mpg123</code> e <code>mpg321</code> è stato possibile riprodurre qualsiasi suoni in MP3.
RFID	Utilizzare le librerie ufficiali di CAEN SpA. per interrogare il modulo. Verificare potenza e portata del segnale.	Le librerie ufficiali non sono utilizzabili su Linux. È stata creata una libreria per supportare il protocollo easy2read via seriale. La portata dell'antenna compresa nel pacchetto è molto limitata: . La potenza del segnale varia tra i 8mW e i 200mW. Gli effetti della potenza sulla portata del segnale sono minimi.
Rete Ethernet	Verificare il funzionamento dell'Ethernet con DHCP e senza.	La rete funziona correttamente, sia con DHCP che con IP statico.
Rete Wireless	Verificare il funzionamento della rete Wireless con il modulo WiFi Linksys con DHCP e senza.	L'antenna si connette automaticamente a reti aperte senza alcun problema, ma non effettua automaticamente richieste DHCP. Per funzionare è stato utilizzato un <i>workaround</i> tramite crontab che ad intervalli regolari lancia il programma dhclient wlan0 per l'acquisizione di un IP in caso non fosse ancora stato assegnato.

Tabella 5.1: Test eseguiti sull'hardware.

Hardware	Descrizione test	Risultati
Batteria e alimentazione	Verificare che la batteria sia ricaricabile, la sua durata e il sistema di alimentazione.	Il sistema di alimentazione a 5V con 2000mA forniti dall'alimentatore sul hub USB funziona. La batteria non viene ricaricata se è connessa contemporaneamente in input ed output: o tensione e corrente sono insufficienti alla ricarica della batteria, oppure più semplicemente la batteria non può essere caricata se collegata anche in output. Collegandola solamente in input la batteria si carica tranquillamente. La durata complessiva è di circa 1 ora. Una possibile soluzione consiste nel cambiare batteria.
LED	Accendere i LED ed eseguire una sequenza di colori.	Non testati. L'alimentazione attuale non è adatta.

5.2 Funzionalità

5.2.1 Test funzionali

Lo studio realizzato dal Dacd prevedeva che l'utente premesse il dispositivo per scattare la foto. Durante i test con il modulo RFID è stato notato come la distanza massima di scansione del modulo RFID fosse di circa un metro nel caso di tag RFID poggiati su superfici o comunque non in mano, mentre nel caso dei braccialetti al polso scendesse ad un desolante cinque centimetri.

Dopo aver contattato l'azienda produttrice del modulo RFID e appurata la necessità di una nuova antenna troppo grande per il prototipo, si è scelto di non utilizzare la pressione di alcun pulsante per far scattare l'immagine. Al contrario, è lo stesso modulo RFID a decidere quando scattare l'immagine in base all'acquisizione di un tag RFID valido.

5.2.2 Test sul campo

Per i test sul campo con le reti pubbliche della città di Lugano è stato creato un account VIP alla rete ed è stato registrato il MAC address del dispositivo con successo. Ad oggi non è ancora chiaro se l'associazione tra mac address e IP avvenga in automatico o se occorra effettuare un login anche con account VIP.

5.2.2.1 Connessione alla rete Wireless di Lugano Cornaredo

Durante i test effettuati allo stadio di Lugano Cornaredo il dispositivo si è connesso tranquillamente alla rete con un buon segnale. I test sono avvenuti nella zona stampa. Nonostante un forte temporale, il dispositivo ha funzionato correttamente.

5.2.2.2 Connessione alla rete Wireless di WiFi Lugano

Il dispositivo deve autonomamente cercare e acquisire l'accesso ad una rete wireless. La rete cablata è utilizzata unicamente in fase di sviluppo.

Dopo i primi test sul campo si è reso necessario creare degli script in esecuzione automatica per l'acquisizione degli indirizzi IP tramite DHCP. I test effettuati allo stadio di Lugano Cornaredo hanno evidenziato come il dispositivo si connettesse alla rete senza effettuare una richiesta DHCP. La richiesta è stata inserita con uno script bash utilizzante il comando **dhclient wlan0**.

Lo stesso non si può dire per i problemi sollevati dai test con la rete wireless presente al Parco Ciani. In nessun modo è stato possibile stabilire una connessione stabile con la rete Wireless di WiFi Lugano durante i test davanti alla darsena al Parco Ciani, nella zona destinata allo stand *Park & Read*. Nonostante il riposizionamento di alcune antenne, il segnale della rete risulta spesso insufficiente per qualsiasi tipo di comunicazione sia con il dispositivo, sia con laptop, tablet e smartphone. In un primo momento si sospettava che fosse necessario un login via browser ad ogni connessione, in realtà è stato confermato dai tecnici della Città di Lugano che la rete wireless pubblica è di qualità scadente. La soluzione per questo problema è stato l'acquisto di un bridge 3G-WiFi grazie al quale il dispositivo riesce a collegarsi alla rete.

Una soluzione alternativa potrebbe essere quella di sostituire l'antenna WiFi del dispositivo con un modulo 3G/GPRS in modo da poter collegarsi alla rete da praticamente qualunque posto. Tuttavia, dato l'uso ipotizzato per questo dispositivo, si veda l'Allegato 1, è una soluzione alquanto costosa.

Il dispositivo è stato configurato in modo da connettersi in maniera automatica a reti aperte senza crittografia, come le reti utilizzate dalla Città di Lugano. Gli SSID delle reti attualmente configurate e a cui il dispositivo è in grado di connettersi autonomamente, sono le seguenti:

- WiFi Lugano (utilizzata nella zona del Lungo Lago, del centro e nel Parco Ciani)



Figura 5.1: Il dispositivo in posizione nella zona Park & Read del Parco Ciani a Lugano.

- Wifi Lugano Cornaredo (utilizzata allo stadio di Lugano Cornaredo)
- RaspberryNetwork (rete di supporto per test e sviluppo)

Nella fase finale del progetto è stato impostato un IP fisso, 192.168.0.150/24, sulla porta Ethernet.

5.2.3 Risposta dell'utente al dispositivo

Nelle ultime settimane di agosto 2013 è stata installato un prototipo completamente funzionante nell'area Park & Read del Parco Ciani della Città di Lugano. Senza comunicazione particolare, nei primi 14 giorni, l'apparecchio è stato utilizzato da una media di 7 persone al giorno con rispettivamente 12 foto giornaliere, 34 visualizzazioni su Internet giornaliere, generando nel contempo 27 condivisioni su Facebook.

Durante la prima settimana di test sul campo il dispositivo avvisava mediante semplici suoni gli eventi. Molti utenti sono rimasti spaesati davanti a questi suoni in quanto, anche se familiari, non suggerivano molto. Si è quindi deciso di cambiare alcuni suoni con delle semplici frasi registrate in precedenza.

Considerando che i file audio sono esterni al programma, è bastato sostituire i suoni con le voci.

5.3 Difetti e limitazioni

Il prototipo funzionale è stato costruito con lo scopo primario di eseguire i compiti per cui è stato progettato. Tuttavia, ci sono alcune piccoli difetti e limitazioni che dipendono sia dal software utilizzato che dall'hardware

5.3.1 Parte software

5.3.1.1 Passaggio in modalità online

Le due modalità online/offline sono state ideate per sopperire alla continua assenza di rete. Con il ritorno in modalità online il dispositivo dovrebbe inviare le foto scattate durante la modalità offline al server. Durante i test in laboratorio con poche immagini in coda, il dispositivo inviava le immagini al server. Durante i test sul campo, però, non è stata sperimentata questa capacità unita alla problematica della mancata registrazione degli utenti, di conseguenza questa sua funzione è da ritenersi ancora in fase sperimentale.

5.3.1.2 Rallentamenti e crash di sistema

Durante lo sviluppo si è potuto notare come a tratti la comunicazione via **ssh** tra un client e il dispositivo tenda a deteriorarsi nel corso del tempo. Non sono ancora note le cause del



Figura 5.2: Utilizzo del dispositivo da parte di un utente.

deterioramento, anche perché è un evento che accade praticamente ad ogni connessione ma di cui non si è ancora capita la causa.

Durante lo sviluppo, il sistema operativo è andato parecchie volte in *Kernel Panic* senza apparente motivo. Le indagini svolte sul problema, l'analisi dei log unita all'irriproducibilità del problema non hanno portato ad alcuna soluzione. Rimuovendo i collegamenti soggetti a disturbo, nello specifico l'alimentazione a batteria e i pulsanti, sembra che il problema si sia andato migliorato.

5.3.1.3 Gestione di rete

La gestione di rete è soggetta ad un *workaround* globale. Per far funzionare il dispositivo in presenza delle reti pubbliche è nata la necessità di utilizzare script bash e lancio programmato di script per l'acquisizione di un DHCP. Questa è solo una soluzione provvisoria che deve essere risolta in collaborazione con chi offre la rete WiFi.

Inoltre, le reti a cui il dispositivo può collegarsi sono limitate e non contemplano reti protette da password o da account.

5.3.2 Parte hardware

Per quanto riguarda i problemi legati all'hardware, il prototipo soffre per la maggior parte di disturbi e interferenze elettriche. Il dispositivo è stato assemblato con componenti *off-the-shelf*: non ci sono cavi isolati o sistemi anti-interferenze. L'idea era quella di costruire un prototipo funzionale e funzionante nel tempo a disposizione in modo da poter compiere i primi test ed effettuare le prime scelte riguardanti l'hardware e il software.

5.3.2.1 Bluetooth

L'utilizzo dell'antenna Bluetooth era pensato per permettere all'applicazione Android di amministrazione di comunicare direttamente con il dispositivo e gestirne il funzionamento. In un primo momento, un conflitto tra servizi nel sistema operativo che utilizzano il Bluetooth in modo concorrente ha ostacolato notevolmente l'utilizzo dell'antenna. Solamente in un secondo momento il conflitto è stato risolto rimuovendo il servizio legato all'interfaccia grafica, non utilizzata, e l'antenna è tornata a funzionare correttamente.

In futuro si pensa di ripristinare il funzionamento del Bluetooth. Si veda il capitolo 6 per ulteriori dettagli.

5.3.2.2 Interferenze

Il difetto più fastidioso è collegabile ad un'interferenza tra i cavi che collegano il Raspberry Pi allo speaker audio. Ad ogni scansione alla ricerca di un tag RFID, l'interferenza genera un tick metallico che viene amplificato e riprodotto dallo speaker. Aggiungendo una messa a terra ai cavi dello speaker risolve il problema. Occorre tuttavia isolare completamente

ogni singolo cavo in quanto anche un'interferenza sui GPIO produceva riavvii indesiderati e addirittura kernel panic nel sistema.

5.3.2.3 Alimentazione

Il circuito di alimentazione con gestione del passaggio da batteria ad alimentazione diretta e viceversa è stato rimosso. Il filo di collegamento era soggetto a troppi disturbi ed interferenze e causava problemi con il corretto funzionamento del dispositivo ed è stato tolto.

I problemi sono nati dal fatto che il circuito costruito, o la batteria stessa, non riusciva a caricare completamente la batteria durante la fase di alimentazione, problema forse dato dall'amperaggio dell'alimentatore o dal tipo di batteria utilizzato.

Allo stato attuale del prototipo, il dispositivo può essere alimentato a corrente oppure a batteria, mediante una presa USB esterna.

Capitolo 6

Conclusioni

6.1 Raggiungimento degli obiettivi

Gli obiettivi si possono ritenere raggiunti, con l'eccezione dell'interazione dell'utente tramite pulsanti che non è stata effettuata a causa dei problemi emersi in fase di test, cioè le limitazioni hardware e l'incompatibilità con il lettore RFID. Tuttavia, i pulsanti sono comunque gestiti a livello software, ma sono stati disattivati.

Il sistema è in grado di associare qualsiasi tag RFID ad un ID univoco registrato su un server. A questo processo è stata associata la possibilità di scattare una fotografia e, attraverso il database sul server, di associarvi ulteriori informazioni correlate al tag RFID come, ad esempio l'indirizzo email.

Sebbene quanto descritto rappresenti la funzionalità basilare del sistema, essa si presta già a numerose applicazioni che esulano dagli obiettivi del progetto. Ad esempio nel campo della logistica esiste la possibilità di catturare l'indirizzo stampato su di un pacco e associare l'immagine ad un id preciso tramite il tag RFID. Oppure, in processi di produzione, è possibile usare la telecamera e le librerie OpenCV per il riconoscimento di oggetti, ognuno dei quali dotato di un RFID di riferimento e confermarne la qualità, il passaggio da un determinato settore dell'impianto, ...

La struttura del software è inoltre flessibile e adattabile in maniera semplice ad altre operazioni, come l'analisi video o il riconoscimento di caratteristiche in una immagine. Inoltre scegliendo l'implementazione modulare con un sistema Linux con librerie Qt e standard POSIX, esiste la possibilità di portare l'intero sistema su di piattaforme diverse dal Raspberry Pi in modo relativamente semplice intervenendo solo laddove è necessario, ad esempio nel demone e nel *cropper*, lasciando intatta l'intera struttura applicativa.

6.2 Sviluppi futuri

6.2.1 Comunicazione bidirezionale con server

Allo stato conclusivo dei lavori, per comunicare con il dispositivo occorre essere nella medesima rete locale. La comunicazione avviene tramite un client **ssh** con il quale è possibile avere un controllo completo del dispositivo.

Un'idea interessante sarebbe quella di poter mandare comandi al dispositivo da remoto, in particolare dal server. Attualmente il dispositivo comunica con il server con un socket TCP/IP, ma non ascolta ciò che arriva dall'altro capo della connessione. D'altro canto il server riceve solamente dati dal socket, senza comunicare alcunché al dispositivo.

Potrebbe risultare molto utile completare questo scambio di informazioni. Se in un futuro i dispositivi creati saranno interconnessi con il server in maniera bidirezionale, un semplice protocollo di comunicazione per agire da remoto sui dispositivi potrebbe risultare estremamente efficace per modificare parametri di funzionamento, reperire i log o addirittura per aggiornare intere parti del sistema.

6.2.2 Aggiornamenti via server

Legata in qualche modo all'idea espressa nella sezione precedente è la possibilità di aggiornamento automatico attraverso il server. Dato che che il sistema si trova ancora in una fase di sviluppo, il deployment di nuove versioni dei software avviene tramite un computer connesso alla medesima rete utilizzando il programma di accesso remoto **scp**, basato su **ssh**. Un semplice demone di sistema che ad intervalli regolari contatti il server per verificare la presenza di aggiornamenti e in seguito aggiorni la versione del software, diventerebbe non solo utile ma anche necessario quando il numero di dispositivi aumenta.

6.2.3 Bluetooth

Un'idea iniziale prevedeva la creazione di un'applicazione per Android con la quale gli utenti potessero direttamente connettersi al dispositivo, identificarsi e scattare una foto, senza utilizzare il lettore RFID. Questa funzione non è però mai stata studiata a fondo o implementata, limitandosi ai test preliminari con il modulo in dotazione.

La comunicazione via Bluetooth è sicuramente qualcosa da non sottovalutare. Potrebbe infatti dimostrarsi un interessante mezzo per sincronizzare più dispositivi tra di loro secondo questo scenario: al momento dello scatto la scena viene catturata in sincronia da più dispositivi e le immagini vengono inviate ad uno di essi che provvede a spedirle al server creando una simpatica cartolina con tutte le immagini.

Anche la configurazione potrebbe avvenire via Bluetooth. Attualmente i dati per il ritaglio dell'immagine sono registrati sul server, mentre quelli di configurazione su file locali. Potrebbe essere interessante disporre di un'applicazione per tablet o smartphone con cui controllare

il dispositivo, posizionarlo, ritagliare le zone scelte e addirittura orientarlo grazie a piccoli motori. Le possibilità sono davvero numerose ed interessanti.

Con i comandi Bluetooth si riuscirebbe anche a risolvere il problema dello spegnimento del dispositivo con un'apposita sequenza di comandi, o addirittura utilizzando un semplice telecomando predisposto per questo protocollo di comunicazione senza fili.

6.2.4 Comunicazioni di rete

Dati i numerosi problemi emersi con la rete WiFi della città di Lugano nel luogo preposto al test del dispositivo, è stato utilizzato un access point WiFi-3G esterno come soluzione temporanea. Recentemente sono stati sviluppati dei moduli 3G per il Raspberry Pi, che potrebbero risultare interessanti. Il lato negativo di questo tipo di comunicazione riguarda il costo relativamente elevato della connessione dati.

6.2.4.1 Voce

Durante i test sul campo si è notato come sia difficile fornire un'informazione efficace e precisa all'utente.

I suoni, in generale, necessitano di una certa amplificazione per evitare che il rumore dell'ambiente circostante o la distanza tra l'altoparlante e l'utente impedisca di udirli. L'utilizzo di semplici suoni può risultare inoltre incomprensibile per l'utente, mentre l'uso di voci necessita un supporto maggiore. Un'idea interessante nata assieme alle voci consisteva nell'introdurre una personalizzazione della voce sia per il messaggio che per la lingua dell'annuncio.

6.2.5 Hardware e LED

Nella fase attuale il prototipo iniziale è funzionante con l'hardware fornito, occorre però integrare tutto in un hardware migliore e più affidabile. Il prossimo passo di ingegnerizzazione deve dunque prevedere la costruzione di una nuova piattaforma, senza disturbi e interferenze, che risolva quindi i problemi emersi nella fase di sviluppo prototipale.

Un componente da migliorare sicuramente è il circuito di alimentazione: occorre utilizzare un'alimentazione a 12V con riduzione a 5V. In questo modo si potranno alimentare i LED e continuare ad alimentare tutti gli altri componenti.

L'idea della batteria di backup è ottima, ma occorre trovare una batteria ricaricabile migliore, adatta al sistema e verificare che il circuito di scambio e di carica funzioni correttamente.

6.2.6 OpenCV

Le librerie OpenCV sono pronte per essere utilizzate: sono funzionanti e compatibili con il cross-compilatore. Addirittura, allo stato attuale, basterebbe sostituire il programma **crop-per** per cambiare completamente il funzionamento del dispositivo.

Un'integrazione più corposa consiste però nel riuscire ad acquisire ed analizzare i frame di un filmato in modo che si possa, per esempio, estrapolare il volto di una persona o quando questa sorride.

Grazie alla ben nota versatilità e alla potenza offerta da OpenCV, le applicazioni con il riconoscimento di oggetti sono praticamente infinite.

6.2.7 Interfaccia grafica

Le librerie Qt, unite al linguaggio QML, offrono una potente architettura software per la creazione di interfacce grafiche. In alcuni disegni del prototipo si pensava anche a schermi da collegare al dispositivo se non addirittura ad una versione portatile del dispositivo. Queste librerie permettono di sviluppare un'interfaccia grafica elegante e performante per queste situazioni.

6.3 Passo di ingegnerizzazione per il prototipo 2

Con questo lavoro si conclude lo studio sul primo prototipo del dispositivo Stouch. Il prossimo passo di ingegnerizzazione, come già anticipato nelle sezioni precedenti, riguarderà sicuramente l'hardware: una nuova piattaforma migliorata per alimentare e collegare i moduli e un miglioramento della struttura meccanica. In seguito, una volta risolti i problemi di fondo, si potrà tornare a verificare il corretto funzionamento del sistema software.

6.4 Conclusioni personali

Per quanto riguarda la componente didattica di questo progetto di diploma, posso tranquillamente affermare di essere soddisfatto del lavoro svolto. Ho avuto modo di partecipare ad un percorso che porta un'idea a diventare qualcosa di materiale e funzionante, seguendone i passi fin dall'inizio. Un ringraziamento particolare all'Ing. Giacomo Poretti per la costante presenza e incredibile disponibilità durante tutto il progetto.

Lo sviluppo di questo software mi ha permesso di entrare in un mondo, quello dei sistemi *embedded* con architettura ARM di cui fanno parte i molto spesso blasonati tablet e smartphone, che di solito viene visto ad alto livello, dimenticando spesso tutto quello che al di sotto di numerosi strati di librerie, framework e servizi viene nascosto. La normalità, se così si può chiamare, prevede di sfruttare quello che l'ambiente offre per costruire il proprio software applicativo. Solitamente, lo spazio di lavoro possiede numerosi paletti e recinti creati dai suoi sviluppatori dentro i quali occorre rimanere per riuscire a produrre il proprio software. Tuttavia, quando questi paletti non ci sono e siamo noi a sceglierli e imporceli, la libertà collide con la possibilità di creare praticamente qualsiasi cosa, come la vogliamo.

Questo progetto si è rivelato un'ottima occasione per entrare in quei momenti di cui non si parla mai. Come studente di ingegneria informatica, parlare di hardware è qualche cosa di

distante, spesso inteso come lavoro altrui a cui ci si deve affidare e su cui non si ha controllo. Invece, questo progetto dimostra che non solo è l'hardware a decidere il successo del software, ma è anche che quest'ultimo può porre limitazioni o requisiti specifici all'hardware. In fin dei conti, è sempre un gioco di dare e ricevere, di provare e riprovare fin quando il silicio prende a funzionare seguendo le rigide linee di un codice.

Elenco degli allegati

Allegato 1 *The Social Touch a new Social Media Marketing Engine* - Descrizione funzionale,

Allegato 2 *Stouch - System architecture, scenarios and design of the Stouch Point*, Interaction Design Lab, SUPSI-DACD

Allegato 3 *CAEN UHF RFID READERS COMMUNICATION PROTOCOL*, Technical Information Manual, Revision n. 15, 05/10/2012

Appendice A

Configurazione di un cross-compiler per il Raspberry Pi

Questo tutorial è stato scritto seguendo la guida del sito Hertaville [17] e spiega come installare e configurare correttamente un cross-compiler in modo che lo si possa utilizzare per compilare software per il processore ARM del Raspberry Pi da un computer più potente, permettendo nel contempo il debug del programma da remoto.

La compilazione di un cross-compiler e delle sue librerie è uno dei procedimenti più complessi nel campo dell'informatica e può richiedere molto tempo allo sviluppatore. Il fatto di potersi basare su un cross-compiler già compilato e che deve essere solo configurato ed installato permette di risparmiare moltissimo tempo in fase di sviluppo. Tuttavia, il procedimento di installazione deve essere eseguito correttamente ed è essenziale che ogni passaggio debba andare a buon fine.

A.1 Definizioni

Il computer dove si sviluppa il codice e dove viene installato il cross-compiler è definito *host*.

Il dispositivo (o computer) dove si esegue il codice cross-compilato è definito *target*.

Il processo di compilare del codice sull'*host* in modo che questo possa essere eseguito sul *target* è definito cross-compilazione. Il cross-compiler produce sull'*host* un programma eseguibile solo sul *target*, poiché contiene le istruzioni macchina nel formato del *target* (che generalmente ha un'architettura diversa da quella dell'*host*).

Tutte le librerie che vengono usate in formato binario nel processo di link durante la cross-compilazione sull'*host*, a partire da quelle standard fino a quelle particolari per una data applicazione, devono essere preventivamente cross-compilate e devono necessariamente risiedere sul computer *host* per permetterne l'uso durante la compilazione: non sono compatibili con le normali librerie del sistema *host*.

Le librerie di runtime a cui l'applicazione fa riferimento durante l'esecuzione devono risiedere sul *target* (e non necessariamente sull'*host*) e devono essere compilate per l'architettura del *target*.

A.2 Preparazione

Se sull'*host* si utilizza un sistema operativo a 64bit, installare le librerie per programmi a 32bit in quanto l'architettura ARM è a 32bit.

```
sudo apt-get install ia32-libs
```

Installare inoltre, se non se ne dispone già, il software di controllo di versione **git**.

```
sudo apt-get install git
```

A.3 Installazione del cross-compiler

Creare una cartella di lavoro dedicata.

```
mkdir ~/raspberrypi
```

Scaricare il contenuto delrepository di GIT con i cross-compiler dedicati a Raspberry Pi.

```
cd raspberrypi  
sudo git clone git://github.com/raspberrypi/tools.git
```

Entrare nella cartella dove sono presenti i compilatori.

```
cd ~/raspberrypi/tools/arm-bcm2708
```

Tra i tre compilatori presenti, verrà utilizzato **gcc-linaro-arm-linux-gnueabihf-raspbian**.

Aggiungere al PATH di sistema la directory con i compilatori necessari per C e C++. Modificare i file **./.bashrc** e **./.profile** aggiungendo il seguente comando alla fine del file.

```
export PATH=$PATH:$HOME/raspberrypi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-  
raspbian/bin
```

I compilatori che verranno usati sono:

- arm-linux-gnueabihf-gcc;
- arm-linux-gnueabihf-g++.

A.4 Verifica del funzionamento corretto dei compilatori

A.4.1 Compilare il linguaggio C

Scrivere un file di test simile al seguente main.c.

```
#include "stdio.h"

int main(void)
{
    printf("C: Hello World!");
    return 0;
}
```

Questo semplice test coinvolge nel processo di link la libreria standard `stdio` verificandone l'esistenza.

Da una shell lanciare il seguente comando:

```
arm-linux-gnueabihf-gcc main.c -o cmain
```

A.4.2 Compilare il linguaggio C++

Scrivere un file di test simile al seguente main.cpp.

```
#include <iostream>

using namespace std;

int main(void)
{
    cout << "C++: Hello World!" << endl;
    return 0;
}
```

Da una shell lanciare il seguente comando:

```
arm-linux-gnueabihf-g++ main.cpp -o cppmain
```

A.4.3 Deployment

Come detto, gli eseguibili prodotti non possono essere lanciati sulla macchina *host* che li ha compilati, occorre prima eseguire il **deployment**. Per questa operazione utilizzeremo i comandi **scp** e **ssh**. Assicurarsi che il Raspberry Pi sia collegato alla rete con un ip noto, ad esempio **192.168.0.150**.

Scaricare quindi i programmi compilati su Raspberry Pi.

```
scp cmain pi@192.168.0.150:~/  
scp cppmain pi@192.168.0.150:~/
```

Eseguire il login sul Raspberry Pi.

```
ssh pi@192.168.0.150
```

Il nome utente standard è: `pi`, mentre la password è `raspberry`. Lanciare i due programmi.

```
./cmain  
./cppmain
```

E verificarne il corretto funzionamento.

Appendice B

Compilare Qt5 per Raspberry Pi

Questo tutorial è stato scritto seguendo la guida ufficiale di Qt Project [10] e correggendo i diversi errori che sono occorsi durante la procedura, dovuti ad un problema di conflitti nel versionamento di alcuni file sorgente sul repository **git**.

B.1 Preparazione

Si presuppone che sul sistema sia già presente un **cross-compiler** e **git**. L'appendice A spiega come installarne e configurarne uno.

B.1.1 Directory di lavoro

La directory di lavoro principale è la medesima utilizzata per il cross-compiler A, ovvero `/raspberrypi`.

B.1.2 Immagine di sistema locale

Per lavorare in maniera rapida ed efficiente occorre disporre di un'immagine locale del sistema che andrà caricato su una scheda SD e utilizzato poi dal Raspberry Pi. Per questo tutorial viene utilizzata un'installazione di Raspbian [8] già configurata su Raspberry Pi con le librerie basilari già installate. L'immagine può essere copiata facilmente, una volta inserita la scheda SD all'interno del computer, con il comando **dd**.

```
sudo dd if=/dev/mmcblk0 of=~/raspberrypi/raspbian.img
```

Una volta creata l'immagine locale è necessario montarla. Questa immagine sarà la copia locale di quella che sarà caricata e utilizzata sul Raspberry Pi e andrà **sempre** tenuta montata durante lo sviluppo del software.

La procedura che segue installerà le librerie cross-compilate necessarie all'esecuzione dei programmi sviluppati sul computer *host* sull'immagine della scheda SD e di conseguenza

sul *target*.

La directory dove verrà sempre montata l'immagine di sistema è `/mnt/rasp-pi-rootfs`.

```
sudo mkdir /mnt/rasp-pi-rootfs
sudo mount -o loop,offset=62914560 ~/raspberrypi/raspbian.img /mnt/rasp-pi-rootfs
```

B.2 Installazione di Qt5 per Raspberry Pi

Spostarsi nella directory di lavoro.

```
cd ~/raspberrypi/
```

Clonare il repository git contenente i **cross-compile-tools** che serviranno durante la procedura di compilazione delle librerie Qt5.

```
git clone git://gitorious.org/cross-compile-tools/cross-compile-tools.git
```

Clonare ed inizializzare il repository git con i sorgenti di Qt5.

```
git clone git://gitorious.org/qt/qt5.git
cd qt5
./init-repository
```

L'operazione potrebbe richiedere molto tempo.

Seguendo la guida [10], in questo momento viene indicato di applicare una patch per un modulo di Qt5, **qtjsbackend**. Purtroppo questa operazione nell'ambiente di sviluppo utilizzato ha provocato un conflitto che è stato risolto solo durante la compilazione delle librerie. La risoluzione dei conflitti è avvenuta mantenendo la revisione **HEAD**. I comandi lanciati sono stati i seguenti:

```
cd ~/raspberrypi/qt5/qtjsbackend
git fetch https://codereview.qt-project.org/p/qt/qtjsbackend refs/changes/56/27256/4 &&
git cherry-pick FETCH_HEAD
```

L'errore rinvenuto è il seguente:

```
remote: Counting objects: 34, done
remote: Finding sources: 100% (20/20)
remote: Total 20 (delta 9), reused 19 (delta 9)
Unpacking objects: 100% (20/20), done.
Da https://codereview.qt-project.org/p/qt/qtjsbackend
 * branch                  refs/changes/56/27256/4 -> FETCH_HEAD
```

```

error: non e' stato possibile applicare bce20ee... [V8] Add support for using armv6 vfp2
      instructions
suggerimento: dopo aver risolto i conflitti, segna i path corretti
suggerimento: con 'git add <path>' o 'git rm <path>' ed eseguire
suggerimento: il commit del risultato con 'git commit'

```

Dato l'esito comunque positivo della procedura, anche in presenza di questi errori risolti in seguito, si consiglia di lanciare comunque questi comandi.

B.2.1 Compilazione di Qt base

Eseguire lo script per la generazione dei link simbolici.

```

cd ~/raspberrypi/cross-compile-tools
sudo ./fixQualifiedLibraryPaths /mnt/rasp-pi-rootfs/ ~/raspberrypi/tools/arm-bcm2708/gcc-
-linaro-arm-linux-gnueabihf-raspbian/bin/arm-linux-gnueabihf-gcc

```

Da notare l'utilizzo del cross compiler locale installato in precedenza (A).

Procedere con la configurazione, compilazione ed installazione delle librerie.

```

cd ~/raspberrypi/qt5/qtbase
./configure -opengl es2 -device linux-rasp-pi-g++ -device-option CROSS_COMPILE=~/
raspberrypi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/arm-linux-
gnueabihf- -sysroot /mnt/rasp-pi-rootfs -opensource -confirm-license -optimized-
qmake -reduce-relocations -reduce-exports -release -make libs -prefix ~/raspberrypi/
qt5pi -no-pch
make
make install

```

Anche se la guida ufficiale utilizza il comando make -j 4, che divide su più processi paralleli la compilazione (in questo caso 4), si sconsiglia questa operazione. Molte librerie dipendono dalla compilazione di librerie precedenti. Se la compilazione non è perfettamente parallela e sincronizzata, alcuni moduli possono fallire con la necessità di dover rilanciare il processo di installazione.

B.2.2 Compilare e installare i moduli aggiuntivi

I moduli aggiuntivi presenti nel repository sono i seguenti e vanno compilati nell'ordine qui indicato:

- qimageformats
- qtsvg

- qtjsbackend
- qtscript
- qtxmlpatterns
- qtdeclarative
- qtsensors
- qt3d (vuoto)
- qtgraphicaleffects
- qtjsondb (vuoto)
- qtlocation (vuoto)
- qtdocgallery (vuoto)

La compilazione e l'installazione avviene nel seguente modo per ogni singolo modulo:

```
cd ~/raspberrypi/qt5/<nome modulo>
~/raspberrypi/qt5pi/bin/qmake .
make
sudo make install
```

Durante l'installazione si sono verificati degli errori di compilazione dovuti alla patch di git non effettuata correttamente. La soluzione è però molto semplice e consiste nel trovare i file contenenti i conflitti e risolverli mantenendo la revision **HEAD**. I file incriminati sono i seguenti indicati dagli errori durante il processo di compilazione.

B.3 Completamento dell'installazione

Ora che l'immagine del sistema è pronta occorre solo copiarla sulla scheda SD e inserirla nel dispositivo.

```
cd ~/raspberrypi/
sync; sudo umount /mnt/rasp-pi-rootfs
sudo dd bs=1M if=raspbian.img of=/dev/mmcblk0; sync
```

Ricordarsi di rimontare l'immagine prima di iniziare lo sviluppo di applicazioni. Ad ogni riavvio l'immagine viene smontata e va rimontata.

Appendice C

Configurare Qt Creator per l'utilizzo con Qt5 su Raspberry Pi

Questo tutorial è stato scritto seguendo la guida ufficiale di Qt Project [10].

C.1 Preparazione

Si presuppone che sul sistema sia già presente un **cross-compiler** e che sia presente un'immagine di sistema con Qt5 installate per Raspberry Pi. L'appendice A spiega come installarne e configurarne un cross-compiler, mentre l'appendice B spiega come compilare per Raspberry le librerie Qt5.

Per questo tutorial è stata utilizzata la versione **2.6.1** di QtCreator.

Assicurarsi che l'immagine locale del Raspberry Pi contenente le librerie compilate sia montata correttamente prima di procedere.

C.2 Configurazione di QtCreator

C.2.1 Installazione delle librerie locali per il sistema

Dato che le librerie Qt sono librerie portabili e *cross-platform*, è consigliabile installare una copia locale per il proprio sistema delle Qt5. In questo modo sarà possibile testare i software localmente per poi effettuare la cross-compilazione per Raspberry Pi.

Installare le librerie Qt5 e QtCreator da repository prima di proseguire.

C.2.2 Impostazione della Tool Chain

- Menu Tools > Options > Build & Run, nel menu laterale Compilers.
- Aggiungere un nuovo compiler con Add -> GCC.

- Chiamarlo con un nome caratteristico, come **ARM GCC**.
- Come *Compiler Path* inserire il percorso al cross-compiler (A), aggiustare il nome utente:

```
/home/<nome utente>/raspberrypi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian  
/bin/arm-linux-gnueabihf-g++
```

L'IDE potrà ora utilizzare il cross-compiler.

C.2.3 Impostazione delle librerie Qt

- Menu Tools > Options > Build & Run, nel menu laterale Qt Versions.
- Aggiungere una nuova libreria e scegliere il percorso del relativo **qmake**, aggiustare il nome utente:

```
/home/<nome utente>/raspberrypi/qt5pi/bin/qmake
```

Se le librerie non vengono rilevate, assicurarsi di aver montato correttamente l'immagine del sistema.

C.2.4 Configurazione del dispositivo Linux

- Menu Tools > Options > Devices.
- Aggiungere un nuovo dispositivo del tipo Generic Linux Device.
- Configurare il dispositivo con le proprie credenziali (di default: pi - raspberry) e l'indirizzo ip del dispositivo.
- Procedere con il test del dispositivo. In caso di esito negativo ricontrillare le credenziali e la configurazione di rete.

C.2.5 Configurazione dei Kit di sviluppo

- Menu Tools > Options > Build & Run nel menu laterale Kits
- Aggiungere un nuovo kit di sviluppo
- Name: **Raspberry Pi**
- Device type: **Generic Linux Device**
- Device: **Raspberry Pi** (quello creato al punto precedente)
- Compiler: **ARM GCC** (quello creato tre punti fa)

- Debugger (compreso nel cross-compiler A scaricato e configurato in precedenza):
~/raspberrypi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/arm-linux
-gnueabihf-gdb
- Qt Version: *Qt 5.1.1 (qt5pi) (B)

C.2.6 Creazione e configurazione di un progetto

Alla creazione di un nuovo progetto, scegliere come device di sviluppo solamente il Kit **Raspberry Pi**. In caso si vogliano fare prove locali, aggiungere anche il Kit di lavoro locale. Nel file *.pro* aggiungere le seguenti linee in modo che Qt Creator sappia dove deve depositare i file.

```
target.path = /home/pi/<cartella di destinazione su raspberry>
INSTALLS += target
```

Nella scheda del progetto, per il Kit *Raspberry Pi* aggiungere la seguente variabile di sistema:

```
LD_LIBRARY_PATH => ~/raspberrypi/qt5pi/lib
```

Il progetto è ora configurato per sviluppare e compilare nella macchina locale, eseguire il *deployment* sul Raspberry Pi, eseguire e debuggere l'applicazione sviluppata direttamente sul dispositivo remoto. Con questo sistema, volendo, è possibile sviluppare su un Raspberry Pi posto all'altro capo del mondo, basta che disponga di un indirizzo IP accessibile.

Appendice D

OpenCV per Raspberry Pi

Questo tutorial è stato scritto basandosi sulla guida di MitchTech [18], sulla guida ufficiale di OpenCV per compilare per ARM [19] e la guida di Mohamed Belhadj [20].

D.1 Preparazione delle librerie

Per prima cosa occorre installare dei pacchetti aggiuntivi sul Raspberry Pi richiesti durante la compilazione. A causa delle dipendenze, i due comandi vanno lanciati in successione e separatamente.

```
sudo apt-get install build-essential cmake cmake-qt-gui pkg-config libpng12-0 libpng12-dev libpng++-dev libpng3 libpnglite-dev zlib1g-dbg zlib1g zlib1g-dev pngtools libtiff4-dev libtiff4 libtiffxx0c2 libtiff-tools

sudo apt-get install libjpeg8 libjpeg8-dev libjpeg8-dbg libjpeg-progs ffmpeg libavcodec-dev libavcodec53 libavformat53 libavformat-dev libgstreamer0.10-0-dbg libgstreamer0.10-0-dev libgstreamer0.10-dev libxine1-ffmpeg libxine-dev libxine1-bin libunicap2-dev libdc1394-22-dev libdc1394-22 libdc1394-utils swig libv4l-0 libv4l-dev python-numpy libpython2.6 python-dev python2.6-dev libgtk2.0-dev pkg-config
```

Questi pacchetti vanno installati direttamente su Raspberry Pi attraverso i repository. Eseguire una nuova copia dell'immagine di sistema prima di proseguire, come spiegato in appendice A.

D.2 Cross-compilazione delle librerie

Scaricare i sorgenti di OpenCV e scompattarli in /raspberry/opencv. Nella stessa cartella creare il file di configurazione raspberry.cmake che verrà utilizzato dal comando cmake per generare un *makefile* configurato correttamente per la macchina.

Il contenuto del file, basato sull'installazione compiuta fino ad ora, è indicato di seguito. Da notare che la prima riga indica la destinazione completa del cross-compiler, occorre quindi modificare la cartella *home* utilizzata.

```
set(toolchain_dir /home/<nome utente>/raspberrypi/tools/arm-bcm2708/gcc-linaro-arm-linux
    -gnueabihf-raspbian)
set(toolchain_bin_dir ${toolchain_dir}/bin)
set(toolchain_libc_dir ${toolchain_dir}/arm-linux-gnueabihf/libc)
set(toolchain_inc_dir ${toolchain_libc_dir}/include)
set(toolchain_lib_dir ${toolchain_libc_dir}/usr/lib)

set(CMAKE_SYSTEM_NAME Linux CACHE INTERNAL "system name")
set(CMAKE_SYSTEM_PROCESSOR arm CACHE INTERNAL "processor")
set(CMAKE_C_COMPILER ${toolchain_bin_dir}/arm-linux-gnueabihf-gcc)
set(CMAKE_CXX_COMPILER ${toolchain_bin_dir}/arm-linux-gnueabihf-g++)
set(CMAKE_C_FLAGS "-isystem ${toolchain_inc_dir}" CACHE INTERNAL "c compiler flags")
set(CMAKE_CXX_FLAGS "-isystem ${toolchain_inc_dir}" CACHE INTERNAL "cxx compiler flags")

set(link_flags -L${toolchain_lib_dir})

set(CMAKE_EXE_LINKER_FLAGS ${link_flags} CACHE INTERNAL "exe link flags")
set(CMAKE_MODULE_LINKER_FLAGS ${link_flags} CACHE INTERNAL "module link flags")
set(CMAKE_SHARED_LINKER_FLAGS ${link_flags} CACHE INTERNAL "shared lnk flags")
set(CMAKE_FIND_ROOT_PATH ${toolchain_libc_dir} CACHE INTERNAL "cross root directory")
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH CACHE INTERNAL "")
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY CACHE INTERNAL "")
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY CACHE INTERNAL "")
```

Creare una directory di lavoro locale.

```
mkdir ~/raspberry/opencvpi
cd ~/raspberry/opencvpi
```

Creare la directory di destinazione nell'immagine locale del Raspberry Pi.

```
mkdir /mnt/rasp-pi-rootfs/opt/opencv
```

Creare il *makefile* con *cmake*. Assicurarsi di avere l'immagine del sistema del raspberry montata, ad esempio in */mnt/rasp-pi-rootfs*. Questa directory sarà il parametro *DCMAKE_INSTALL_PREFIX*.

```
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=/mnt/rasp-pi-rootfs/opt/opencv -
DCMAKE_TOOLCHAIN_FILE:PATH=../opencv/raspberrypi.cmake -DBUILD_EXAMPLES=OFF -
DBUILD_NEW_PYTHON_SUPPORT=OFF -DBUILD_PACKAGE=OFF -DBUILD_SHARED_LIBS=OFF -
```

```
DBUILD_TESTS=OFF -DWITH_1394=OFF -DWITH_FFMPEG=OFF -DWITH_GSTREAMER=OFF -DWITH_GTK=
OFF -DWITH_QT=OFF -DWITH_V4L=OFF -DWITH_PYTHON=OFF ../opencv
```

Compilare e installare.

```
make
sudo make install
```

Completiamo l'installazione caricando la nuova immagine di sistema sulla SD card:

```
cd ~/raspberrypi/
sync; sudo umount /mnt/rasp-pi-rootfs
sudo dd bs=1M if=raspbian.img of=/dev/mmcblk0; sync
```

D.3 Configurazione del progetto in QtCreator

Si assume che sul sistema sia già correttamente installato e configurato QtCreator C.

Nel progetto di Qt Creator aggiungere le seguenti linee:

```
INCLUDEPATH += /mnt/rasp-pi-rootfs/opt/opencv/include \
/mnt/rasp-pi-rootfs/opt/opencv/include/opencv

LIBS += -L/mnt/rasp-pi-rootfs/opt/opencv/lib \
-L/mnt/rasp-pi-rootfs/opt/opencv/share/OpenCV/3rdparty/lib/ \
-lopencv_contrib -lopencv_stitching -lopencv_nonfree -lopencv_superres -lopencv_ts - \
lopencv_videostab \
-lopencv_gpu -lopencv_legacy -lopencv_ml -lopencv_objdetect -lopencv_calib3d - \
lopencv_photo -lopencv_video \
-lopencv_features2d -lopencv_highgui -lIImImf -llibjasper -llibtiff -llibpng -llibjpeg - \
lopencv_flann \
-lopencv_imgproc -lopencv_core -lzlib -lrt
```


Appendice E

WiringPi per Raspberry Pi

WiringPi [13] è una libreria sviluppata da Gordon Henderson specifica per la gestione dei GPIO del Raspberry Pi.

A differenza delle altre librerie che devono essere cross-compilate, questa piccola libreria si può rapidamente installare nel sistema operativo.

E.1 Installazione

Sul Raspberry Pi, scaricare la libreria da <http://wiringpi.com/download-and-install/>. Estrarre il contenuto dell'archivio in una directory:

```
tar xfz wiringPi-02a3bd8.tar.gz
```

Entrare nella cartella appena decompressa e compilare la libreria:

```
cd wiringPi-02a3bd8  
./build
```

Una volta completata l'installazione occorre generare una nuova immagine di sistema locale, come spiegato nelle Appendici A, B e D.

E.2 Sistemazione dei link

Per far sì che la libreria sia utilizzabile dal cross-compiler occorre prima di tutto sistemare alcuni link simbolici partendo dall'immagine locale montata:

```
cd /mnt/rasp-pi-rootfs/usr/lib  
sudo mv libwiringPi.so libwiringPi.so.bkp  
sudo mv libwiringPiDev.so libwiringPiDev.so.bkp  
sudo ln -s ../../local/lib/libwiringPi.so.2.0 libwiringPi.so
```

```
sudo ln -s ../local/lib/libwiringPiDev.so.2.0 libwiringPiDev.so
```

In questo modo il cross-compiler potrà localizzare le librerie.

E.3 Configurazione di un progetto in QtCreator

Modificare il file .pro aggiungendo queste linee:

```
INCLUDEPATH += /mnt/rasp-pi-rootfs/usr/local/include  
LIBS += -L/mnt/rasp-pi-rootfs/usr/local/lib -lwiringPi
```

Una volta configurato il progetto, questo non può essere compilato localmente dato che la macchina locale non possiede dei GPIO :)

E.4 Mappatura dei GPIO

In figura E.1 è mostrato come la libreria mappa i pin disponibili.

Raspberry Pi P1 Header					
PIN #	NAME			NAME	PIN #
	3.3 VDC Power		1	2	5.0 VDC Power
8	SDA0 (I2C)	3	4	DNC	
9	SCL0 (I2C)	5	6	0V (Ground)	
7	GPIO 7	7	8	TxD	15
	DNC	9	10	RxD	16
0	GPIO 0	11	12	GPIO1	1
2	GPIO2	13	14	DNC	
3	GPIO3	15	16	GPIO4	4
	DNC	17	18	GPIO5	5
12	MOSI	19	20	DNC	
13	MISO	21	22	GPIO6	6
14	SCLK	23	24	CE0	10
	DNC	25	26	CE1	11

<http://www.pi4j.com>

Figura E.1: Come i pin sono segnati nella libreria WiringPi.

Appendice F

Struttura della directory dell'applicazione

L'applicazione risiede nella directory /opt/stouch. Tutti i file necessari al suo funzionamento sono in questa directory. La struttura delle directory dell'applicazione e dei file utilizzati è la seguente:

```
/opt/stouch/
├── bkp/
│   └── Prototipo1b
│       └── stouchdaemon
├── crop/
│   └── *.jpg
├── cropper
├── daemon/
│   ├── client.pid
│   └── stouchd.lock
├── img/
│   └── *.jpg
├── log/
│   └── *.log
├── Prototipo1b
├── script/
│   └── checkip
├── sound/
│   ├── *.mp3
│   └── README .2 sounds.config
└── stouchapp -> Prototipo1b
    └── stouch.config
        └── stouchdaemon
```

Il demone, file bash **stouchd**, risiede invece nella directory di sistema /etc/init.d.

Appendice G

Raccolta di comandi utili

G.1 Connessione remota

Per connettersi da un altro computer nella stessa rete del Raspberry Pi viene utilizzato il seguente comando:

```
ssh pi@<Raspberry Pi IP>
```

Mentre per inviare e recuperare file dal dispositivo viene utilizzato il seguente comando duplice:

```
scp <file da inviare> pi@<Raspberry Pi IP>:<destinazione>
```

```
scp pi@<Raspberry Pi IP>:<file da recuperare> .
```

G.2 Installazione

L'installazione dell'applicazione avviene in due passi.

Il primo passo consiste nel copiare la cartella dell'applicazione `stouch`, presente sul DVD allegato, nella directory `/opt/` sul Raspberry Pi.

Il secondo passo prevede la registrazione del demone di sistema in modo che possa avviarsi in maniera automatica all'avvio del sistema. La registrazione avviene copiando il (o creando un link al) file `stouchd` nella directory `/etc/init.d`. In seguito lanciare il seguente comando per registrarlo:

```
sudo update-rc.d stouchd defaults
```

G.3 Avviare e fermare il demone

Per avviare il demone sul Raspberry Pi:

```
sudo /etc/init.d/stouchd start
```

Per fermare il demone E l'applicativo:

```
sudo killall stouchapp stouchd  
sudo /etc/init.d/stouchd stop
```

Bibliografia

- [1] Raspberry pi | model b, accessories, camera and projects - element14. <http://www.element14.com/community/groups/raspberry-pi>.
- [2] Beagleboard.org - beaglebone black. <http://beagleboard.org/Products/BeagleBone%20Black>.
- [3] Apc » rock. <http://apc.io/products/rock/>.
- [4] mini-itx.com - store - asrock mini-itx motherboards. <http://www.mini-itx.com/store/?c=96#B85M-ITX>.
- [5] Raspberry pi raspberry pi. <http://www.pi-shop.ch/>.
- [6] Modmypi | raspberry pi camera board (5mp, 1080p, v1.3). <https://www.modmypi.com/raspberry-pi-camera-board>.
- [7] Caen rfid - uhf readers and tags. caenrfid.it.
- [8] Frontpage - raspbian. <http://www.raspbian.org/>.
- [9] Cmsis - cortex microcontroller software interface standard - arm. <http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php>.
- [10] Qt project. Beginner's guide to cross-compile qt5 on raspberrypi. http://qt-project.org/wiki/RaspberryPi_Beginners_guide, 2013.
- [11] Qt5 on raspberry pi (from devaamo summit 2012) - youtube. <http://www.youtube.com/watch?v=0j-Wakm5B84>.
- [12] Gaetano Di Blasio. Oracle java me embedded per l'internet delle cose. <http://www.tomshw.it/cont/news/oracle-java-me-embedded-per-l-internet-delle-cose/48210/1.html>.
- [13] Gordon Henderson. Wiring pi library. <https://projects.drogon.net/raspberry-pi/wiringpi/>, 2013.

- [14] Shahmir Javaid. Beginners guide to creating a daemon in linux. <http://shahmirj.com/blog/beginners-guide-to-creating-a-daemon-in-linux>, 2010.
- [15] Devin Watson. Linux daemon writing howto. <http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html>, 2004.
- [16] Maya Posch. How to really, truly use qthreads; the full explanation. <http://mayaposch.wordpress.com/2011/11/01/how-to-really-truly-use-qthreads-the-full-explanation/>, 2011.
- [17] Hussam Al-Hertani. Development environment for the raspberry pi using a cross compiling toolchain and eclipse. <http://hertaville.com/2012/09/28/development-environment-raspberry-pi-cross-compiler/>, 2013.
- [18] Raspberry pi + opencv. <http://mitchtech.net/raspberry-pi-opencv/>, 2012.
- [19] OpenCV foundation. Cross compilation for arm based linux systems. http://docs.opencv.org/doc/tutorials/introduction/crosscompilation/arm_crosscompile_with_cmake.html#arm-linux-cross-compile, 2011 - 2013.
- [20] Mohamed belhadj | install opencv 2.31 on ubuntu 11.10 and configure qtcreator. <http://madmed88.tumblr.com/post/13290466909/install-opencv-2-31-on-ubuntu-11-10-and-configure>.
- [21] Simone Piccardi. *Guida alla Programmazione in Linux*. 2010.
- [22] Beej Jorgensen. Beej's guide to unix ipc. <http://beej.us/guide/bgipc/output/html/multipage/index.html>, 2010.
- [23] Unix daemon server programming. <http://www.enderunix.org/docs/eng/daemon.php>, 2001.
- [24] The opencv reference manual 2.4.6.0. <http://docs.opencv.org/opencv2refman.pdf>, 2013.
- [25] Gary A. Stafford. Object tracking on the raspberry pi with c++, opencv, and cvblob. <http://programmaticponderings.wordpress.com/2013/02/09/opencv-and-cvblob-with-raspberry-pi/>, 2013.
- [26] Face detection. http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/robot/face_detection/.
- [27] Pierre Raufast. Opencv and pi camera board. <http://thinkrpi.wordpress.com/2013/05/22/opencv-and-camera-board-csi/>, 2013.

- [28] Alessio Placitelli. Cross-compilation in cygwin for the raspberry-pi: configure and cmake. <http://www.a2p.it/wordpress/tech-stuff/development/cross-compilation-in-cygwin-for-the-raspberry-pi-configure-and-cmake/>, 2012.
- [29] How to cross compile opencv 2.2.0 for beagleboard-xm. <http://os-infoboys.blogspot.fr/2011/04/how-to-cross-compile-opencv-220-for.html>, 2011.
- [30] Josiah Yoder. How to compile opencv based programs in linux. <http://opencv.willowgarage.com/wiki/CompileOpenCVUsingLinux>, 2011.
- [31] Building opencv for arm cortex-a8. http://processors.wiki.ti.com/index.php/Building_OpenCV_for_ARM_Cortex-A8, 2012.
- [32] Qt Project. Qt 5.0 doc. <http://qt-project.org/doc/qt-5.0/qtdoc/index.html>, 2013.
- [33] Tim Hutt. Beginner's guide to cross-compile qt5 on raspberrypi with qtmultimedia and libcec support! <http://creeder.com/?page=RaspberryPi>, 2012.
- [34] Luca Carlon. Building the qt fundamental modules. <http://thebugfreeblog.blogspot.ch/2013/03/bring-up-qt-501-on-raspberry-pi-with.html>, 2013.
- [35] Roberto Raggi. Qt quick for c++ developers. http://qt-project.org/videos/watch/qt_quick_for_c_developers, 2013.
- [36] J: .ohan Thelin. Building qml run-time. <http://www.slideshare.net/e8johan/building-the-qml-runtime>, 2012.
- [37] Quit coding. <http://quitcoding.com/>, 2013.
- [38] Cross compiling qt for the masses. <https://blog.qt.digia.com/blog/2012/04/13/cross-compiling-qt-for-the-masses/>, 2012.
- [39] A simple http server. <http://doc.qt.digia.com/solutions/4/qtservice/qtservice-example-server.html>.
- [40] Qt Project. Threading basics. <http://qt-project.org/doc/qt-4.8/thread-basics.html>, 2011.
- [41] Jasmin Blanchette. Monitors and wait conditions in qt. <http://doc.qt.digia.com/qq/qq21-monitors.html>, 2007.
- [42] Qvision: Computer vision library for qt. <http://qvision.sourceforge.net/>, 2011.
- [43] Cross compile wiringpi with crosstool. <http://www.raspberrypi.org/phpBB3/viewtopic.php?f=33&t=39198>.

- [44] Beagle board. <http://beagleboard.org/>.
- [45] Apc - mini android pc system. <http://apc.io/specifications/>.
- [46] Serial programming howto. <http://www.tldp.org/HOWTO/Serial-Programming-HOWTO/x115.html>.
- [47] Serial programming/termios. http://en.wikibooks.org/wiki/Serial_Programming/termios.

Allegati