ITIS "Mario Delpozzo"

Elaborato - Esame di Stato

LIRC stateless per dispositivi home

Jonathan Isoardo

Docente di riferimento
Andrea MAGANZA

Giugno 2021

Indice

Capitolo 1

Introduzione

Al giorno d'oggi la domotica è ormai entrata a far parte della vita di un gran numero di persone e la possibilità di facilitare azioni relative alla quotidianità attraverso dispositivi elettronici e diventata l'obbiettivo di tante piccole e grandi aziende.

Un caso interessante risale a qualche anno fa, quando erano spesso presenti nei telefoni dei trasmettitori infrarossi che permettevano ai cellulari di funzionare anche come telecomandi per televisioni, stereo, condizionatori, ecc... Ad oggi è molto raro, se non impossibile trovare un telefono con un trasmettitore infrarossi integrato; ovviamente sul mercato esistono svariati dispositivi che sopperiscono a questa mancanza, ma un altra opzione è quella di costruire una soluzione personalizzata come in questo caso.

Lo scopo del progetto è quello di fornire all'utente un interfaccia grafica in grado di riprodurre uno o più telecomandi infrarossi e che gli permetta quindi di controllare gli elettrodomestici correlati, senza usufruire del telecomando originale.

L'elaborato sviluppa i seguenti punti:

- Introduzione a **LIRC**
- Installazione e configurazione di **LIRC**
- Creazione di un supporto hardware per i segnali infrarossi
- Creazione di un applicazione per l'utilizzo di **LIRC**

Andando poi ad analizzare un esempio di controllo remoto.

Inoltre il progetto è Open Source e disponibile per il download su GitHub[1], dove è riportato passo passo il procedimento per installare e configurare tutto il necessario per far funzionare l'applicazione.

[1] https://www.github.com/joj-32/RPiremote

1.1 Scelta dell'hardware

Per raggiungere lo scopo prefissato è necessario configurare un terminale Linux affinché **LIRC** (una volta installato e configurato) possa usufruire di periferiche Hardware, quali un trasmettitore e un ricevitore di infrarossi. In questo caso specifico viene utilizzato un Raspberry Pi, al quale sono collegati due circuiti fatti a mano: uno per il trasmettitore e uno per il ricevitore. Alternative valide sono utilizzare degli shield prefabbricati, composti da ricevitore e trasmettitore oppure ancora dei dongle collegabili ad un qualsiasi tipo di PC tramite USB.

La scelta dell'Hardware ricade su un RPi, piuttosto che su un più classico PC, perché la dimensione del computer a scheda singola gli permette di essere trasportato in giro all'occorrenza o più semplicemente di prendere una postazione fissa all'interno di un salotto senza essere di ingombro o risultare spiacevole alla vista. Inoltre le schede Raspberry Pi sono ad oggi largamente utilizzati per ottenere altri benefici di natura domotica all'interno della propria casa per esempio attraverso il più famoso Home Assistant[2].

1.2 Scelta del software

La scelta più opportuna è di fare in modo che l'interfaccia sia raggiungibile in maniera comoda tramite il telefono, che è il dispositivo più adatto all'utilizzo di una simile applicazione perché maggiormente maneggiabile, esattamente come un telecomando. Il sistema più pratico per raggiungere questo obbiettivo è quello di avviare un server sulla macchina con **LIRC** che, una volta connessa alla rete, rende possibile l'accesso all'interfaccia web da qualsiasi dispositivo che sia connesso alla stessa LAN.

La soluzione migliore per avviare un server sul Raspberry Pi, assicurandosi di avere accesso a delle API per utilizzare **LIRC**, è quello di utilizzare Java Script con NodeJS e un pacchetto npm tra i vari creati per iterfacciarsi a **LIRC**.

Oltre al linguaggio di programmazione è però importate stabilire anche il DBMS da utilizzare per immagazzinare i dati dei vari telecomandi disponibili sull'applicazione. Visto il fatto che il DataBase viene creato e gestito direttamente dall'utente, acceduto solo da un applicazione locale (il nostro programma) e non deve contenere dati complessi si può optare per un semplice SQLite.

[2] https://www.home-assistant.io/

1.3 Monitoraggio progetto

Ai fini del monitoraggio del progetto ho utilizzato TeamGantt[5] Lo sviluppo del progetto si suddivide in 4 gruppi principali composti come segue:

- Analisi del problema
  - Studio di LIRC
  - Scelta dell'hardware
  - Scelta del software
  - Creazione Gantt
- Sviluppo hardware
  - Circuito ricevitore
  - Circuito trasmettitore
- Sviluppo software
  - Web app
  - Database
- Conclusione
  - Testing
  - Redazione presentazione

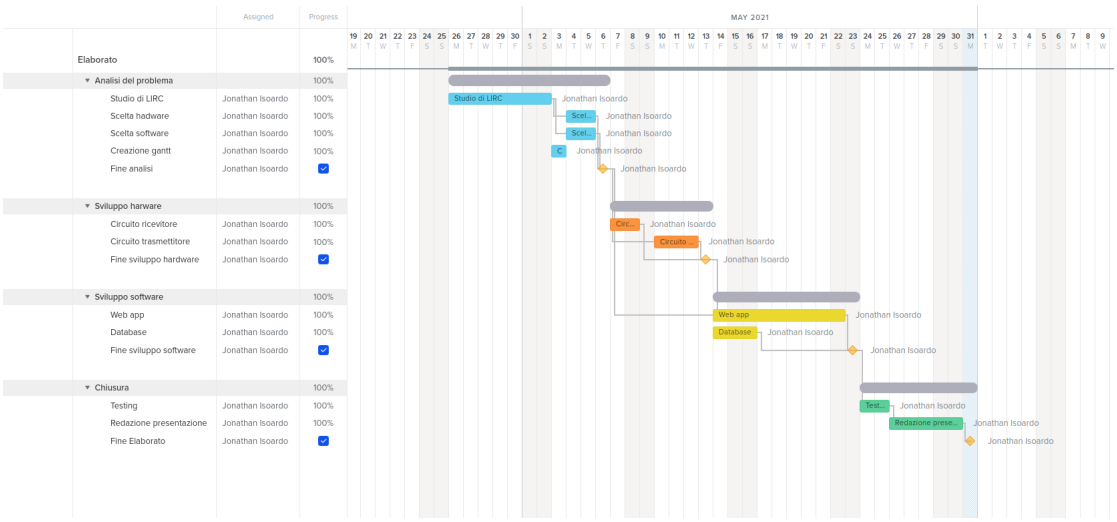

Figura 1.1: Diagramma di Gantt

Ad ogni modo il Diagramma è reperibile sulla piattafoma di teamgantt[3] (sul mio profilo personale).

[3]https://app.teamgantt.com

Capitolo 2

LIRC

Linux Infrared Remote Control (**LIRC**) è un pacchetto che consente di decodificare e inviare segnali infrarossi di molti i telecomandi di uso comune.

Le informazioni reperibili in rete riguardo **LIRC** non sono molte, complice il fatto che lo sviluppo del pacchetto è stato abbandonato nel 2017. Questo fa si che la maggior fonte di informazioni sia il sito ufficiale[3].

2.1 Limiti

LIRC ad oggi presenta non pochi problemi che bisogna tenere di conto nel momento in cui si intende sviluppare un applicativo che usufruisca del pacchetto.

2.1.1 Distribuzione

LIRC è pacchettizzato solo da alcune distribuzioni Linux, tra queste c'è Debian e di conseguenza anche il sistema operativo ufficiale Raspberry Pi: Rasperry Pi OS (ex Raspbian). Altre distribuzioni famose sono: Arch Linux Gentoo, RedHat e SUSE.

2.1.2 Aggiornamenti

Come già detto **LIRC** non viene aggiornato ormai da 5 anni e questo non sarebbe un problema, non fosse che nel 2018 (un anno dopo l'ultimo rilascio) l'overlay per la configurazione di **LIRC** "lirc_rpi" è deprecato per essere sostituito da altri due overlays: "gpio_ir" e "gpio_ir_tx". Questo rende alcuni comandi di **LIRC** completamente o parzialmente inutilizzabili.

2.1.3 Funzionalità

I problemi del pacchetto relativi alle sue funzionalità sono indipendenti da sistemi operativi o aggiornamenti di sorta. In **LIRC** non è mai stato implementato il supporto per la creazione di telecomandi stateful. Questo porta non tutti i tipi di telecomandi a poter essere analizzati e utilizzati tramite **LIRC**. Per sopperire a questa mancanza sono state create delle soluzioni non ufficiali[1].

2.1.4 Sintassi

LIRC necessità di immagazzinare i dati relativi ai vari telecomandi registrati dall'utente, ma per farlo non utilizza nessun DBMS, bensì immagazzina i telecomandi su file di testo all'interno della directory */etc/lirc/lircd.conf.d* con una sintassi personalizzata e interpretata a DOC dal pacchetto stesso.

L'utilizzo di questa sintassi è dovuto al fatto che all'inizio dello sviluppo di **LIRC**, nel 1999, non erano ancora diffusi linguaggi come il JavaScript Object Notation (JSON)

A sinistra si può notare un file corrispondente ad un telecomando scritto con la sinstassi di **LIRC**, sulla destra è invece presente lo stesso file riscritto in sintassi JSON.

```
begin remote
  name            devinput
  bits            16
  eps             30
  aeps            100
  pre_data_bits   16
  pre_data        0x0001
  post_data_bits  32
  post_data       0x00000001
  gap             132799
  toggle_bit      0

  begin codes
    KEY_0         11
    KEY_1         2
    KEY_102ND     86
    ...
    BTN_WEST      0x134
    BTN_WHEEL     0x150
    BTN_Z         0x135
  end codes
end remote
```

```
{
  "name": "devinput",
  "bits": 16,
  "eps": 30,
  "aeps": 100,
  "pre_data_bits": 16,
  "pre_data": "0x0001",
  "post_data_bits": 32,
  "post_data": "0x00000001",
  "gap": 132799,
  "toggle_bit": 0,

  "codes": {
    "KEY_0": "11",
    "KEY_1": "2",
    "KEY_102ND": "86",
    ...
    "BTN_WEST": "0x134",
    "BTN_WHEEL": "0x150",
    "BTN_Z": "0x135"
  }
}
```

Questa scelta di immagazzinamento dei dati si ripercuote purtroppo sulla fruibilità di **LIRC**, essendo che è impossibile interpretare un documento scritto con una sintassi simile a meno che non si crei un lettore apposito o in alternativa un convertitore.

2.2 Funzionamento

La parte più importante di **LIRC** è il demone lircd, che decodifica i segnali IR ricevuti dai driver del dispositivo e fornisce le informazioni su un socket. Lo stesso demone accetta anche comandi per l'invio di segnali infrarossi, se l'hardware lo supporta. Nonostante in svariati kernel Linux, il supporto agli infrarossi è implementato anche senza l'installazione di **LIRC** (che lo renderebbe ridondante); il pacchetto può comunque essere utilizzato per le più svariate applicazioni: mouse, telecomandi, controlli da remoto, programmazione di eventi degli elettrodomestici da computer, ecc...

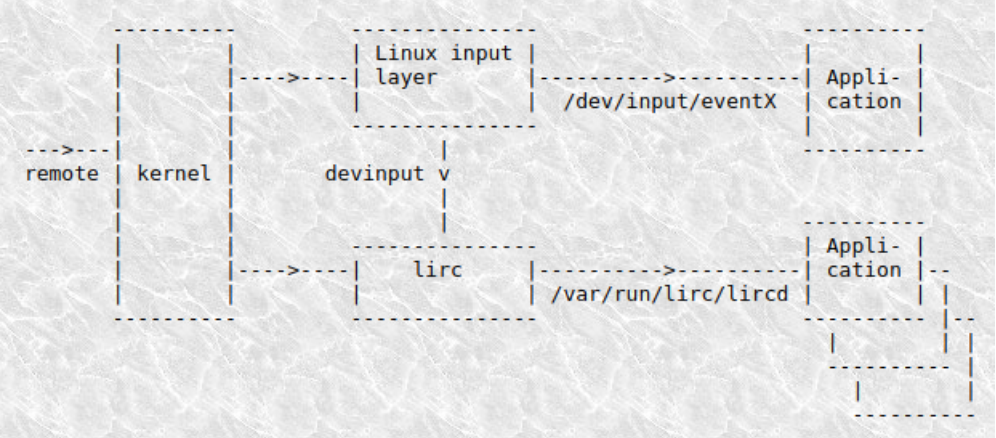

Figura 2.1: Rappresentazione di **LIRC** all'interno del sistema operativo

Come è possibile vedere nella figura 2.1, **LIRC** è in grado di interfacciarsi sia con il Linux input layer, sia direttamente con il kernel a seconda delle esigenze dell'utente. Nel caso in cui, ad esempio, un telecomando sia supportato dall'input layer, ma non da **LIRC**, il pacchetto può comunque usufruire dei dati offerti dal layer integrato attraverso il *devinput*, file presente in **LIRC** al momento dell'installazione. A seconda poi dell'approccio utilizzato, i dati verranno inviati alle varie applicazioni tramite *eventX*: nel caso in cui si utilizzi solamente il Linux input layer; mentre, al contratio, si usufruirà del demone *lircd* nel caso in cui venga utilizzato **LIRC**.[1]

[1]https://www.lirc.org/html/configuration-guide.html#overall-configuration-decisions

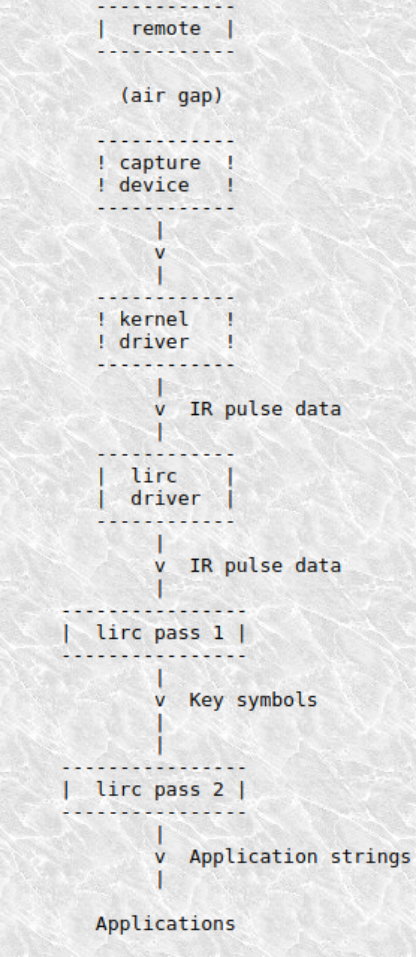

Figura 2.2: Schema del funzionamento di **LIRC**

Nella figura 2.2 è invece possibile vedere graficamente i vari passi che compiono i dati trasmessi dal telecomando per arrivare fino all'applicazione desiderata. In ordine: il telecomando invia dei segnali IR che attraversano l'aria fino a colpire un ricevitore. Successivamente il ricevitore di infrarossi comunicherà i dati al kernel, convertendo il segnale da analogico a digitale. I dati arriveranno poi a **LIRC** attraverso una delle due modalità precedentemente descritte e raffigurate in Figura 2.1. In seguito **LIRC** si occuperà di ricondurre i dati ricevuti al proprio comando, denominato "Key simbol" in figura 2.2. Infine grazie alle "Application strings", sempre in Figura 2.2, **LIRC** è in grado di eseguire un determinato comando con l'applicazione indicata.[2] Ovviamente il procedimento è riproducibile in senso inverso, solitamente partendo da un determinato Key symbol. Ovviamente è necessario che il "capture device" venga sostituito e/o affiancato da un emettitore, solitamente un led infrarossi. Una volta che i dati hanno fatto il percorso inverso, **LIRC** è in grado di far trasmettere al led la sequenza infrarossi desiderata, permettendo quindi di controllare i dispositivi elettronici prescelti.

Qui possiamo vedere l'esempio di un'Application string con Key symbol corrispondente a KEY_RED (button = KEY_RED) che fa eseguire il comando echo "foo" (config = echo "foo") al programma irexec (prog = irexec)[3].

```
begin
  remote = mceusb
  button = KEY_RED
  prog   = irexec
  config = echo "foo"
end
```

[2] https://www.lirc.org/html/configuration-guide.html#basic-setup-flow
[3] https://www.lirc.org/html/configuration-guide.html#key-symbols-to-app-strings

2.3 Installazione e configurazione

Per installare **LIRC** sulla propria macchina si può fare riferimento alla guida sul sito ufficiale[4], che prevede di scaricare i file, per poi compilarli e installarli a mano con una combinazione di comandi:

```
make
sudo make install
```

Essendo però che **LIRC** è pacchettizzato ufficialmente su Advanced Packaging Tool (APT), è anche possibile seguire un qualsiasi altro tutorial[5] che utilizza APT a patto che la propria distribuzione Linux lo utilizzzi come packet manager o abbia la possibilità di installarlo. Su Raspberry Pi OS APT, è ovviamente preinstallato. In questo caso l'unico comando da eseguire è:

```
sudo apt-get install lirc
```

2.3.1 Impostazioni importanti

Una volta installato **LIRC** è essenziale andare a modificare almeno un'impostazione del file *lirc_option.conf* situato in */etc/lirc*. Più precisamente è necessario sostituire la riga:

```
driver = devinput
```

con:

```
driver = default
```

Questo consente a **LIRC** di accere direttamente al kernel senza passare dal Linux input layer, dato che potrebbe portare a problemi, causa il fatto che **LIRC** non viene aggiornato da anni.

Inoltre molti (se non tutti) i tutorial che spiegano come configurare **LIRC** reperibili online consigliano di cambiare la riga:

```
device = auto
```

e inserire al suo posto:

```
device = /dev/lirc0
```

Questo tuttavia è del tutto superfluo perché la scelta del dispositivo (device) può essere effettuata ogni qualvolta che si utilizza un comando relativo a **LIRC** grazie al parametro opzionale -d. Esempio:

```
mode2 -d /dev/lirc0
```

[4]https://www.lirc.org/html/install.html#compile
[5]https://help.ubuntu.com/community/LIRC

2.3.2 Abilitare i dispositivi

Utilizzando un Raspberry Pi è necessario indicare al kernel che, in fase di avvio, devono essere creati dei dispositivi virtuali *lirc0* e *lirc1* situati nella cartella */dev*, che sono abilitati a leggere/inviare dati attraverso i pin del General Porpouse Input Output (GPIO).

Per fare questo bisogna modificare il file di configurazione di avvio chiamato *config.txt* e situato in */boot*. Più precisamente è necessario decommentare due delle righe già scritte, passando da questa situazione:

```
# Uncomment this to enable infrared communication.
#dtoverlay=gpio-ir,gpio_pin=17
#dtoverlay=gpio-ir-tx,gpio_pin=18
```

a questa:

```
# Uncomment this to enable infrared communication.
dtoverlay=gpio-ir,gpio_pin=17
dtoverlay=gpio-ir-tx,gpio_pin=18
```

Come indicato nelle due righe soprastanti i pin del GPIO interessanti sono rispettivamente il 17, per il ricevitore, e il 18, per il trasmettitore. I dispositivi virtuali creati vengono sempre assegnati come *lirc1*, per il ricevitore, e *lirc0*, per il trasmettitore. Nel caso in cui si voglia solamente ricevere, piuttosto che trasmettere, basta decommentare solo la riga interessata; in questo caso all'avvio verrà creato un solo dispositivo virtuale chiamato *lirc0*.

2.4 Comandi

LIRC offre una vasta serie di comandi e strumenti utili a gestire i segnali infrarossi[6] tuttavia quelli essenziali sono:

- mode2[7]
- irrecord[8]
- irsend[9]

[6] https://www.lirc.org/html/programs.html#list
[7] https://www.lirc.org/html/mode2.html
[8] https://www.lirc.org/html/irrecord.html
[9] https://www.lirc.org/html/irsend.html

2.4.1 mode2

mode2 è un comando utilizzato generalmente per visualizzare su stdout i segnali ricevuti. I segnali visualizzati sono posti sotto forma di una sequenza di numeri che indicano alternativamente la presenza o l'assenza di un segnale. Può però anche essere utilizzato per la generazione di telecomandi o per le Application String di Figura 2.2.

2.4.2 irrecord

irrecord è un comando utilizzato per registrare multipli Key simbols di Figura 2.2 all'interno dello stesso file, detto telecomando. Questo file viene salvato come *nomeTelecomando.lircd.conf* all'interno della cartella che contiene già il file di *devinput*, ossia *etc/lirc/lircd.conf.d*.

2.4.3 irsend

irsend è un comando che permette di inviare segnali infrarossi tramite un trasmettitore, solitamente un led. È in grado di inviare qualsiasi segnale, a patto che sia presente come Key simbol in uno dei telecomandi registrati.

2.5 Problemi

Il grande problema che si riscontra, quando si prova ad utilizzare il comando irrecord, sta nel fatto che tutti i telecomandi che si proveranno a registrare daranno come risultato dei telecomandi con Key Simbols tutti corrispondenti a 0x0. Questo è causato dal fatto che **LIRC** è molto meno aggiornato di un qualsiasi kernel, il pacchetto non è infatti in grado di riconoscere a dovere la parte finale dei segnali infrarossi, interpretando i valori di Timeout come parte del segnale. Per aggirare il problema si può registrare i segnali con mode2, modificarli manualmente e poi convertirli in un telecomando tramite irsend. I comandi da utilizzare sono:

```
mode2 -m -d /dev/lirc1 > nomeTelecomando.lircd.conf
```

e dopo aver effettuato le opportune modifiche[10]

```
irrecord -a nomeTelecomando.lircd.conf
```

[10]https://github.com/joj-32/RPiremote/blob/main/README.md

Il primo comando scrive su file una lista di numeri corrispondenti al segnale infrarossi crudo (raw). Da ognuno de segnali va rimossa la parte finale, corrispondente al tempo trascorso tra la pressione di un pulsante e il successivo. Va poi inserita l'intestazione e la del telecomando, va assegnato un nome (Key symbol) ad ognuno dei comandi e infine va chiusa l'intestazione a fine file.

```
Running as regular user pi
 16777215

    9213     4391      704      451      684      451
     685      450      687      451      685      451
     687      451      687      451      687     1564
     685     1559      688     1541      709     1560
     689     1558      693     1558      689     1558
     689     1559      689      450      687      428
     711     1561      688      449      689      449
     686     1559      688      450      689      449
     716      403      712     1538      714      429
     714     1537      716     1559      698      451
     699     1562      694     1537      719     1560
     708    39842

    9341     2140      640   142731-pulse   512730-space

  489835-space

    9284     4463      643      500      644      478
     666      501      642      477      666      500
     645      500      644      501      645     1610
     646     1609      644     1611      646     1609
     646     1590      664     1611      644     1610
     645     1609      643      500      643      501
     642     1613      641      501      642     1610
     645     1610      641      479      663      478
     665      499      640     1610      639      501
     640     1612      639      502      642      502
     641     1591      661     1610      640     1612
     641    40003

    9208     2188      641   142061-pulse   502061-space

  481908-space

    9165     4464      637      498      638      499
     669      468      669      468      638      499
```

```
begin remote

  name   RGBled_mini
  flags RAW_CODES
  eps            25
  aeps          100

  ptrail          0
  repeat     0    0
  gap        100000

  begin raw_codes
    name ON
      9213     4391      704      451      684      451
       685      450      687      451      685      451
       687      451      687      451      687     1564
       685     1559      688     1541      709     1560
       689     1558      693     1558      689     1558
       689     1559      689      450      687      428
       711     1561      688      449      689      449
       686     1559      688      450      689      449
       716      403      712     1538      714      429
       714     1537      716     1559      698      451
       699     1562      694     1537      719     1560
       708

    name OFF
      9284     4463      643      500      644      478
       666      501      642      477      666      500
       645      500      644      501      645     1610
       646     1609      644     1611      646     1609
       646     1590      664     1611      644     1610
       645     1609      643      500      643      501
       642     1613      641      501      642     1610
       645     1610      641      479      663      478
       665      499      640     1610      639      501
       640     1612      639      502      642      502
       641     1591      661     1610      640     1612
       641
```

Figura 2.3: A sinistra il file generato da mode2, a destra lo stesso file dopo le modifiche

Nel caso in cui persistano i problemi si può provare a patchare **LIRC** con degli aggiornamenti non ufficiali[11]. Se ancora non funziona è segno del fatto che il telecomando non è probabilmete supportato.

[11] https://gist.github.com/billpatrianakos/cb72e984d4730043fe79cbe5fc8f7941

Capitolo 3

Hardware

Presupponendo di disporre di un Raspberry Pi con Raspberry Pi OS, aggiornato all'ultima versione, è necessario creare 2 circuiti, rispettivamente per il trasmettitore e per il ricevitore. I pin utilizzati (17 e 18) non sono scelti casualmente, ma sono quelli già predisposti dal Raspberry Pi, tra tutti e 40 quelli presenti nell'interfaccia GPIO, per la gestione dei segnali IR.

3.1 Ricevitore

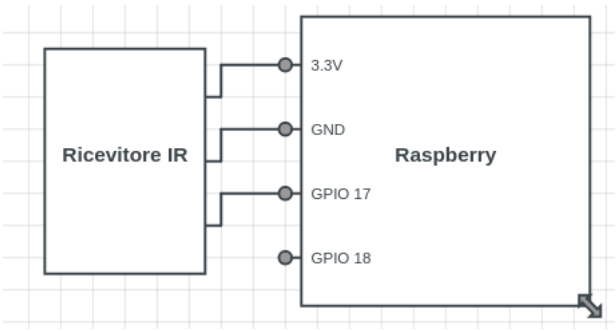

Figura 3.1: Circuito del ricevitore[2]

Come è possibile vedere in Figura 3.1 è necessario disporre unicamente di un ricevitore infrarossi affinché il circuito possa funzionare correttamente.

Per testare il ricevitore basta eseguire il comando seguente e premere dei pulsanti di un telecomando nella direzione del ricevitore, dovrebbero comparire sul teminale delle sequenze di numeri.

```
mode2 -d /dev/lirc1
```

3.2 Trasmettitore

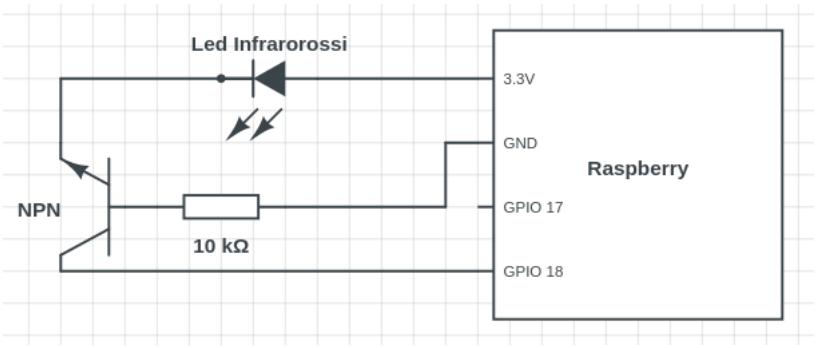

Figura 3.2: Circuito del trasmettitore[2]

Come è possibile vedere in Figura 3.2 è necessario disporre, oltre che di un led infrarossi, di una resistenza (10kΩ) e di un transistor NPN affichè il pin 18 riceva segnali solo quando necessario.

Per testare il trasmettitore l'opzione più semplice è quella di eseguire un semplice script in Python, come quello riportato sotto, che accende e spegne ripetutamente il led. Essendo che le frequenze infrarossi sono impercettibili per l'occhio umano si può controllare il funzionamento, o meno, del led tramite una telecamera (ad esempio quella del telefono).

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
while True:
GPIO.output(18, True)
time.sleep(1)
GPIO.output(18, False)
time.sleep(1)
```

Capitolo 4

Software

L'obbiettivo dell'applicazione, chiamata RPiremote, è quello di essere facilmente utilizzabile da tutte le persone connesse alla stessa LAN del Raspberry Pi, plausibilmente attraverso uno smartphone. Inoltre è importante che l'interfaccia grafica riesca a riprodurre in maniera fedele qualsiasi telecomando si voglia registrare.

Lo sviluppo del software è suddivisibile in due parti principali:

- Creazione della web app
- Creazione del DataBase

4.1 Web app

RPiremote è una web app scritta in Java Script con HTML, CSS, Bootstrap e EJS. L'applicazione utilizza Node.js per avviare un server sul RPi ed Express per gestire le richieste sul server.

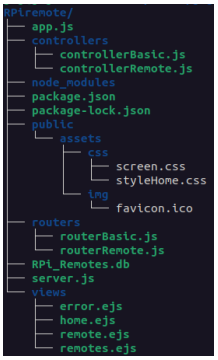

Figura 4.1: Struttura ad albero del progetto

Il file principale all'interno di Rpiremote è *server.js*, infatti il server viene avviato tramite il comando:

```
node server.js
```

All'interno del file ci sono le righe minime ed indispensabili per avviare l'applicazione contenuta nel file *app.js*

```
const app = require('./app')

const port = 8080
app.listen(port, () => console.log(`App listening on port ${port}`))
```

Secondo in ordine di importanza è il file *app.js*, che fa da "padre" a tutto il resto dell'applicazione. Al suo interno sono presenti le importazioni e impostazioni dei vari pacchetti, in aggiunta alla definizione dei routers del server.

```
const express = require('express')
const bodyParser = require('body-parser')
const lirc = require('lirc_node')
lirc.init()

const app = express()

const routerBasic = require('./routers/routerBasic')
const routerRemote = require('./routers/routerRemote')

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({
    extended: true
}))

app.use(express.static(__dirname + '/public'))

app.set('view engine', 'ejs')

app.use('/', routerBasic)
app.use('/remote', routerRemote)

module.exports = app
```

4.1.1 Routers

Nella cartella *routers* sono presenti i file *routerBasic.js* e *routerRemote.js* che gestiscono le richieste rispettivamente sulle routes */* e */remotes* per *routerBasic.js* e */:remote* per *routerRemote.js*. Il loro scopo è quello di reindirizzare le richieste HTTP ricevute sul server ai controller appositi, situati nella cartella *controllers*, in modo ordinato e comprensibile.

4.1.2 Controllers

Nella cartella *controllers* sono presenti le parti di codice contenenti le operazioni da eseguire, una volta ricevuta una richiesta. Tramite Express la richiesta viene processata e poi viene restituita una pagina HTML all'utente.

Il file *controllerBasic.js* si occupa di gestire le richieste GET e POST, effettuate sulla home (*home.ejs*) e sulla pagina contenente la lista di tutti i telecomandi (*remotes.ejs*). Nel caso di una richiesta di tipo GET verrà effettuata una semplice **read** sul DataBase, con l'obbiettivo di visualizzare tutti i telecomandi disponibili. Nel caso di una richiesta di tipo POST invece, uno switch si occupa di capire che tipo di operazione effettuare sul DB: **create**, **update** o **delete**.

Il file *controllerRemote.js* si occupa invece, di gestire le richieste GET e POST, effettuate sulla pagina del telecomando (*remote.ejs*). Nel caso di una richiesta GET verrà semplicemente renderizzata la pagina del telecomando, diversa per ciascun modello. Nel caso di una richiesta POST, effettuata premendo uno dei pulsanti del telecomando, viene ricaricata la stessa pagina inviando un segnale infrarossi, tramite lirc_node.

4.1.3 Views

Nella cartella *views* sono presenti i file HTML, salvati con l'estensione *.ejs* perchè utilizzano l'Embedded JavaScript templating (EJS). L'utilizzo di EJS serve a renderizzare dinamicamente le pagine sulla base dei parametri che Express manda nel corpo della richiesta. La funzione principale è quella di garantire una personalizzazione dei pulsanti del telecomando: colore, forma, dimensione, disposizione ed eventuali scritte aggiuntive; in base al modello.

4.1.4 Pacchetti

I pacchetti di Node Packet Manager (npm) utilizzati dall'applicazione tolti quelli correlati ad Express sono 2[4]:

- sqlite3[1]
- lirc_node[2]

[1]https://www.npmjs.com/package/sqlite3
[2]https://www.npmjs.com/package/lirc_node

sqlite3

sqlite3 è un modulo semplice ed intuitivo che permette all'applicazione di eseguire delle query a picare su un qualsiasi DataBase SQLite.

lirc_node

lirc_node è un modulo npm che agisce come uno intermediario tra **LIRC** e Node.js. È progettato per essere utilizzato in una situazione in cui si desidera controllare dispositivi a infrarossi utilizzando **LIRC** da un'applicazione Node.js.

Ovviamente lirc_node non è l'unico modulo npm che offre delle API verso **LIRC**, ma rimane uno dei principali motivi per il quale RPiremote è sviluppato in Java Script. L'estrema semplicità che offre il pacchetto permette infatti di avere unicamente 3 righe dedicate in tutto il codice.

```
const lirc = require('lirc_node')
lirc.init()

lirc.irsend.send_once(remote, command)
```

4.2 DataBase

Il DataBase, corrispondente al file *RPi_Remotes.db*, contiene inizialmente 2 tabelle: Remotes e Added_remotes. Le altre tabelle devono essere create dall'utente, che genererà in base ai telecomandi che vuole memorizzare. Ogni dettaglio estetico memorizzato nelle tabelle deve rispettare determinati canoni, essendo gestito da CSS e Bootstrap all'interno di tag EJS.

4.2.1 Remotes

In Remotes sono immagazzinati i nomi (id) di tutti i modelli di telecomando registrati da **LIRC**. Oltre ai nomi è anche presente una colonna per indicare un icona simbolica per il telecomando (icon), le opzioni disponibili sono: tv, cast, speaker, lightbulb, lamp e controller.

id	icon

Tabella 4.1: Tabella Remotes

4.2.2 Added_remotes

In Added_remotes è presente la lista di telecomandi da visualizzare nella home dell'applicazione. In questo caso è possibile effettuare una maggiore personalizzazione sulle schede dei telecomandi, scegliendone il nome (name), l'icona (icon), il colore di sfondo (color), e il colore dell'icona (icon_color). Tutte queste opzioni sono sempre modificabili dall'utente tramite un apposito tasto, posto a fianco del nome di ogni telecomando. Ovviamente è sempre presente il modello di riferimento del telecomando (id).

name	icon	color	icon_color	id

Tabella 4.2: Tabella Added_remotes

4.2.3 Altre tabelle

Per finire, è possibile inserire un numero infinito di tabelle, in corrispondenza di infiniti possibili telecomandi. Ogni tabella deve avere il nome del telecomando del quale immagazzina i dati e, in aggiunta, deve essere inserita come id nella tabella Remotes. Essendo che ogni record di questa tabella rappresenta un pulsante, tutte le colonne hanno l'obbiettivo di definire le caratteristiche estetiche del pulsante: colore (color), forma (shape: circle o rectangle), testo (text), dimensione (dimension: small o large) e numero di pulsanti sulla stessa riga (row). Caso a parte è la colonna del comando (command), nella quale va inserito il nome del Key simbol collegato al pulsante.

command	color	shape	text	dimension	row

Tabella 4.3: Tabella nomeTelecomando

Capitolo 5

Conclusioni

In conclusione, si può definire come completato con successo il progetto. L'applicazione sviluppata è sufficientemente versatile ed è in grado di riprodurre fedelmente qualsiasi tipo di telecomando.

5.1 Sviluppi futuri

Un aggiornamento futuro del software dovrebbe avere lo scopo di facilitare la creazione di nuovi telecomandi: sia nei file di **LIRC**, sia nel DataBase.

Per automatizzare la creazione dei file di tipo *nomeTelecomando.lircd.conf*, andrebbe aggiunto all'interfaccia un pulsante, che grazie a lirc_node invia un comando di tipo irrecord al RPi. In seguito, bisognerebbe aggiornare **LIRC**, mettendo mano al codice sorgente (scritto in C), e fare in modo che il comando irrecord torni a funzionare correttamente.

Per automatizzare, invece, l'inserimento dei nuovi telecomandi all'interno del DB, è necessario scrivere uno script (plausibilmente in JS), in grado di convertire i file di tipo *.lircd.conf* in file JSON. A quel punto basterebbe eseguire una transizione sul DataBase per aggiungere il telecomando alla tabella *Remotes* e poi per creare la tabella del telecomando.

```
INSERT INTO Remotes (id) VALUES ('nomeTelecomando');

CREATE TABLE 'nomeTelecomando' (
"command"         TEXT NOT NULL UNIQUE,
"color" TEXT NOT NULL DEFAULT '',
"shape" TEXT NOT NULL DEFAULT '',
"text"  TEXT NOT NULL DEFAULT '',
"dimension"       TEXT NOT NULL DEFAULT '',
"row"   INTEGER NOT NULL DEFAULT 3,
PRIMARY KEY("command")
);
```

E poi compilare la tabella tramite i dati contenuti nel file JSON precedentemente creato.

Acronimi

LIRC Linux Infrared Remote Control. 3, 4, 6–13, 19, 21

API Application Programming Interface. 4, 19

APT Advanced Packaging Tool. 10

CSS Cascading Style Sheets. 16, 19

DB DataBase. 4, 16, 18, 19, 21

DBMS DataBase Managment System. 4, 7

EJS Embedded JavaScript templating. 16, 18, 19

GPIO General Porpouse Input Output. 11, 14

HTML HyperText Markup Language. 16, 18

HTTP Hypertext Transfer Protocol. 17

JS Java Script. 4, 16, 19, 21

JSON JavaScript Object Notation. 7, 21

LAN Local Area Network. 4, 16

npm Node Packet Manager. 4, 18, 19

RPi Raspberry Pi. 4, 6, 11, 14, 16, 21

Bibliografia

[1] Ion Ursachi. *Controllo remoto dei condizionatori d'aria attraverso LIRC*. Alma Mater Studiorum · Università di Bologna, 2018.

[2] *CircuitLab*. URL: `https://www.circuitlab.com/`.

[3] *LIRC - Linux Infrared Remote Control*. URL: `https://www.lirc.org/`.

[4] *npm*. URL: `https://www.npmjs.com/`.

[5] *TeamGantt*. URL: `https://www.teamgantt.com/`.