Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Lightboard:
un programma efficiente per costruire una LIM con Raspberry Pi e Wiimote

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Giovanni Incammicia

Sessione III
Anno Accademico 2013/2014

Introduzione

Lightboard è un programma che permette di creare una lavagna interattiva multimediale con il solo ausilio di una penna ad infrarossi, un Wiimote ed un computer con un sistema GNU/Linux. Già esistono programmi open source e completi che mirano a tale scopo, tuttavia questi programmi spesso non competono con le LIM commerciali.
Lightboard deriva dall'unione delle parole *lightweight* e *whiteboard*.
Infatti quello che rende diverso lightboard rispetto ad altri programmi (come python-whiteboard) è che punta ad avere buone prestazioni su computer con poche risorse. Con la diffusione di "mini-computer" come il Raspberry Pi, sono aumentati i progetti hardware, perché allora non creare una lavagna multimediale interattiva utilizzando Raspberry Pi come calcolatore?
Chi ha provato software come python-whiteboard (o analoghi basati su linguaggi di programmazione interpretati/semi-interpretati) su Raspberry Pi, si sarà accorto che consumano troppo e di conseguenza non permettono un utilizzo fluido, in alcuni casi risultano addirittura inutilizzabili.
Ecco perché è stato realizzato lightboard, un software derivato da python-whiteboard da cui prende soltanto l'interfaccia grafica, lasciando il cuore dell'elaborazione ad un motore ben più veloce, scritto in linguaggio C.

Precedenti lavori

> As of June 2008, Nintendo has sold nearly 30 million Wii game consoles. [..] This makes the Wii Remote one of the most common computer input devices in the world.

Così scriveva Johnny Chung Lee nel suo sito web [14] quando iniziò il suo viaggio per scoprire le potenzialità di uno tra i più comuni dispositivi di input per computer. Ispirandosi alle idee di J. C. Lee, molti appassionati hanno scritto applicazioni e librerie in molti linguaggi di programmazione. Nel gennaio 2011, un gruppo di programmatori, docenti, collaboratori accademici, supportati dal Laboratorio Innovazione Tecnologica Supporto Apprendimento (LITSA) dell'Università di Trento, crea WiildOs, un sistema operativo Open Source per lavagne multimediali interattive. [1]

Guida alla lettura

Scopo di questa trattazione è quello di fornire al lettore le basi teoriche per replicare l'esperimento e/o modificarlo, nonché di mostrare il percorso che ha portato al conseguimento dell'obiettivo prefissato.
Nel primo capitolo vedremo l'infrastruttura generale del progetto ed il modo in cui lavorano i vari moduli.
Il secondo capitolo mostra le basi teoriche per poter comprendere il progetto dal punto di vista della comunicazione Bluetooth tra la macchina che esegue lightboard ed il Wiimote.
Il terzo capitolo fornisce una visione generale sul protocollo grafico X e sulle librerie grafiche utilizzate nel progetto.
Dal quarto capitolo inizia il lavoro effettuato su lightboard, prima di tutto viene fatta un'analisi delle prestazioni di python-whiteboard al fine di individuare i punti deboli, e quindi comprendere su cosa focalizzare l'attenzione durante la realizzazione.
Il quinto capitolo è dedicato proprio alla realizzazione, allo studio ed alla spiegazione del funzionamento di python-whiteboard e della libreria utilizzata come driver del Wiimote: *libcwiid*.
Il sesto capitolo contiene i dati dei benchmark effettuati su lightboard in alcune fasi della realizzazione, infine li pone a confronto con i dati di python-whiteboard.
In appendice A troviamo la guida utente e la documentazione tecnica, que-

st'ultima è indirizzata ai programmatori che vogliono modificare l'applicazione.
Il codice utilizzato è fornito con licenza GPL ed è reperibile al seguente URL:
`https://github.com/GiovanniIncammicia/lightboard`

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Infrastruttura

Il *Wii Remote* (*Wiimote*) è il principale controller per la console Nintendo Wii. È un dispositivo senza fili che usa la tecnologia standard Bluetooth per comunicare con la Wii.

È costruito intorno ad un *Broadcom BCM2042 System-on-chip Bluetooth*. [25] Una delle funzionalità principali del Wiimote è il rilevamento del movimento, che permette all'utente di interagire e manipolare oggetti sullo schermo attraverso il riconoscimento dei gesti e puntare attraverso l'uso dell'accelerometro e del sensore ottico. [24]

Figura 1.1: Wiimote

A questo scopo è dotato di una fotocamera ad infrarossi monocromatica con una risoluzione di 128x96.
Il circuito interno di elaborazione delle immagini della telecamera può tracciare fino a quattro punti. Il processore integrato usa un'analisi sub-pixel 8x, che fornisce una risoluzione di 1024x768 ai punti monitorati. Il campo visivo del Wiimote è di 33 gradi orizzontali e 23 gradi verticali e può individua-

re sorgenti fino a 850nm (anche se con un'intensità dimezzata rispetto alle sorgenti a 940nm). [25]

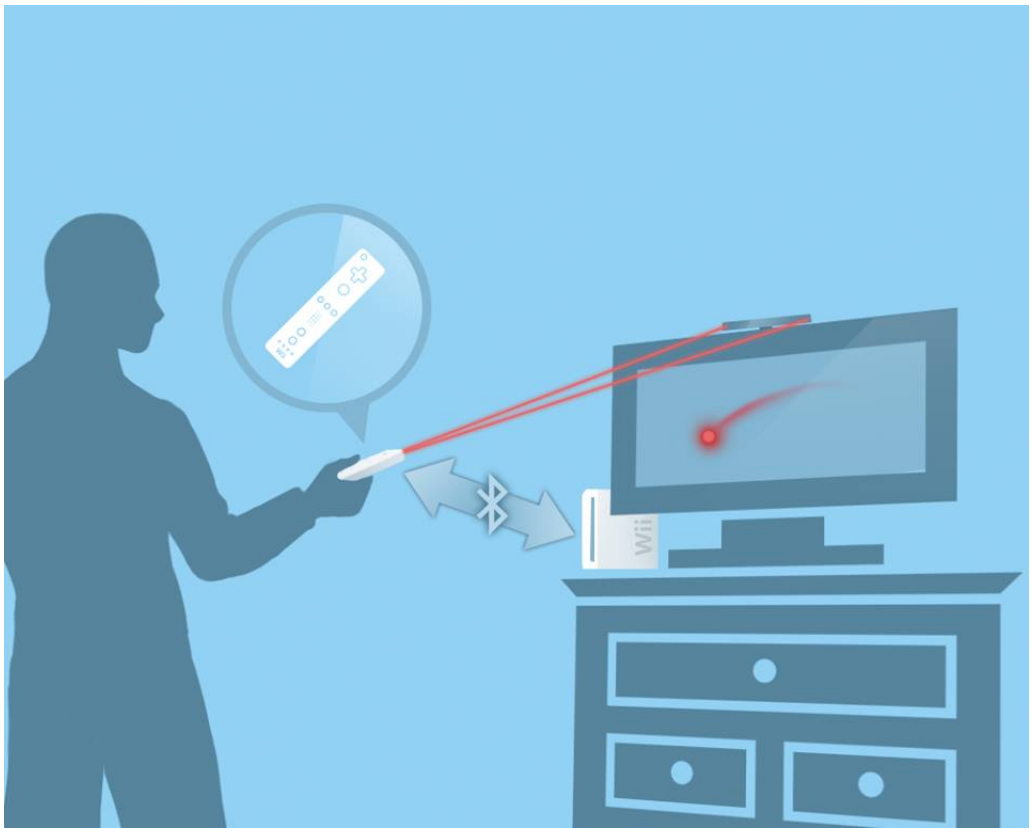

Figura 1.2: Infrastruttura hardware originale

In figura 1.2 è descritto il funzionamento originale del Wiimote.
Vediamo come il Wiimote rilevi i segnali IR trasmessi dalla *sensor bar* della Wii (una barra che si può posizionare sopra o sotto il televisore, munita di 10 led ad infrarossi). Il Wiimote è collegato via Bluetooth alla Wii alla quale invia i dati rilevati, la Wii li elabora e, per triangolazione, riesce a capire dove si trova il Wiimote.

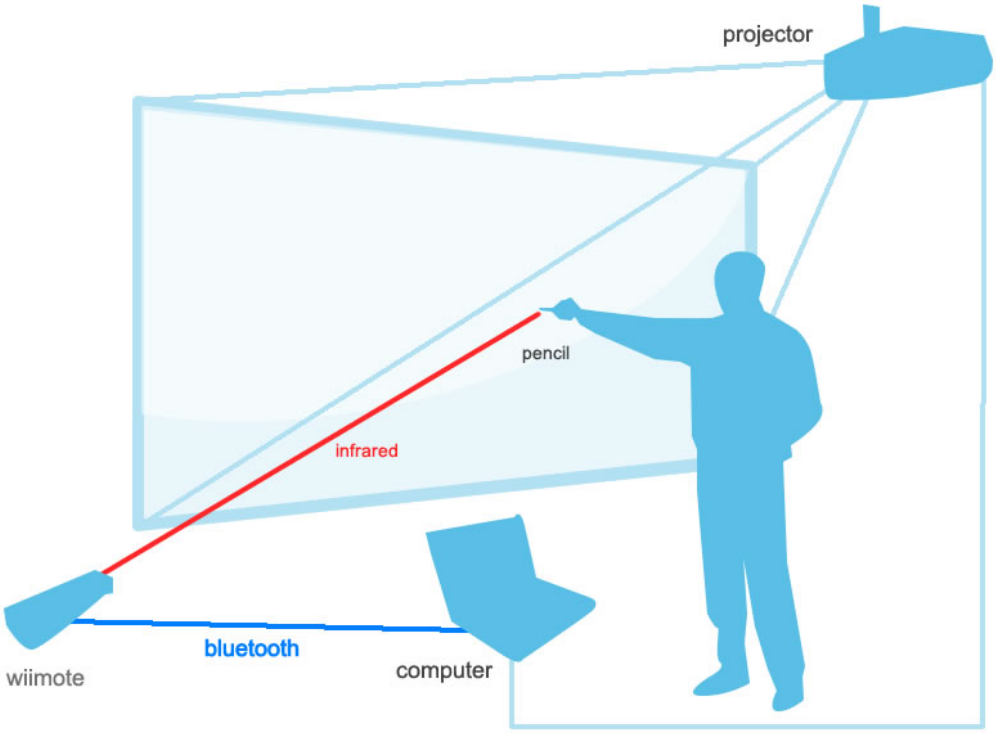

Figura 1.3: Infrastruttura hardware LIM

1.1 Funzionamento di Lightboard

In figura 1.3 invece è descritto il funzionamento del Wiimote per creare una lavagna multimediale interattiva (LIM). In questo caso il Wiimote è fermo e punta allo schermo, mentre la penna ad infrarossi si muove. Durante la fase iniziale si effettua la calibrazione, nella quale dovremo puntare la penna ad infrarossi in quattro punti agli angoli dello schermo; questo serve al Wiimote per capire dove si trova (rispetto allo schermo proiettato).

Da questo momento, ogni movimento della penna verrà individuato dal Wiimote, ed il programma sull'host si occuperà di fare alcuni calcoli (che vedremo in seguito) per capire l'effettiva posizione della penna sullo schermo.

Nel diagramma di deployment in figura 1.4 è illustrata l'infrastruttura del software.

La *user interface* (interfaccia utente) è la stessa di python-whiteboard, riusa-

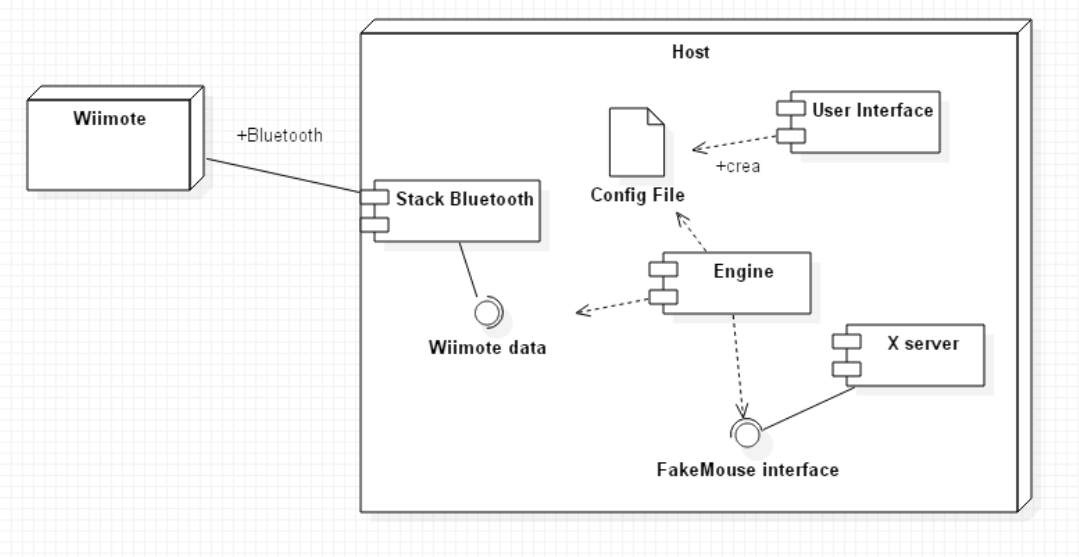

Figura 1.4: Diagramma di deployment lightboard

ta per permettere all'utente di configurare il software senza dover interagire con il terminale.
Successivamente tutte le configurazioni e la calibrazione vengono salvate in un file chiamato *.lbconfig.ini* nella cartella *ui/stuff*. Dopo aver premuto il pulsante *Activate*, entra in gioco l'*engine* scritto in C, il motore di elaborazione di lightboard. Esso legge il file, si connette al Wiimote ed inizia a ricevere ed elaborare i dati della telecamera IR muovendo il mouse grazie a *XTest Extension*, un'estensione minimale della più famosa libreria XLib.

1.2 Conversione coordinate

Vediamo ora come fa il software a tradurre le coordinate spaziali, prese dalla telecamera del Wiimote, in coordinate dello schermo.

Problema

Marcare quattro punti su una superficie proiettata, che corrispondono a quattro punti di un quadrato (coordinate dello schermo).

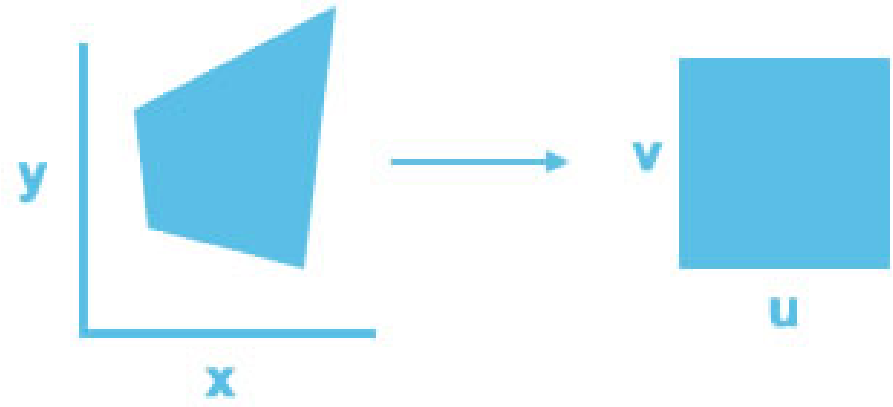

Figura 1.5: Problema calibrazione

Soluzione

Nel nostro caso abbiamo i valori `u` e `v` e dobbiamo trovare `x` e `y`.
Vediamo adesso il procedimento formale, in seguito vedremo come esso è

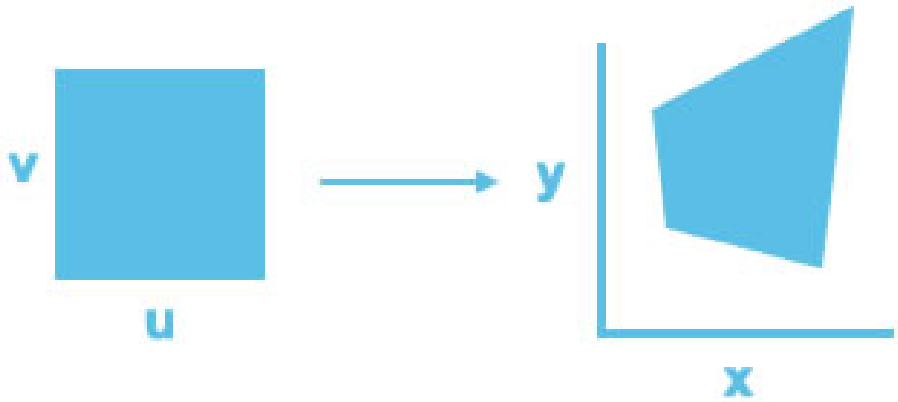

Figura 1.6: Soluzione calibrazione

implementato in lightboard.

$$\langle x', y', w \rangle$$

$$x = \frac{x'}{w} \quad y = \frac{y'}{w} \tag{1.1}$$

$$\langle x', y', w\rangle = \langle u, v, 1\rangle \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & 1 \end{bmatrix} \tag{1.2}$$

Dalle formule 1.1 e 1.2 otteniamo le seguenti:

$$x = \frac{au + bv + c}{gu + hv + 1} \tag{1.3a}$$

$$y = \frac{du + ev + f}{gu + hv + 1} \tag{1.3b}$$

Eseguendo i calcoli dalle formule 1.3 troviamo:

$$x = au + bv + c - xgu - xhv \tag{1.4a}$$

$$y = du + ev + f - ygu - yhv \tag{1.4b}$$

Usando i quattro angoli dello schermo abbiamo il seguente sistema di equazioni:

$$\begin{bmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0x_0 & -v_0x_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1x_1 & -v_1x_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2x_2 & -v_2x_2 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3x_3 & -v_3x_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{1.5}$$

Nella sezione 5.3.3 vedremo un esempio pratico del calcolo.[1] [15]

[1] Per trovare $\langle u, v\rangle$ partendo da $\langle x, y\rangle$ bisogna trovare la matrice inversa

Capitolo 2

Bluetooth

Bluetooth è una tecnologia senza fili standard per lo scambio di dati su breve distanze che utilizza onde radio UHF (*ultra high frequency*) nella banda ISM (*Industral Scientific and Medical*) da 2.4 a 2.485 GHz.

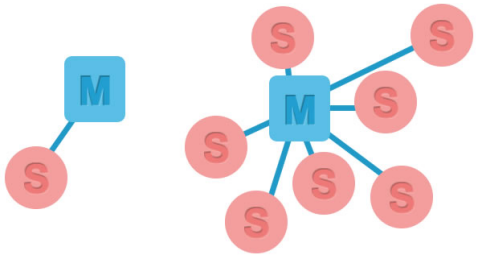

Figura 2.1: Piconet Bluetooth

Bluetooth è un protocollo a pacchetto con una struttura master-slave. Un master può comunicare con massimo sette slave in una piconet.

Profili Bluetooth

I profili Bluetooth sono protocolli aggiuntivi costruiti sulle basi dello standard Bluetooth. Mentre le specifiche di quest'ultimo definiscono come funziona la tecnologia, i profili definiscono come essa viene usata. [2]
Per usare la tecnologia Bluetooth, un dispositivo deve essere in grado di interpretare alcuni profili.
Ogni specifica di profilo contiene informazioni su:

- le dipendenze da altri formati;

- i formati di interfaccia utente consigliati;
- parti specifiche dello stack Bluetooth usati dal profilo. Per adempiere al proprio compito, ogni profilo usa particolari opzioni e parametri ad ogni livello dello stack.

2.1 Stack del protocollo Bluetooth

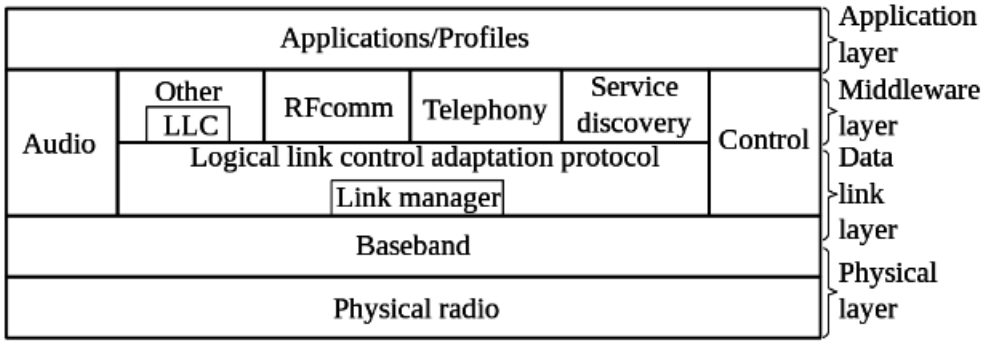

Figura 2.2: Stack Bluetooth

Bluetooth è definito con un'architettura a livelli che consiste di protocolli centrali (*core*), di collegamento (*cable replacement*), di controllo della telefonia (*telephony control*) e adottati. I protocolli obbligatori per ogni stack Bluetooth sono: LMP, L2CAP e SDP. In aggiunta, i dispositivi che comunicano via Bluetooth possono universalmente utilizzare questi protocolli: HCI e RFCOMM. [5]

Lo standard si può dividere in due parti: uno *stack controller*, che contiene le interfacce radio, ed uno *stack host*, che utilizza dati di alto livello. Lo stack host è generalmente implementato come parte di un sistema operativo o come pacchetto installabile.

Stack controller

- LMP

Stack host

- L2CAP
- SDP [17]

Link Manager Protocol (LMP)

Serve per impostare e controllare il collegamento radio tra due dispositivi.

Logical Link Control and Adaptation Protocol (L2CAP)

Serve per fare il *multiplexing* di più connessioni logiche tra due dispositivi che usano diversi protocolli di livello superiore (permette segmentazione e riassemblamento dei pacchetti).
In *Basic mode* fornisce pacchetti con *payload* fino a 64KB e MTU di default a 672 byte e 48 byte di minimo obbligatorio.
In *Retransmission and flow control mode*, può essere configurato sia per dati isocroni, sia per dati affidabili eseguendo ritrasmissione e CRC. Sono state poi aggiunte due modalità che rendono obsoleta l'ultima.
In *Enhanced retransmission mode*, è stata migliorata la modalità originale (canale affidabile).
In *Streaming mode*, non c'è ritrasmissione o controllo del flusso (canale inaffidabile).

Service Discovery Protocol (SDP)

Permette ad un dispositivo di trovare servizi offerti da altri dispositivi ed i loro parametri.
Permette anche all'host di determinare quale profilo Bluetooth può usare il dispositivo con cui sta cercando di connettersi.

2.2 Human Interface Device Profile (HID)

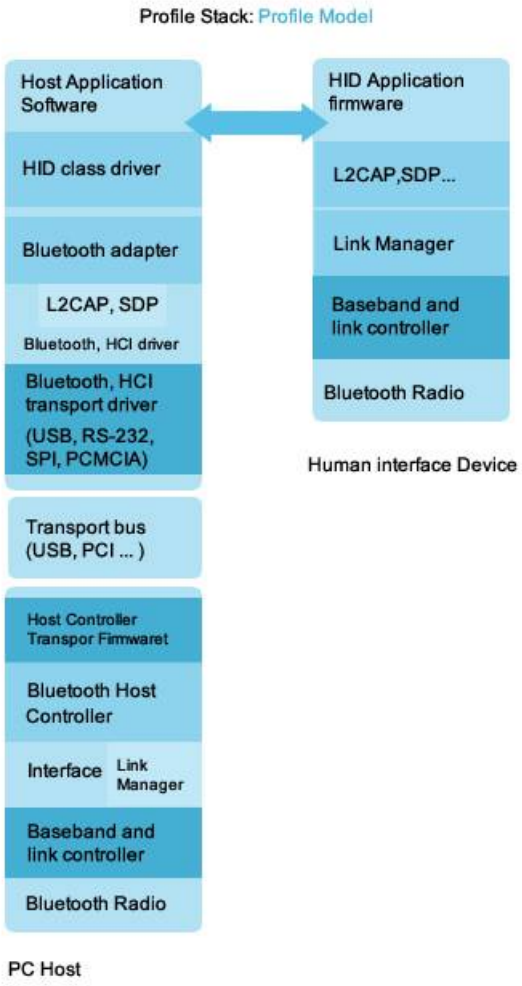

Figura 2.3: Stack HID

Wiimote usa il profilo standard Bluetooth HID (BT-HID) per comunicare con l'host, il quale è basato sullo standard USB HID. [25] La classe USB HID è una parte della specifica USB per le periferiche per computer; essa specifica una classe di dispositivi che si interfacciano con l'uomo, come tastiere, mouse, controller per giocare ecc. [23] Bluetooth HID è progettato per garantire un collegamento a bassa latenza e basso consumo energetico. [16] In figura 2.3 vediamo lo stack del profilo BT-HID che mostra i livelli software che risiedono nell'host (master) e nell'HID (slave), HID funziona direttamente sul protocollo L2CAP e non usa altri protocolli Bluetooth oltre a LMP e SDP. Nell'esempio in figura 2.3 l'host è un PC e sul processore sono eseguiti i livelli superiori del software Bluetooth e, attraverso un bus di trasporto come USB, è collegato ad un modulo radio Bluetooth.

L'HID ha il firmware integrato con quello della radio, eseguiti sulla stessa CPU per il minor costo di implementazione possibile. Altre implementazioni sono possibili ed ugualmente valide. [11]

In figura 2.4 vediamo un esempio di interfaccia BT-HID.

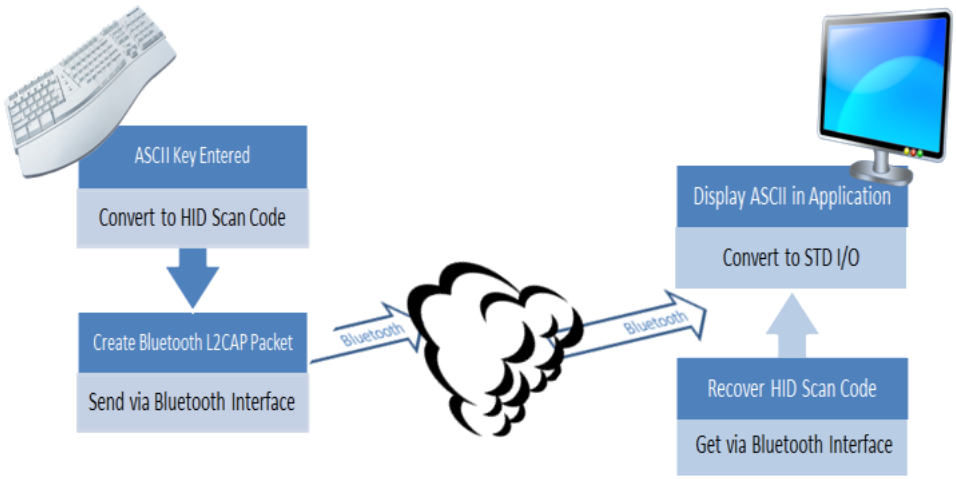

Figura 2.4: Interfaccia BT-HID

2.2.1 Iniziare una connessione

Ogni dispositivo Bluetooth in *discoverable mode* trasmette le seguenti informazioni su richiesta:

- nome del dispositivo,
- classe del dispositivo,
- lista di servizi,
- informazioni tecniche (funzionalità, fabbricante, specifiche Bluetooth usate).

Ogni dispositivo può effettuare un'indagine (*inquiry*) per trovare altri dispositivi a cui connettersi, ogni dispositivo può essere configurato per rispondere a queste richieste.
Se il dispositivo che cerca di connettersi conosce l'indirizzo dell'altro, quest'ultimo risponde ad una richiesta di connessione diretta ed eventualmente trasmette le informazioni mostrate nella lista sopra. L'uso dei servizi di un

dispositivo potrebbe richiedere un appaiamento (*pairing*) o l'accettazione da parte del proprietario, ma la connessione semplice può essere inizializzata da ogni dispositivo e rimane finché non si esce dal raggio. Alcuni dispositivi possono effettuare una sola connessione, in tal caso non compariranno nelle indagini finché non si disconnettono.
Ogni dispositivo ha un indirizzo univono a 48 bit, questi indirizzi generalmente non vengono mostrati nelle indagini, al loro posto si usano dei nomi impostati dagli utenti.

2.2.2 Appaiamento

Motivazioni

Molti dei servizi offerti su Bluetooth possono esporre dati privati o permettere alla parte connessa di controllare il dispositivo Bluetooth. Per motivi di sicurezza è necessario essere in grado di riconoscere specifici dispositivi e controllare per quali di essi è permessa una connessione.
Allo stesso modo è utile per i dispositivi Bluetooth poter stabilire una connessione senza l'intervento dell'utente (ad esempio quando entrano nel raggio Bluetooth).

Processo

Per risolvere questi conflitti, Bluetooth usa un processo chiamato *bonding*, un legame (bond) è generato tramite un processo di appaiamento. Il processo di appaiamento è attivato da una specifica richiesta dall'utente di generare un legame (ad esempio se l'utente esplicitamente richiede di aggiungere un dispositivo Bluetooth), oppure è attivata automaticamente quando ci si connette ad un servizio dove l'identità del dispositivo è richiesta per motivi di sicurezza.
A questi due casi ci si riferisce rispettivamente come *bonding dedicato* e *bonding generale.* [5]

2.3 Wiimote

2.3.1 Connettere il Wiimote

Il Wiimote può essere messo in discoverable mode premendo i bottoni 1 e 2 insieme. In discoverable mode, un numero di led da uno a quattro, in base al livello di batteria, iniziano a lampeggiare. Durante l'indagine da parte dell'host verranno trovati tutti i Wiimote in discoverable mode. Ora l'host può stabilire una connessione di banda base Bluetooth (*baseband*), non è richiesto l'appaiamento. Dopo che una connessione di banda base Bluetooth è stabilita (con o senza appaiamento), i canali HID possono essere aperti ed usati per leggere e scrivere *report* dal/verso il Wiimote. La banda base Bluetooth è la parte del sistema che specifica o implementa l'accesso al mezzo e le procedure di livello fisico tra dispositivi Bluetooth. Si occupa delle specifiche della piconet tra il livello fisico (radio) e quello di collegamenti dati. [6]
Stabilire una connessione HID con il Wiimote può essere fatto sul PSM `0x11` per il canale di controllo e sul PSM `0x13` per il canale dei dati utilizzando il protocollo L2CAP. I PSM (Protocol Service Multiplexer), sono l'analogo delle porte utilizzate nel TCP, ma utilizzano i numeri dispari nel range 1-32767. [8] Lo standard HID permette ai dispositivi di auto descriversi usando il *report descrittore* HID. Questo report include un'enumerazione di report che il dispositivo comprende.
I report sono unidirezionali, ed il descrittore HID elenca per ogni porta, la direzione (input/output) ed il payload. Come tutti i dispositivi BT-HID, il Wiimote manda il report descrittore quando richiesto tramite il protocollo SDP. Comunque non riporta nessuna informazione riguardo i dati all'interno dei report, solo la lunghezza in byte (rendendo così inutile l'utilizzo dei driver standard HID).

Un report di input è mandato dal Wiimote all'host, un report di output è mandato dall'host al Wiimote. I primi vengono letti sul canale dei dati, che è lo stesso sul quale vengono mandati i secondi. Il canale di controllo può

essere utilizzato, con il Wiimote classico, per i report di output. [25]

2.3.2 Latenza e prestazioni

Il requisito di prestazione essenziale è che BT-HID garantisca una reattività ai cambiamenti in input simile a quella dei dispositivi cablati.
Le applicazioni ad alte prestazioni, come i videogiochi, richiedono un tempo di risposta dalla pressione del bottone o dal movimento del joystick nell'ordine di una singola finestra video (*frame*), ovvero 16,7ms per un aggiornamento video a 60Hz.
L'implementazione del collegamento Bluetooth dovrebbe aggiungere non più di 5ms di latenza rispetto ad un'implementazione cablata. [11]

Capitolo 3

X Window System

Anche conosciuto come X11, X Window System è un protocollo grafico comune sui sistemi operativi UNIX-like che consente la creazione di un sistema client/server di interazione, anche remota, con programmi grafici. X fornisce le basi per un ambiente GUI: disegnare e muovere finestre su un display e interagire con un mouse ed una tastiera. [27]
X Window System utilizza una terminologia fuorviante per i concetti di client e server.

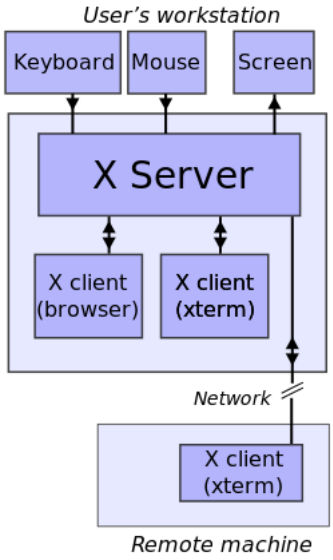

Figura 3.1: X11: Un semplice esempio

Server X

Un server X è un'applicazione che risiede sul computer che ha la tastiera, il monitor ed il mouse connessi. Le responsabilità del server includono la gestione dello schermo, la gestione degli input da tastiera e dal mouse e la gestione degli input/output che provengono da altri dispositivi. Questo

confonde alcune persone perché si aspettano che il server X sia una macchina grande e potente e che il client X sia invece la macchina sulla loro scrivania.

Client X

Ogni applicazione X, come XTerm o Firefox, è un client. Un client manda messaggi al server per eseguire operazioni come: disegnare una finestra in certe coordinate, fare click con il mouse ecc. Il server invece manda indietro risposte che includono eventi quali ad esempio il click di un bottone da parte di un utente. [21]

In figura 3.1 vediamo un semplice esempio dove il server X riceve un input da una tastiera ed un mouse locali e mostra l'output su uno schermo. Un browser ed un emulatore di terminale sono eseguiti su una stazione di lavoro utente ed un emulatore di terminale è eseguito su un computer remoto, ma controllato e monitorato dalla macchina utente.

3.1 XTest Extension Library

Questa estensione è un insieme minimale di estensioni per client e server per testare il server X11 senza intervento dell'utente.

Descrizione

Le funzioni fornite da questa estensione ricadono in due gruppi:

- **Operazioni client** - queste applicazioni manipolano il comportamento lato client che altrimenti sarebbe nascosto.
- **Richieste server** - queste richieste permettono di accedere a risorse del server altrimenti in sola lettura (ad esempio il cursore associato ad una data finestra), oppure permettono una sintesi limitata degli eventi dei dispositivi di input (ad esempio muovere il dispositivo di puntamento o premere un bottone). [10]

3.2 Framework Qt

Qt (cute) è un framework di applicazioni multi-piattaforma.
I programmi creati con Qt possono avere un'interfaccia grafica nativa, in questo caso è classificato come un *widget toolkit*.
Non è però limitato alle interfacce grafiche, è anche usato per sviluppare programmi senza interfaccia grafica, come strumenti a linea di comando e terminali per server.

3.2.1 Concetti base di Qt

- **Astrazione completa dall'interfaccia grafica** - quando è stato rilasciato, Qt usava il proprio motore grafico ed i propri controlli, rendendolo così indipendente dalla piattaforma; questo portava occasionalmente a discrepanze laddove l'emulazione era imperfetta. Le versioni recenti di Qt usano le API native delle differenti piattaforme.

- **Segnali e slot** - un costrutto introdotto in Qt per la comunicazione tra oggetti che permette di implementare facilmente il pattern *Observer*.

- **Compilatore di meta-oggetti** - il compilatore di meta-oggetti è uno strumento che funziona sul sorgente di un programma Qt. Esso interpreta alcune macro dal codice C++ e le usa per aggiungere codice con meta-informazioni sulle classi usate nel programma. Queste meta-informazioni sono usate da Qt per fornire funzionalità non disponibili nativamente nel C++ (cioè segnali e slot, introspezione e chiamate di funzione asincrone). [19]

3.2.2 PyQt4

Qt non è disponibile soltanto per i programmi in C++, sono stati sviluppati infatti molti moduli che implementano le funzionalità di Qt in altri linguaggi di programmazione.

Prenderemo in esame un particolare esempio, PyQt, utilizzato in python-whiteboard. PyQt comprende numerosi componenti. Primi fra tutti ci sono alcuni moduli di estensione di Python, tra quelli utilizzati in python-whiteboard troviamo:

- **QtCore** - contiene il nucleo delle classi senza interfaccia grafica, inclusi il ciclo di eventi ed il meccanismo di segnali e slot di Qt;
- **QtGui** - contiene la maggior parte delle classi per l'interfaccia grafica. [3]

Capitolo 4

Ipotesi di lavoro

I principali problemi di python-whiteboard riguardano il consumo di risorse e, di conseguenza, le prestazioni generali del programma.
Python è un ottimo linguaggio di programmazione ad alto livello, intuitivo ed elegante, che permette di scrivere programmi compatti e meno dispersivi rispetto al C, però è meno indicato per eseguire operazioni di basso livello, in questi casi risulta più lento del C, che invece proprio in questi casi mostra la sua forza e flessibilità.
Questo è dovuto al fatto che Python è un linguaggio semi-interpretato, mentre C è compilato (il compilatore gcc inoltre offre ottimi strumenti di ottimizzazione del codice). Proprio per questo python-whiteboard offre prestazioni scadenti rispetto alle lavagne multimediali in commercio. Da questi presupposti, si è pensato di lasciare la comunicazione con il Wiimote e l'elaborazione del segnale IR (operazioni di più basso livello) ad un programma scritto in C e mantenere l'interfaccia grafica scritta in Python.

4.1 Risorse richieste da python-whiteboard

Abbiamo analizzato il tempo di elaborazione di un singolo segnale, al fine di confrontarlo con lo stesso, elaborato in linguaggio C. Quello che python-whiteboard fa è:

1. ricevere il segnale IR,

2. trasformarlo in coordinate dello schermo,

3. spostare il cursore in quelle coordinate.

È stato registrato il tempo medio che passa tra un segnale ed il successivo per avere una stima di quanti segnali arrivano al secondo al processo. Il risultato, con un campione di 300 segnali, è di 0,009822 secondi, che si traduce in 101,81 segnali al secondo, una frequenza approssimata di 102Hz. La massima frequenza di report di dati per i device BT-HID non dovrebbe superare i 125Hz. [11] Chiaramente questo numero è basso rispetto ai segnali che il processo può elaborare in un secondo, infatti il tempo medio, con un campione di 1000 segnali, che python-whiteboard impiega da quando arriva un segnale, a quando il mouse viene spostato è di 0,001167 secondi.
Il margine dunque è molto ampio, python-whiteboard potrebbe elaborare i segnali di 9 Wiimote contemporaneamente senza congestionare lo stack di rete Bluetooth. Questo dato non è così sorprendente, tuttavia sappiamo che la latenza dei collegamenti via cavo USB è di 1ms [22], i dispositivi Bluetooth HID non dovrebbero aggiungere più di 10ms alla latenza dei collegamenti via cavo. [11] Python-whiteboard aggiunge 1,16ms alla latenza, se sommiamo tutti i ritardi troviamo che, dal momento in cui si accende il led sulla penna ad infrarossi, al momento in cui viene spostato il mouse, passano approssimativamente 12,16ms.

Altri due fattori da analizzare per testare le prestazioni di python-whiteboard sono il consumo di CPU medio durante l'elaborazione del segnale IR o in stato di riposo (IDLE), ed il consumo di RAM, anch'esso nei due casi precedenti. Questo dato è per noi più indicativo in quanto ci dice che impatto potrà avere il software su un hardware con poche risorse.

4.1.1 Test effettuati su Asus X53S

- **Processore** - Intel Core i7-2670QM 2,20GHz x 8

- **RAM** - 3,6 GB
- **OS type** - 64 bit
- **OS** - Ubuntu 14.10

	Elaborazione IR	**IDLE**
Consumo CPU (%)	95,9	42
Consumo RAM (MB)	148,96	145,58

Tabella 4.1: Test effettuati su Asus X53S

4.1.2 Test effettuati su Raspberry Pi Model B

- **Processore** - 700MHz Low Power ARM1176JZ-F Applications Processor
- **RAM** - 512 GB
- **OS type** - 32 bit
- **OS** - Raspbian December 2014

	Elaborazione IR	**IDLE**
Consumo CPU (%)	82,2	60,9
Consumo RAM (MB)	51	51

Tabella 4.2: Test effettuati su Raspberry Pi

Impressioni sull'utilizzo

Sulla prima macchina, il programma funziona abbastanza bene, il cursore segue la penna IR con ritardo, ma è fluido ed utilizzabile.
Su Raspberry Pi invece è completamente inutilizzabile, il ritardo è di circa 5 secondi.
Questi dati rendono evidente la necessità di creare una versione *lightweight* dei programmi per *whiteboard* esistenti.

Capitolo 5

Analisi e realizzazione

Di seguito verrà descritto il processo di analisi delle varie librerie ed applicazioni utilizzate nel progetto e la sua successiva realizzazione.
L'analisi segue il modello *top-down*, si sono quindi analizzati i componenti partendo dal livello più alto, quello dei moduli e delle interazioni tra essi, passando per le sequenze di attivazione delle funzioni e routine, fino ad analizzare il comportamento interno delle singole procedure. Durante questo processo sono stati prodotti alcuni diagrammi di progettazione utilizzando il linguaggio di modellazione UML, alcuni di questi verranno analizzati in questo capitolo, altri, più tecnici e specifici, saranno analizzati nella documentazione in Appendice A.

5.1 Studio di Cwiid

Abbiamo visto nella sezione 2.3.1 che il Wiimote non fa uso dei tipi di dato standard e dei descrittori HID, descrive solo la lunghezza del report, lasciando il contenuto indefinito.
La prima problematica da affrontare è stata dunque scegliere il driver per Wiimote da utilizzare nel progetto.
In merito sono stati valutati i driver per Wiimote più utilizzati, e quindi con

più esempi e documentazione; per quanto riguarda la programmazione C ne troviamo due: wiiuse e cwiid.

Wiiuse

Wiiuse è una libreria che permette di connettersi a più Wiimote. Supporta l'accelerometro, la rilevazione IR ed alcune estensioni.
La libreria è gestita da un singolo thread ed è non-bloccante, il che la rende un'API leggera e pulita.

Cwiid

Cwiid è una collezione di strumenti per Linux scritti in C per interfacciarsi con il Wiimote, contiene i seguenti strumenti:

- una libreria (libcwiid),
- un driver per utilizzare il Wiimote come un joystick (wminput),
- un'applicazione di test/demo (wmdemo),
- un'interfaccia grafica per gestire i dati ricevuti dal Wiimote (wmgui). [26]

La libreria è gestita da più thread, il che la rende un'API performante e modulare. Dopo aver analizzato entrambe le librerie, è stata scelta libcwiid per la presenza di numerose applicazioni di esempio che avrebbero reso più scorrevole lo studio della libreria; una rilevanza fondamentale nella scelta l'hanno avuta anche la sua natura modulare ed il multi-threading.

5.1.1 Analisi statica

Libcwiid è costituita di vari moduli, qui li analizziamo brevemente, verranno trattati approfonditamente nella documentazione in Appendice A. Dal diagramma dei componenti in figura 5.1 vediamo che la libreria è composta di numerosi moduli, lightboard interagisce con ogni modulo, di seguito vediamo per quali scopi e come.

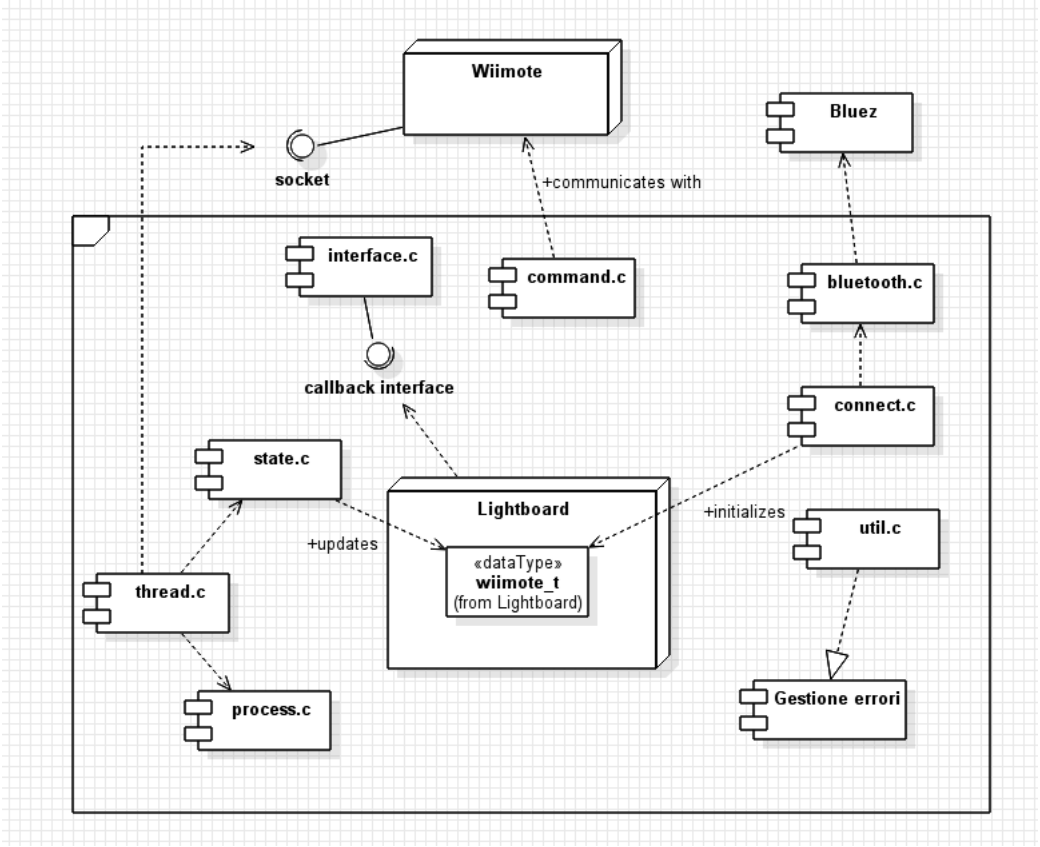

Figura 5.1: Diagramma dei componenti di libcwiid

bluetooth.c

Questo modulo si occupa della connessione al Wiimote utilizzando il protocollo Bluetooth, permette di cercare dei dispositivi e connettersi ad essi, utilizza la libreria Bluetooth standard e lo stack bluez.

connect.c

Questo modulo si occupa della connessione al Wiimote a livello di processo; dopo essersi connessi utilizzando il protocollo Bluetooth, bisogna creare i canali di comunicazione (socket) sui PSM `0x11` e `0x13` come visto nella sezione 2.3.1. Permette inoltre di creare ed inizializzare la struttura `wiimote_t` che conterrà tutte le informazioni relative al Wiimote connesso.

command.c

Questo modulo si occupa di parlare direttamente al Wiimote, permettendo così all'applicazione di specificare i comandi da dare al dispositivo, tra i comandi principali troviamo la richiesta dello stato del Wiimote e l'impostazione delle modalità di report.

state.c

Questo modulo permette di aggiornare lo stato in modo da tener traccia di tutte le impostazioni del Wiimote, di quelle di inizializzazione e di quelle date dall'applicazione tramite il modulo command.c, salvando il tutto nella struttura `wiimote_t`.

interface.c

Questo modulo fornisce all'applicazione l'interfaccia per elaborare i messaggi ricevuti dal Wiimote, permette di abilitare e disabilitare l'elaborazione dei messaggi, nonché di impostare la funzione di callback che si occupa di elaborare tali messaggi.

thread.c

Questo modulo definisce i thread che andranno a costituire l'infrastruttura finale del programma, i thread utilizzati sono tre (escludendo il main):

- **thread router** - è il thread principale, si occupa di leggere dal socket dei dati (interrupt) i messaggi ricevuti dal Wiimote, processarli e smistarli alle funzioni di callback. I messaggi di stato vengono smistati al thread status, mentre i messaggi che contengono dati vengono smistati al thread mesg_callback;

- **thread status** - si occupa di leggere dalla pipe dedicata allo stato del Wiimote che viene riempita quando esso viene modificato e di gestire il

cambiamento aggiornando la struttura `wiimote_t` attraverso il modulo state.c;

- **thread mesg_callback** - si occupa di leggere dalla pipe dedicata ai messaggi del Wiimote. Questo thread chiama la funzione di callback definita dal programmatore, dando così massima flessibilità di personalizzazione del comportamento.

process.c

Questo modulo processa i messaggi ricevuti dal thread router in una forma più facilmente leggibile dalle funzioni di callback che si occuperanno di elaborarli.

util.c

Questo modulo contiene alcune funzioni di utilizzo generale, tra cui anche la gestione degli errori.

5.1.2 Analisi dinamica

Una volta fatta un'analisi statica della libreria e delle sue funzioni, un'analisi dinamica della stessa ha permesso di capire la sequenza di attivazione delle funzioni ed il loro funzionamento. Il processo si divide in tre fasi distinte:

1. inizializzazione,
2. abilitazione dei segnali IR,
3. lettura dei segnali IR.

Inizializzazione

In questa fase si effettua la connessione al Wiimote e si richiede lo stato per la prima volta, così da inizializzare la struttura `wiimote_t`.

Qui entrano in gioco il thread router che leggerà lo stato dal socket di interrupt, lo processerà tramite il modulo process.c e lo invierà al thread status tramite la pipe di stato.
Infine si imposta la funzione di callback definita dal programmatore.
Finita questa fase saranno attivi tutti i moduli e l'applicazione potrà iniziare la sua routine utilizzando la struttura `wiimote_t` inizializzata, come riferimento al Wiimote.

Abilitazione dei segnali IR

In questa fase si abilita il Wiimote a mandare i report con i dati della telecamera IR.
La libreria libcwiid permette di abilitare anche altri tipi di report, come i dati dell'accelerometro o del microfono, che tuttavia non ci interessano per l'applicazione.
La sequenza per abilitare i segnali IR consiste di sei punti, in ognuno dei quali bisogna inviare un messaggio al Wiimote ed aspettare la risposta, anche qui il thread router si occupa di ricevere, processare e smistare i messaggi, stavolta su una pipe dedicata a questo: la pipe rw (read/write).
Qui il modulo state.c fornisce l'interfaccia per abilitare i segnali IR sul Wiimote, il modulo process.c, chiamato dal thread router, si occupa di iniziare la sequenza che sarà effettuata da una funzione del modulo util.c.
Finita questa fase inizieranno ad arrivare, sul canale degli interrupt, i messaggi relativi ai dati IR raccolti dalla telecamera ad infrarossi del Wiimote.

Lettura dei segnali IR

A questo punto il thread principale può mettersi in attesa dell'input da parte dell'utente, che servirà a chiudere l'applicazione, il resto viene fatto dal thread router, che riceverà i segnali IR e li manderà alla funzione di callback.
I dettagli relativi alla funzione di callback saranno spiegati in seguito, mentre un'analisi dettagliata della sequenza di operazioni qui introdotte, verrà fatta nella documentazione in Appendice A.

5.2 Studio di python-whiteboard

WiildOs

> WiildOs non è un semplice programma per la gestione della tua lavagna interattiva multimediale, ma un nuovo sistema operativo modulare, semplice, completo, libero, open source e gratuito. [1]

WiildOs è costruito su Ubuntu selezionando software adatti all'educazione ed all'apprendimento tramite il computer e le LIM.
Esso utilizza python-whiteboard, un'applicazione che permette di trasformare ogni schermo proiettato, in una LIM.
Per questo motivo abbiamo scelto python-whiteboard come punto di partenza per il progetto, analizzando i suoi punti di forza e le sue debolezze.

Python-whiteboard

Python-whiteboard è un software open source scritto in Python e sviluppato da Pere Negre, è compatibile con sistemi GNU/Linux.
In questo capitolo faremo, come nel caso di libcwiid, due tipi di analisi: statica e dinamica.
Per un'analisi più completa si rimanda alla documentazione in Appendice A.

5.2.1 Analisi statica

Ancora una volta utilizziamo un diagramma dei componenti per descrivere la struttura generale di python-whiteboard.

pywhiteboard.py

Questo è il modulo centrale, qui viene creata l'istanza della finestra principale del programma, da cui si accede alla schermata di configurazione ed alla calibrazione, nonché ad alcune informazioni generali sul Wiimote connesso.

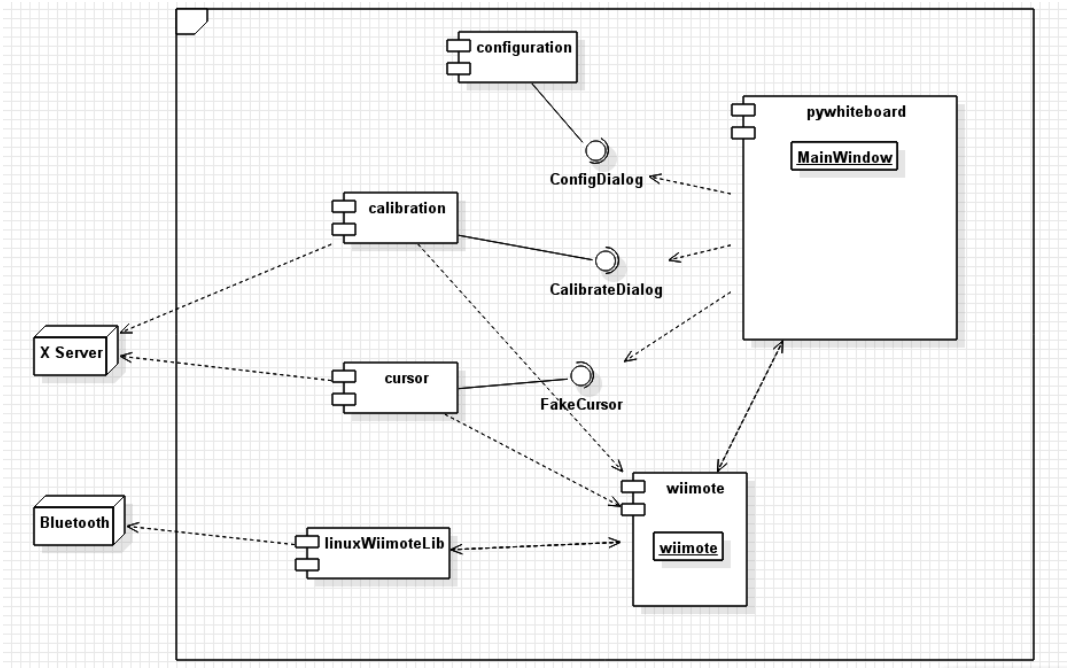

Figura 5.2: Diagramma dei componenti di python-whiteboard

Questo modulo permette di gestire anche i profili, disconnettere il Wiimote, o passare dalla modalità di solo puntamento al click e viceversa.

configuration.py

Questo modulo fornisce l'interfaccia per accedere al pannello di configurazione, dal quale potremo modificare le impostazioni relative al programma o al Wiimote.

calibration.py

Questo modulo permette di gestire la calibrazione del Wiimote, utilizza una comoda schermata con quattro punti agli angoli dello schermo. Bisogna puntare ogni angolo in successione per permettere al programma di mappare i segnali IR rilevati dal Wiimote, in coordinate dello schermo.

cursor.py

Questo è il modulo che gestisce il cursore, interagisce direttamente con la funzione di callback del Wiimote, ogni volta che il Wiimote rileva un segnale IR, viene chiamata la funzione di callback, che si occuperà di elaborare questo segnale, muovere il cursore ed eventualmente cliccare nel punto rilevato.

wiimote.py

Questo modulo contiene tutti i dati del Wiimote e le funzioni di callback. Contiene la classe `Wiimote` che viene istanziata al momento della connessione con il dispositivo ed è necessaria per la maggior parte delle azioni del programma.

linuxWiimoteLib.py

Questo modulo è l'unico collegamento tra il Wiimote ed il modulo wiimote.py, è nascosto al resto delle classi e si occupa di comunicare con il dispositivo utilizzando il protocollo Bluetooth.

thread.py

Questo modulo non è stato inserito nel diagramma in figura 5.2 perché è semplicemente un'estensione del modulo QThread, è importante precisare però che python-whiteboard è un'applicazione multi-thread; sono gestite tramite thread le seguenti procedure:

- la funzione di callback del Wiimote relativa alla ricezione dei dati IR,
- la connessione con il Wiimote,
- la gestione del cursore.

5.2.2 Analisi dinamica

Il flusso di esecuzione di python-whiteboard si articola in tre fasi:

1. connessione,
2. calibrazione,
3. lettura dei segnali IR

Connessione

In questa fase si connette il Wiimote, vengono creati gli oggetti relativi al dispositivo ed alla finestra di configurazione, inoltre viene creata la finestra di connessione e successivamente viene creato l'oggetto del modulo linuxWiimoteLib.py e la funzione di callback dei segnali IR.
Questa funzione non è la stessa che verrà utilizzata nella fase tre per la rilevazione dei segnali IR, serve soltanto al processo di calibrazione e funziona in modo diverso.

Calibrazione

In questa fase viene aperta la schermata di calibrazione e la funzione di callback si occuperà di rilevare i segnali IR fino al completamento della procedura, una volta registrati i quattro punti il Wiimote viene calibrato e vengono fatti i calcoli matriciali anticipati nel capitolo sull'infrastruttura.

Lettura dei segnali IR

Alla fine della calibrazione verrà creato un oggetto di tipo `FakeCursor` che si occuperà di creare la nuova funzione di callback e di creare un thread che si occupi di gestire il click del mouse.

5.3 Lightboard

5.3.1 wmdemo.c

Wmdemo.c è un programma contenuto in cwiid che serve ad illustrare le funzionalità del pacchetto.

Una volta fatto partire il programma ci si troverà davanti ad un menù a linea di comando, le azioni disponibili sono:

- accendere/spegnere i led,
- attivare/disattivare la vibrazione,
- attivare/disattivare il report dei dati dell'accelerometro,
- attivare/disattivare il report dei dati dei bottoni,
- attivare/disattivare il report delle estensioni,
- attivare/disattivare il report dei dati della telecamera IR,
- attivare/disattivare il report dello stato,
- attivare/disattivare i messaggi nella funzione di callback,
- richiedere un messaggio di stato e stamparlo.

Wmdemo.c offre inoltre un esempio di funzione di callback, che semplicemente stampa i report in una forma più leggibile, ed è un ottimo banco di prova per analizzare la libreria e le sue funzionalità.

5.3.2 Fasi preliminari

Alleggerire il codice

Uno degli scopi del progetto è di rendere lightboard utilizzabile, senza un consumo eccessivo di risorse, su sistemi come Raspberry Pi.
Per prima cosa quindi si è cercato di togliere da wmdemo.c tutte le funzionalità non utili al fine di lightboard, e di conseguenza anche dalla libreria, in modo da snellire il codice e quindi il consumo di RAM. Le funzionalità che ci servono sono:

- attivare/disattivare il report dei dati della telecamera IR,

- attivare i messaggi nella funzione di callback

Riguardo alle funzionalità escluse, le spiegazioni sono diverse:

- la gestione dei led, della vibrazione, dei bottoni e dell'accelerometro non è interessante ai fini del progetto;
- la gestione delle estensioni non aggiunge funzionalità utili ai fini del progetto;
- la gestione dello stato è stata eliminata perchè l'idea è quella di visualizzare le informazioni sul Wiimote attraverso l'interfaccia grafica.

Per una trattazione più completa riguardo i report del Wiimote si rimanda alla documentazione in Appendice A.

Integrazione con python-whiteboard

A questo punto ci si è occupati dell'integrazione dell'interfaccia grafica di python-whiteboard con la base del motore di elaborazione.
Python-whiteboard si occupa di configurare il Wiimote, mentre lightboard elabora i segnali IR e muove il cursore.
La problematica principale legata a questa fase è stata quella di come far comunicare le due applicazioni. Lasciando aperte entrambe le applicazioni e creando un canale di comunicazione tra i due si sarebbe ottenuto il massimo della flessibilità, sarebbe stato possibile modificare le configurazioni a tempo di esecuzione tramite l'interfaccia ed eseguirle in tempo reale su lightboard.
Questo però porta a conseguenze non conformi con le specifiche di progetto, in particolare tenere aperte entrambe le applicazioni porterebbe ad un consumo di RAM e CPU maggiore.
Anche modificando python-whiteboard in modo che resti in attesa di una modifica nelle impostazioni (e non in attesa di un input IR), python-whiteboard consuma considerevolmente le risorse della macchina.
Un approccio del genere quindi non porterebbe a nessun vantaggio effettivo rispetto all'utilizzare soltanto python-whiteboard. La seconda idea è

quella di scrivere tutte le configurazioni su di un file alla chiusura di python-whiteboard e farle leggere a lightboard.
In questo caso si perde in flessibilità, una volta chiuso python-whiteboard infatti non sarà possibile cambiare le impostazioni finché non si riavvia l'applicazione. In compenso però restringiamo l'esecuzione di python-whiteboard al solo avvio del programma, permettendo così ogni possibile ottimizzazione durante la fase più importante, cioè quella di rilevamento dei segnali IR e conseguente movimento del mouse.

Si è quindi adottato il secondo metodo, dall'interfaccia di python-whiteboard sarà sufficiente premere il bottone *Activate* per salvare sul file le configurazioni e chiudere l'interfaccia grafica. I dati salvati sul file sono:

1. indirizzo Bluetooth,
2. dati di calibrazione,
3. impostazioni di configurazione.

Lightboard a questo punto legge il file, si connette al Wiimote specificato dall'indirizzo e registra i dati relativi alla calibrazione in modo che possano essere utilizzati successivamente dalla funzione di callback. I dettagli sul formato del file verranno discussi nell'Appendice A.

Bisogna però fare una nota sulla funzione utilizzata per chiamare python-whiteboard dal codice di lightboard.
Inizialmente era stata usata la funzione della libreria standard C, `system`. Questa funzione richiede l'attivazione di un programma molto complesso, bash, solamente per elaborare alcuni parametri e questo mina la sicurezza dei propri programmi.
Si è preferito usare quindi una funzione della libreria *libs2argv* scritta dal Professor Renzo Davoli.
Le funzioni della piccola libreria s2argv consentono di elaborare l'unica stringa contenente tutti gli argomenti, compreso quello di indice 0, in modo flessibile, comodo, sicuro ed efficiente. La libreria infatti gestisce argomenti contenenti spazi in modo simile alle shell, fornisce il vettore argv nel formato

opportuno per le funzioni di libreria o, tramite la execs, provvede direttamente a richiamare la system call ed evitare che i programmatori usino scorrettamente le funzioni system o popen (o eseguano "sh -c"). [9]

5.3.3 Xtest ed elaborazione del segnale IR

A questo punto resta soltanto da interpretare i segnali IR in base ai dati di calibrazione e muovere il cursore di conseguenza.

Calibrazione

La schermata di calibrazione di python-whiteboard indica i punti ai quattro angoli dello schermo, situati a queste coordinate:

```
[40,40], [rx-40,40], [rx-40,ry-40], [40, ry-40]
```

Dove `rx` è la risoluzione orizzontale e `ry` è la risoluzione verticale dello schermo utilizzato. Una volta che si sono memorizzati i dati relativi ai quattro punti, si eseguono queste operazioni:

1. si crea la matrice `A`,
2. si crea la matrice `x`,
3. si calcola il risultato del sistema di equazioni dato dalle due matrici,
4. si memorizzano i dati.

Le coordinate del Wiimote rientrano nel range [0-1024] per la coordinata orizzontale e [0-768] per quella verticale. Di seguito un esempio di coordinate valide:

```
wii = [ [57,511], [712,368], [752,8], [31,21] ]
screen = [ [40,40], [1326,40], [1326,728], [40,728] ]
```

Per ogni coppia di coordinate si creano due righe nella matrice `A` 8x8 in questo modo:

```
riga 1 = ( wii[i][0], wii[i][1], 1, 0, 0, 0,
         -screen[i][0] * wii[i][0], -screen[i][0] * wii[i][1] )

riga 2 = ( 0, 0, 0, wii[i][0], wii[i][1], 1,
         -screen[i][1] * wii[i][0], -screen[i][1] * wii[i][1] )
```

Inoltre per ogni coppia di coordinate si creano due righe nella matrice `x` in questo modo:

```
riga 1 = ( screen[i][0] )
riga 2 = ( screen[i][1] )
```

Nel nostro esempio la matrice `A` con i calcoli esplicitati è:

$$\begin{bmatrix} 57 & 511 & 1 & 0 & 0 & 0 & -40*57 & -40*511 \\ 0 & 0 & 0 & 57 & 511 & 1 & -40*57 & -40*511 \\ 712 & 368 & 1 & 0 & 0 & 0 & -1326*712 & -1326*368 \\ 0 & 0 & 0 & 712 & 368 & 1 & -40*712 & -40*368 \\ 752 & 8 & 1 & 0 & 0 & 0 & -1326*752 & -1326*8 \\ 0 & 0 & 0 & 752 & 8 & 1 & -728*752 & -728*8 \\ 31 & 21 & 1 & 0 & 0 & 0 & -40*31 & -40*21 \\ 0 & 0 & 0 & 31 & 21 & 1 & -728*31 & -728*21 \end{bmatrix} \tag{5.1}$$

Mentre la matrice `x` è:

$$\begin{bmatrix} 40 \\ 40 \\ 1326 \\ 40 \\ 1326 \\ 728 \\ 40 \\ 728 \end{bmatrix} \tag{5.2}$$

Risolvendo il sistema di equazioni dato dalle due matrici (5.1 e 5.2) nella forma `Ax = h`, otteniamo una matrice 1x8 con i seguenti risultati:

$$\begin{bmatrix} 1,2256 \\ -0,0740 \\ 2,8908 \\ -0,3140 \\ -1,3729 \\ 754,4122 \\ -0,0004 \\ -0,0002 \end{bmatrix} \tag{5.3}$$

Questi risultati vengono memorizzati in otto variabili e salvati nel file di configurazione che sarà passato all'engine.[1] L'engine eseguirà questi calcoli per convertire le coordinate della telecamera del Wiimote in coordinate dello schermo:

```
screen_x = ( (matr[0]*x) + (matr[1]*y) + matr[2] ) /
           ( (matr[6]*x) + (matr[7]*y) + 1 )

screen_y = ( (matr[3]*x) + (matr[4]*y) + matr[5] ) /
           ( (matr[6]*x) + (matr[7]*y) + 1 )
```

Dove `screen_x` e `screen_y` sono le coordinate che verranno passate alle funzioni `XTestFakeMotionEvent` e `XTestFakeButtonEvent`, rispettivamente per muovere il mouse ed effettuare il click. `x` ed `y` sono invece le coordinate lette dal report IR del Wiimote e `matr[i]` contiene i valori della matrice `h` (5.3).

Movimento e click

Una volta che l'engine ha calcolato le nuove coordinate, la funzione `XTestFakeMotionEvent` di Xtest si occuperà di spostare il mouse. Per quanto riguarda il click invece, `XTestFakeButtonEvent` funziona così:

[1] I calcoli, risolti con il metodo di Gauss, si trovano a questo url: http://bit.ly/1EdUCPz

- se il parametro booleano `is_press` è vero, avremo una pressione del bottone;
- se `is_press` è falso, avremo un rilascio del bottone.

Detto questo, l'algoritmo utilizzato per il click consiste in:

- quando si rileva un segnale IR si fa un click (`is_press = true`),
- quando si riceve un report senza dati IR si rilascia il click (`is_press = false`).

Ciò permette di emulare il funzionamento del mouse senza utilizzare euristiche basate sulla durata del click, questo è permesso dal fatto che il Wiimote manda sempre report IR, anche quando non rileva dati.

5.4 Ottimizzazione

Come vedremo nel capitolo 6, realizzare il motore di elaborazione in C ha migliorato di molto le prestazioni, è stato difficile quindi migliorare ulteriormente il programma. Una delle ottimizzazioni che sono state fatte riguarda la modalità di solo puntamento. Come detto nel capitolo precedente, il Wiimote manda report di dati anche quando non rileva segnali IR, abbiamo visto che questo è molto utile per gestire correttamente il click del cursore, ma nel caso del solo puntamento questo è inutile. In questa modalità quindi i messaggi senza dati IR validi vengono scartati dal thread router senza essere smistati al thread mesg_callback, questo diminuisce il consumo di risorse nelle fasi IDLE del programma.

5.4.1 Ottimizzazioni in fase di compilazione

Un buon miglioramento delle prestazioni generali del programma si è ottenuto attraverso le modalità di ottimizzazione del compilatore gcc.
Normalmente gcc cerca di ridurre il tempo di compilazione e di far produrre al debug i risultati attesi. Le linee di codice C sono quindi indipendenti tra

loro: se si ferma l'esecuzione con un breakpoint, si può assegnare un nuovo valore ad una variabile o cambiare il *program counter* a qualsiasi altra linea della funzione, ottenendo esattamente il risultato che ci si aspetta dal codice sorgente.
Accendendo i flag dell'ottimizzazione invece, il compilatore cerca di incrementare le prestazioni del programma e/o la dimensione del codice, a costo di un maggior tempo di compilazione e della possibilità di fare debug del codice.
Il compilatore esegue ottimizzazioni basate sulle conoscenze che esso ha del programma. Accendere i flag dell'ottimizzazione è possibile aggiungendo `-Ox` al comando di compilazione, dove `x` è il livello di ottimizzazione voluto, oppure specificando i flag delle singole ottimizzazioni.

-O1

Con il livello 1, il compilatore cerca di ridurre la dimensione del codice ed il tempo di esecuzione, senza nessuna ottimizzazione che comporta un aumento rilevante nel tempo di compilazione.

-O2

Con il livello 2, gcc esegue quasi tutte le ottimizzazioni supportate che non comportano un grosso compromesso spazio-velocità. Comparato al livello 1, questa opzione aumenta sia il tempo di compilazione, sia le prestazioni del codice generato.

-O3

Il livello 3 è il massimo livello di ottimizzazione possibile con gcc (usando i flag standard).

Lightboard, nella versione 0.1, utilizza il livello 2, per evitare instabilità nel programma. Per una trattazione più completa rifersi alla documentazione ufficiale. [4]

Capitolo 6

Risultati

In questo capitolo saranno esposti i dati relativi alle analisi delle prestazioni di lightboard e, successivamente, messi in relazione con quelli visti nella sezione 4.1.
Sulla prima macchina, descritta nella sezione 4.1.1 sono stati fatti tre benchmark, relativi a tre momenti diversi dello sviluppo di lightboard, su Raspberry Pi invece i primi due test sono stati eseguiti insieme.
I test analizzano, tramite htop, il consumo di RAM e CPU dei vari thread e del server X; saranno riportati anche i dettagli relativi ai benchmark effettuati su python-whiteboard.[1]

Prima di vedere i benchmark relativi al consumo di risorse della macchina analizziamo, come già fatto per python-whiteboard, il tempo di elaborazione richiesto da lightboard per elaborare un singolo segnale IR.
Esaminando un campione di 1000 segnali, la media dei tempi di elaborazione è di 0,0005291 secondi, il 45,34% del tempo impiegato da python-whiteboard che abbiamo visto nella sezione 4.1.
Questo dato è interessante perché ci dice che lightboard impiega meno della metà del tempo di python-whiteboard per elaborare un segnale, rimanendo quindi in IDLE per più tempo.

[1] Ogni dato corrisponde ad una media di 10 rilevazioni.

6.1 Benchmark su Asus X53S

Prima di vedere i risultati dei benchmark daremo una breve spiegazione della terminologia usata.

VIRT

`VIRT` specifica la memoria virtuale utilizzata dal processo, che equivale alla somma delle seguenti:

- memoria attualmente usata,
- memoria mappata nel processo (ad esempio la RAM della scheda video per il server X),
- i file sul disco che sono mappati nel processo (per la maggior parte sono librerie condivise),
- memoria condivisa con altri processi.

`VIRT` rappresenta la memoria a cui il programma può accedere nel momento attuale.

RES

`RES` indica la memoria residente, una rappresentazione accurata di quanta memoria fisica è attualmente consumata dal processo.

SHR

`SHR` indica quale parte di `VIRT` è utilizzata attualmente come memoria condivisa o per le librerie. Nel secondo caso non significa necessariamente che l'intera libreria è residente. Ad esempio, se un programma usa solo alcune funzioni della libreria, essa è mappata interamente e sarà calcolata in `VIRT` e `SHR`, ma solo la parte della libreria contenente le funzioni utilizzate sarà caricata e calcolata in `RES`.

CPU

La percentuale di tempo del processore usato dal processo.

Command

Il nome del comando che ha fatto partire il processo. [20]

6.1.1 Benchmark 0

Questo benchmark è stato eseguito prima delle ottimizzazioni e raccoglie soltanto le prestazioni del programma durante la rilevazione dei segnali IR.

VIRT (MB)	RES (KB)	SHR (KB)	CPU (%)	Command
502	55212	45684	7,6	/usr/bin/X
176	2328	2088	3,8	./lightboard
176	2328	2088	2,4	./lightboard
176	2328	2088	0,9	./lightboard

Tabella 6.1: Asus X53S Benchmark 0 - lightboard IR

VIRT (MB)	RES (KB)	SHR (KB)	CPU (%)	Command
796	94600	62644	45,3	py-whiteboard
796	94600	62644	30,1	py-whiteboard
796	94600	62644	15,3	py-whiteboard
505	57936	48400	5,2	/usr/bin/X

Tabella 6.2: Asus X53S Benchmark 0 - python-whiteboard IR

6.1.2 Benchmark 1

Questo benchmark serve a valutare le prestazioni del programma quando non rileva nessun segnale IR.

VIRT (MB)	RES (KB)	SHR (KB)	CPU (%)	Command
176	2496	2256	2,4	./lightboard
176	2496	2256	1,4	./lightboard
176	2496	2256	1,4	./lightboard
506	55716	45852	0,0	/usr/bin/X

Tabella 6.3: Asus X53S Benchmark 1 - lightboard IDLE

VIRT (MB)	RES (KB)	SHR (KB)	CPU (%)	Command
796	94752	62728	21,0	py-whiteboard
796	94752	62728	16,2	py-whiteboard
796	94752	62728	4,8	py-whiteboard
506	54324	44348	0,0	/usr/bin/X

Tabella 6.4: Asus X53S Benchmark 1 - python-whiteboard IDLE

6.1.3 Benchmark 2

Questo benchmark è l'ultimo che è stato fatto per la versione 0.1 di lightboard, raccoglie le informazioni sulle prestazioni del software dopo le ottimizzazioni di cui abbiamo parlato nella sezione 5.4.
Qui sono state rifatte le analisi di python-whiteboard (per tenere conto delle variazioni dovute alla macchina), ma non verranno mostrate, tuttavia esse sono state aggiunte ai dati delle precedenti analisi per il calcolo della media.

VIRT (MB)	RES (KB)	SHR (KB)	CPU (%)	Command
498	50696	41164	7,6	/usr/bin/X
176	2324	2084	3,3	./lightboard
176	2324	2084	1,9	./lightboard
176	2324	2084	0,9	./lightboard

Tabella 6.5: Asus X53S Benchmark 2 - lightboard IR

VIRT (MB)	RES (KB)	SHR (KB)	CPU (%)	Command
112	2324	2084	0,9	./lightboard
112	2324	2084	0,9	./lightboard
492	42952	33348	0,0	/usr/bin/X
112	2324	2084	0,0	./lightboard

Tabella 6.6: Asus X53S Benchmark 2 - lightboard IDLE

6.2 Benchmark su Raspberry Pi

Per questo benchmark valgono le stesse premesse del precedente, l'unica differenza è che il Raspberry Pi non è multi-thread.

6.2.1 Benchmark 0

VIRT (KB)	RES (KB)	SHR (KB)	CPU (%)	Command
25104	14000	5424	16,9	/usr/bin/X
28168	1084	860	4,8	./lightboard

Tabella 6.7: Raspberry Pi Benchmark 0 - lightboard IR

VIRT (KB)	RES (KB)	SHR (KB)	CPU (%)	Command
159000	38000	22000	77,1	py-whiteboard
24044	13000	5556	5,1	/usr/bin/X

Tabella 6.8: Raspberry Pi Benchmark 0 - python-whiteboard IR

6.2.2 Benchmark 1

VIRT (KB)	RES (KB)	SHR (KB)	CPU (%)	Command
25104	14000	5424	1,3	/usr/bin/X
28168	1084	860	0,3	./lightboard

Tabella 6.9: Raspberry Pi Benchmark 1 - lightboard IDLE

VIRT (KB)	RES (KB)	SHR (KB)	CPU (%)	Command
159000	38000	22000	58,3	py-whiteboard
24044	13000	5556	2,6	/usr/bin/X

Tabella 6.10: Raspberry Pi Benchmark 1 - python-whiteboard IDLE

6.2.3 Benchmark 2

VIRT (KB)	RES (KB)	SHR (KB)	CPU (%)	Command
25204	14000	5176	16,2	/usr/bin/X
28164	1076	852	4,1	./lightboard

Tabella 6.11: Raspberry Pi Benchmark 2 - lightboard IR

VIRT (KB)	RES (KB)	SHR (KB)	CPU (%)	Command
25204	14000	5176	1,8	/usr/bin/X
28164	1076	852	0,5	./lightboard

Tabella 6.12: Raspberry Pi Benchmark 2 - lightboard IDLE

6.3 Confronto con python-whiteboard

In questa sezione confronteremo i dati e calcoleremo le percentuali di utilizzo delle risorse, rispetto a python-whiteboard.
Questo vuol dire che il consumo di RAM di python-whiteboard verrà calcolato come 100%, se lightboard consuma la metà, avremo un 50% rispetto a python-whiteboard. Per quanto riguarda la memoria verrà analizzata soltanto la memoria residente, `RES`. Spiegazione delle etichette utilizzate:

- **RES (KB)** - specifica il totale di memoria residente utilizzata dai thread,
- **CPU (KB)** - specifica il totale di CPU utilizzata dai thread,
- **RES (%)** - specifica la percentuale di memoria residente utilizzata dai thread rispetto a quella utilizzata da python-whiteboard,
- **CPU (%)** - specifica la percentuale di CPU utilizzata dai thread rispetto a quella utilizzata da python-whiteboard.

6.3.1 Risultati su Asus X53S

Program	RES (KB)	CPU (KB)	RES (%)	CPU (%)
lightboard	57540	14,7	37,72	15,33
py-whiteboard	152536	95,5	100,00	100,00

Tabella 6.13: Asus X53S Risultati Benchmark 0 - IR

Program	RES (KB)	CPU (KB)	RES (%)	CPU (%)
lightboard	58212	5,2	39,05	12,38
py-whiteboard	149076	42,0	100,00	100,00

Tabella 6.14: Asus X53S Risultati Benchmark 1 - IDLE

Program	RES (KB)	CPU (KB)	RES (%)	CPU (%)
lightboard IR	53020	13,7	34,76	14,29
lightboard IDLE	45276	1,8	30,37	4,29

Tabella 6.15: Asus X53S Risultati Benchmark 2

6.3.2 Risultati su Raspberry Pi

Program	RES (KB)	CPU (KB)	RES (%)	CPU (%)
lightboard	15084	21,7	29,58	26,40
py-whiteboard	51000	82,2	100,00	100,00

Tabella 6.16: Raspberry Pi Risultati Benchmark 0

Program	RES (KB)	CPU (KB)	RES (%)	CPU (%)
lightboard	15084	1,6	29,58	2,63
py-whiteboard	51000	60,9	100,00	100,00

Tabella 6.17: Raspberry Pi Risultati Benchmark 1

Program	RES (KB)	CPU (KB)	RES (%)	CPU (%)
lightboard IR	15076	20,3	29,56	24,70
lightboard IDLE	15076	2,3	29,56	3,78

Tabella 6.18: Raspberry Pi Risultati Benchmark 2

6.3.3 Analisi dei dati

Dai dati vediamo che lightboard consuma considerevolmente poco rispetto a python-whiteboard, inoltre è stato raggiunto l'obiettivo di rendere lightboard fruibile su Raspberry Pi.

In generale il miglioramento più evidente è stato nella fase di riposo del programma, il che porta ad avere un aumento di prestazioni particolarmente interessante nel caso della modalità di click. L'utilizzo è scorrevole e senza evidenti ritardi, non si notano grandi differenze rispetto all'utilizzo su una macchina più potente.

Conclusioni

Abbiamo visto come sia possibile migliorare le prestazioni dei programmi attualmente utilizzati per creare lavagne multimediali interattive, sostituendo il motore di elaborazione con uno scritto in C.
Lightboard permette di sperimentare l'utilizzo di una LIM basata su Raspberry Pi e Wiimote, senza eccessivi cali prestazionali dovuti alle poche risorse del sistema. Il lavoro fatto con lightboard, ed i lavori successivi, puntano a raggiungere prestazioni simili a quelle ottenute dalle LIM commerciali, utilizzando quantità sempre minori di risorse, così da permettere la creazione di sistemi integrati e facilmente installabili e facilitare l'utilizzo di tecnologie *open source* nelle scuole italiane.

Sviluppi futuri

Per ottenere queste prestazioni si sono dovuti fare alcuni compromessi che hanno escluso funzionalità, a volte, decisamente importanti.
Di seguito un elenco delle idee che potrebbero portare ad un'evoluzione di lightboard:

- rendere lightboard compatibile con il Wiimote Plus,
- implementare una libreria personalizzata per comunicare con il Wiimote, distaccandosi così da licwiid,
- implementare un'interfaccia grafica personalizzata, distaccandosi così da python-whiteboard,

- rendere lightboard un demone configurabile,
- pacchettizzare lightboard e fornire una procedura di installazione.

Lightboard

Manuale e Documentazione Tecnica

Versione 1.0

Giovanni Incammicia

Introduzione

Scopo di questa trattazione è quello di fornire un documento di veloce consultazione per comprendere il progetto ed il funzionamento del software. Nel primo capitolo di questo documento vedremo come installare ed usare lightboard per creare una lavagna interattiva multimediale. Il primo capitolo è destinato all'utente di lightboard.
Dal secondo capitolo la documentazione diventa più tecnica e destinata agli sviluppatori che vogliono ampliare lightboard con nuove funzionalità o renderlo più accessibile.
Il secondo capitolo si concentrerà sul funzionamento di basso livello del Wiimote, di come ci si interfaccia ad esso e delle sue caratteristiche interne.
Il terzo capitolo è dedicato a *libcwiid*, la libreria C utilizzata in lightboard per comunicare con il Wiimote, la sua struttura ed il suo funzionamento.
Il quarto capitolo parla di *python-whiteboard*, l'interfaccia grafica del progetto. Come è strutturata e come funziona saranno le domande a cui cercheremo di rispondere in questa breve trattazione.
Il quinto capitolo spiega quali sono le modifiche principali che sono state fatte in lightboard ed i dettagli tecnici dell'implementazione.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Guida per l'utente

In questa sezione spiegheremo come eseguire ed utilizzare lightboard. Lightboard, nella versione 0.1, non prevede una routine di installazione.

1.1 Librerie necessarie

Lightboard utilizza una versione personalizzata di libcwiid, non sarà quindi necessario installare il pacchetto libcwiid-dev.
Le librerie da installare sono:

- python-numpy,
- python-bluez,
- python-xlib,
- libbluetooth-dev,
- libxtst-dev.

1.2 Guida all'esecuzione

Questi sono i passaggi necessari all'esecuzione di lightboard:

1. posizionarsi nella cartella lightboard e lanciare il comando `make`;

2. lanciare il comando `./lightboard`;

3. a questo punto si aprirà un terminale e l'interfaccia di python-whiteboard, essa risulterà privata di alcune configurazioni, alcune delle funzionalità di python-whiteboard sono state alterate;

4. una volta connesso il Wiimote e calibrato lo schermo, cliccare su *Activate* per chiudere l'interfaccia utente;

5. da questo momento si potrà utilizzare la penna ad infrarossi proprio come si faceva con python-whiteboard.

1.3 Configurazioni

In figura 1.1 si vede l'interfaccia di lightboard (python-whiteboard), di seguito saranno spiegate le configurazioni disponibili nella versione 0.1 del programma. Si riportano di seguito le frasi in inglese previste dall'interfaccia in figura 1.1, con opportuna spiegazione.

- *Don't wait for devices, pick the first one* - questa impostazione permette di selezionare il primo Wiimote disponibile e connettersi automaticamente ad esso, in caso contrario verranno mostrati tutti i Wiimote rilevati durante la fase di ricerca e si potrà scegliere quale connettere;

- *Do calibration after connection* - questa impostazione permette di eseguire automaticamente la calibrazione subito dopo la connessione, in caso contrario sarà possibile calibrare premendo il bottone *Calibrate* nella schermata principale;

- *Ir sensitivity* - questa impostazione è disattivata nella versione 0.1 del software;

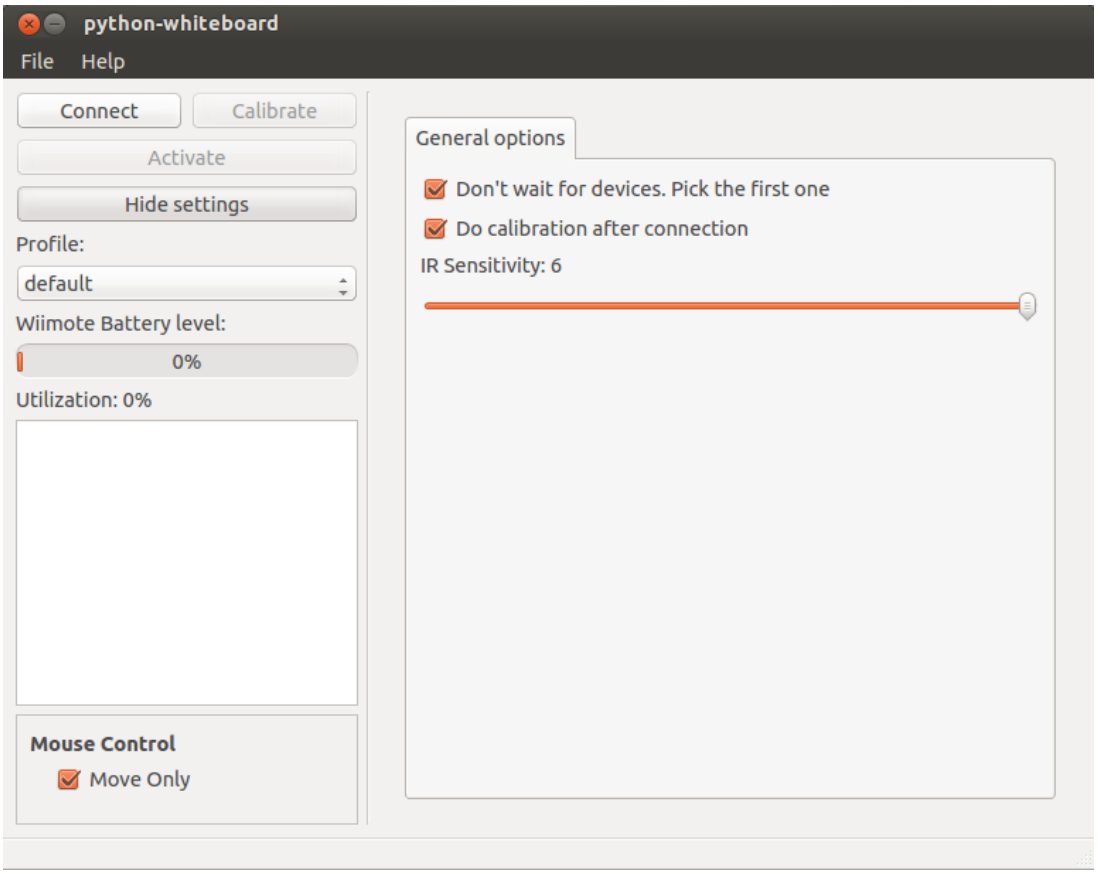

Figura 1.1: Screenshot interfaccia di lightboard

- *Move only* - questa casella di controllo permette di specificare se utilizzare la penna ad infrarossi come un puntatore, spostando quindi il cursore nel punto rilevato dalla telecamera IR, o se usarla come sostituto del mouse, effettuando un click sinistro nella posizione rilevata;
- *Profile* - i profili permettono di salvare impostazioni personalizzate per un uso futuro.

Capitolo 2

Come funziona il Wiimote

In questa sezione spiegheremo il funzionamento del Wiimote e come interfacciarsi con esso. Il Wiimote utilizza hardware chiuso, per questo motivo le informazioni sul suo funzionamento interno sono il risultato di analisi di *reverse-engineering* [25]. Vedremo inoltre come comunicare con il Wiimote a livello di Bluetooth, il significato dei messaggi ed il procedimento di configurazione della telecamera IR. Si consiglia di leggere questa sezione con una conoscenza di base del protocollo Bluetooth e del profilo BT-HID, utilizzato dal Wiimote. Si può trovare una breve trattazione dell'argomento nella tesi associata [13], per approfondimenti ulteriori leggere le specifiche del profilo Bluetooth HID [11].

2.1 Comunicazione Bluetooth

Il protocollo Bluetooth HID (Human Interface Device) definisce un insieme di servizi che possono essere usati tra un host ed un dispositivo.
Questo profilo ha bisogno di due o più canali *L2CAP* per trasmettere pacchetti di controllo e pacchetti di dati.
Un host BT-HID deve aprire due canali: *Control* e *Interrupt*.
Il canale di controllo è impostato come un servizio di tipo *Best Effort*.
Dati a bassa latenza sono invece trasportati sul canale di interrupt, il tipo di

servizio deve dunque essere impostato a *Guaranteed* per assicurare la qualità del servizio.

BT-HID è costruito su alcuni concetti fondamentali, un *report descrittore* ed i *report*.
I report sono gli agglomerati di dati che sono scambiati tra un dispositivo ed un client software.
I report descrittore descrivono il formato ed il significato di ogni agglomerato di dati che il dispositivo supporta.

2.1.1 Report

Quando le applicazioni ed i dispositivi HID si scambiano dati, lo fanno tramite report. Ci sono tre tipi di report:

- **Report di input** - agglomerati di dati che vengono mandati dal dispositivo HID all'applicazione, tipicamente quando lo stato di un controllo cambia;
- **Report di output** - agglomerati di dati che vengono mandati dall'applicazione al dispositivo, tipicamente per dare un comando al dispositivo, come accendere i led;
- **Report di funzionalità** - agglomerati di dati che possono essere manualmente letti e/o scritti, sono tipicamente correlati ad informazioni di configurazione.

Tra report e canali ci sono le seguenti relazioni:

- i report di funzionalità bi-direzionali sono trasportati sul canale di controllo,
- i report di input e output sono trasportati sul canale di interrupt,
- tutti i dispositivi BT-HID dovrebbero comunicare attraverso un canale di controllo ed un canale di interrupt.

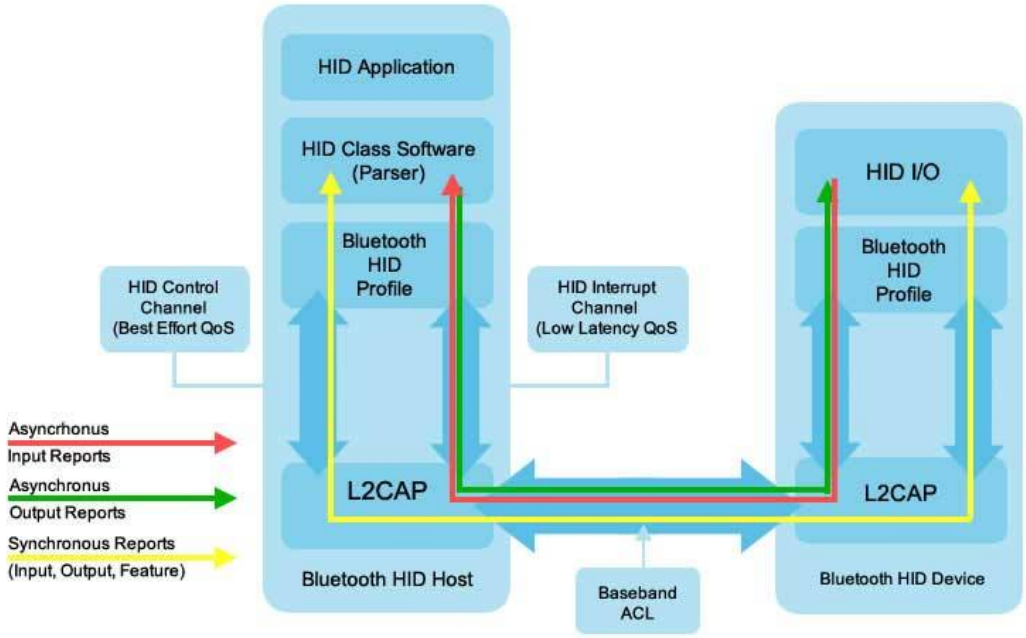

Figura 2.1: Tipi di report e mappatura dei canali L2CAP

Intestazione di transazione BT-HID

Tutti i messaggi scambiati tra un dispositivo HID ed un'applicazione sono preceduti da un'intestazione di transazione che divide in due campi: il tipo di transazione ed un parametro (che dipende dal tipo).

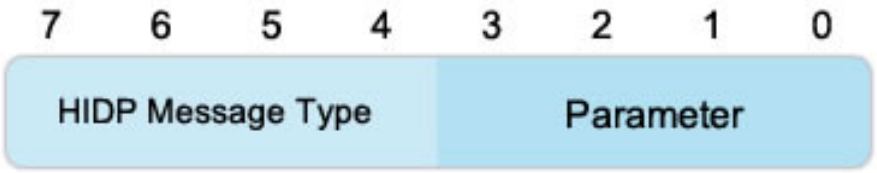

Figura 2.2: Header di un messaggio

Le transazioni consistono di un *payload* di richiesta al dispositivo, ad esso seguiranno, in risposta, dei dati o un messaggio di *handshake*.
Le richieste sono quelle indicate da `GET` o `SET`; se la richiesta è corretta seguirà un pacchetto di tipo `DATA` (seguito eventualmente da pacchetti di tipo `DATC`) o un `HANDSHAKE`.

Hex	Tipo di transazione	Lunghezza payload (byte)
0	HANDSHAKE	1
1	HID_CONTROL	1
2-3	Reserved	
4	GET_REPORT	1-4
5	SET_REPORT	1+payload report dati
6	GET_PROTOCOL	1
7	SET_PROTOCOL	1
8	GET_IDLE	1
9	SET_IDLE	2
A	DATA	1+payload report dati
B	DATC	1+continuo payload report dati
C-F	Reserved	

Tabella 2.1: Tipi di transazione supportati

Sul canale di interrupt vengono trasmessi i pacchetti di tipo DATA e DATC, gli altri utilizzano il canale di controllo.
Non vedremo i dettagli relativi ad ogni tipo di transazione, per approfondimenti consultare la sezione 7.4 delle specifiche BT-HID [11].

2.1.2 Report del Wiimote

Per convenzione mostreremo i pacchetti includendo tra parentesi il comando HID, seguito dall'ID del report ed il payload, come descritto nelle sezioni 7.3 e 7.4 delle specifiche BT-HID [11]. Ogni byte è scritto in esadecimale omettendo lo 0x e separato da spazi.
Prendiamo per esempio il messaggio:

```
(a1) 30 00 00
```

Dalla tabella 2.1 vediamo che il tipo a indica un messaggio di tipo DATA, il parametro 1 indica che si tratta di un messaggio di input (vedere tabella 15

I/O	ID	Byte	Funzione
O	0x10	1	Sconosciuta
O	0x11	1	LED
O	0x12	2	Modalità report dati
O	0x13	1	Abilitare la telecamera IR
O	0x14	1	Abilitare il microfono
O	0x15	1	Richiesta di informazioni di stato
O	0x16	21	Scrivere in memoria o nei registri
O	0x17	6	Leggere la memoria o i registri
O	0x18	21	Dati microfono
O	0x19	1	Disattivare il microfono
O	0x1A	1	Abilitare la telecamera IR 2
I	0x20	6	Informazioni di stato
I	0x21	21	Leggere i dati dalla memoria o dai registri
I	0x22	4	Notifica report di output, risultato funzione
I	0x30-0x3f	2-21	Report dati

Tabella 2.2: Report del Wiimote

della sezione 7.4.9 delle specifiche BT-HID [11]). `0x30` è l'ID del report, dalla tabella 2.2 vediamo che esso determina un report dati, mentre `0x00 0x00` è il payload.

Informazioni comuni dei report output

In ogni report output singolo, il bit 0 (`0x01`) del primo byte controlla la funzionalità di vibrazione, il bit 2 (`0x04`) è usato in molti report output come flag ON/OFF per la specifica funzionalità da esso controllata. Per esempio, mandando `0x04` al report `0x19`, sarà disattivato il microfono:

```
(a2) 19 04
```

Mandando invece `0x00` lo si attiverà:

```
(a2) 19 00
```

Questo comportamento è condiviso dai seguenti report:

- Modalità report dati (`0x12`),
- Abilitare la telecamera IR (`0x13`),
- Abilitare il microfono (`0x14`),
- Disattivare il microfono (`0x19`),
- Abilitare la telecamera IR 2 (`0x1A`).

Per quanto riguarda la vibrazione invece non abbiamo un tipo di report specifico, in generale utilizzando il bit 0 possiamo attivarla o disattivarla da qualsiasi report output, questo ad esempio la attiva:

```
(a2) 11 01
```

Informazioni comuni dei report input

I primi due byte di tutti i report input, eccetto 0x3d, contengono i bottoni principali (`BB BB`). Questo include tutti i report di stato `0x2*`, non solo i report di dati `0x3*`.
`0x3d` è un'eccezione, dato che restituisce soltanto informazioni di estensione.

2.2 Report

In questa sezione verranno illustrati i report utilizzati nel progetto per interfacciarsi con il Wiimote e ricevere i dati della telecamera IR.

2.2.1 Status reporting

0x20 Status

Per richiedere il report di stato, bisogna mandare un qualsiasi messaggio che inizi con `0x15`.

```
(a2) 15 00
```

Questo richiede un report di stato (e disattiva la vibrazione).
In risposta avremo:

```
(a1) 20 BB BB LF 00 00 VV
```

`BB` indica i dati relativi ai bottoni, `VV` è il livello di batteria, `LF` è una maschera di bit, indicata nella tabella 2.3

Bit	Maschera	Significato
0	0x01	La batteria è quasi scarica
1	0x02	Un'estensione è collegata
2	0x04	Microfono abilitato
3	0x08	Telecamera IR abilitata
4	0x10	LED 1
5	0x20	LED 2
6	0x40	LED 3
7	0x80	LED 4

Tabella 2.3: Maschera di bit report input

0x21 Dati di lettura della memoria

Questo report viene mandato quando viene fatta una lettura di memoria. Restituisce da 1 a 16 byte di dati per volta.

```
(a1) 21 BB BB SE AA AA DD DD DD DD DD
     DD DD DD DD DD DD DD DD DD DD DD
```

`S` è la dimensione in byte, meno uno, del pacchetto dati corrente.
`E` è il flag di errore, quelli conosciuti sono: 0 per nessun errore, 7 quando si cerca di fare una lettura da un registro in sola lettura e 8 quando si cerca di leggere da un indirizzo di memoria non esistente.

`AA AA` indicano i due byte meno significativi dell'indirizzo assoluto di cui si è richiesta la lettura (il resto dell'indirizzo non è restituito, possiamo però trovarlo nella richiesta).
I byte `DD` indicano i dati, riempiti con degli zeri fino ad avere 16 byte, se più di 16 byte sono richiesti, si riceveranno più pacchetti, con l'indirizzo `AA AA` incrementato di 16 ogni volta.

0x22 Report di notifica, risultato di funzione

Questo report di input è inviato all'host in seguito ad un errore relativo ad un report output, oppure per notificare il risultato di una funzione. È inviato quando il bit 1 di ogni byte di un report di output vale 1.

```
(a1) 22 BB BB RR EE
```

`RR` è il numero di report output che il Wiimote notifica di aver ricevuto.
`EE` è il codice di errore o il risultato della funzione: `00`=successo, `03`=errore, `04,05,08`=sconosciuto.

2.2.2 Data reporting

Il Wiimote ha varie modalità di *data reporting*. Ognuna di questa modalità combina certe funzionalità del nucleo, con dati di periferiche esterne e li manda all'host tramite uno dei report ID, in base alla modalità.
La modalità di data reporting è impostata mandando un comando di due byte al report `0x12`:

```
(a2) 12 TT MM
```

Il bit 2 di `TT` specifica se si desidera un report continuo.
Se il bit 2 (`0x04`) vale 1, il Wiimote manderà report a prescindere se ci siano state, o meno, modifiche nei dati. Al contrario, normalmente il Wiimote manda i dati quando subiscono una modifica.
`MM` specifica la modalità di reporting. Ogni modalità è specificata dall'ID di

report output che viene inviato, per esempio questo imposta la modalità a `0x33`:

```
(a2) 12 00 33
```

Da questo momento i dati arriveranno con l'ID `0x33`; l'ID di default per i report è `0x30`.

Qui sono riportati soltanto i report di interesse per il progetto, ovvero il report che contiene i dati della telecamera IR; per una trattazione completa consultare il sito wiibrew [25]

0x33 Bottoni, accelerometro e telecamera IR con 12 byte di dati

Questa modalità restituisce dati dai bottoni, dall'accelerometro e dalla telecamera IR:

```
(a1) 33 BB BB AA AA AA II II II II II II II II II II II II
```

Dove `BB BB` indicano i dati dei bottoni, `AA AA AA` quelli dell'accelerometro e i 12 byte `II` sono quelli della telecamera IR.

2.2.3 Memoria e registri

Il Wiimote include una memoria EEPROM integrata, parte di essa è accessibile agli utenti.
Questa area utente è usata per memorizzare le costanti di calibrazione ed altri dati di cui non ci occuperemo in questo documento.
Sia la memoria integrata, sia i registri sono accessibili usando lo stesso report.
Per leggere dati, i comandi sono mandati al report output `0x17`:

```
(a2) 17 MM FF FF FF SS SS
```

`FF FF FF` è l'offset, `SS SS` è la dimensione da leggere in byte (entrambi in formato big-endian).
Il bit 2 (`0x04`) di `MM` seleziona lo spazio degli indirizzi. Azzerare questo bit permette di leggere la memoria EEPROM, impostarlo ad 1 significa leggere

dai registri di controllo.
I dati letti sono restituiti attraverso il report input `0x21` visto nella sezione 2.2.1. Per scrivere dati, i comandi vanno mandati al report output `0x16`:

```
(a2) 16 MM FF FF FF SS DD DD DD DD DD
     DD DD DD DD DD DD DD DD DD DD DD
```

Il significato dei byte è lo stesso del report `0x21`, eccetto per la dimensione che può essere massimo 16 byte. Sul report `0x22` arriva una notifica (acknowledgement) che non è ancora stata analizzata.
La scrittura sulla memoria EEPROM non verrà descritta in questo documento, in quanto non interessante ai fini del progetto.

Registri di controllo

Il Wiimote ha alcuni registri di memoria che corrispondono a diverse periferiche. Queste includono il microfono, il controller delle estensioni e la telecamera IR.
La periferica a cui accedere è selezionata dal primo byte dell'indirizzo, i 16 bit più bassi specificano il registro da accedere per quella periferica. Le periferiche conosciute sono indicate nella tabella 2.4

Inizio	**Fine**	**Uso**
`0xA20000`	`0xA20009`	Impostazioni microfono
`0xA40000`	`0xA400FF`	Impostazioni e dati del controller delle estensioni
`0xA60000`	`0xA600FF`	Impostazioni e dati del wii motion plus
`0xB00000`	`0xB00033`	Impostazioni telecamera IR

Tabella 2.4: Registri delle periferiche conosciute

2.3 Telecamera IR

Il Wiimote include una telecamera monocromatica da 128x96 di risoluzione con un processore di immagini integrato. La telecamera guarda attraverso

un filtro che fa passare solo gli infrarossi.
Il processore di immagini integrato può seguire fino a 4 oggetti in movimento e questi dati sono gli unici disponibili per l'host.
Il processore interno usa un'analisi 8x che fornisce una risoluzione di 1024x768 ai punti tracciati. La telecamera IR viene abilitata impostando il bit 2 sul report output `0x13` e `0x1A`:

```
(a2) 13 04
(a2) 1A 04
```

Il primo abilita un *pixel clock* a 24MHz sul pin 7 della telecamera. Il secondo azzera il pin 4, probabilmente abilita la telecamera in una modalità a basso consumo.

La telecamera IR ha un campo visivo effettivo di circa 33 gradi orizzontali e 22 gradi verticali. Riesce a rilevare sorgenti da 940nm con approssimativamente il doppio dell'intensità rispetto a sorgenti da 850nm.

2.3.1 Inizializzazione

La seguente procedura attiva la telecamera IR:

1. abilitare la telecamera IR (Manda `0x04` al report output `0x13`),
2. abilitare la telecamera IR 2 (Manda `0x04` al report output `0x1A`),
3. scrivere `0x08` nel registro `0xB00030`,
4. scrivere il blocco di sensibilità 1 nel registro `0xB00000`,
5. scrivere il blocco di sensibilità 2 nel registro `0xB0001A`,
6. scrivere il numero della modalità nel registro `0xB00033`,
7. scrivere `0x08` nel registro `0xB00030` (di nuovo).

Dopo questi passaggi, il Wiimote si troverà in uno di questi tre stati:

- la telecamera non sta memorizzando dati,

- la telecamera memorizza dati a metà della sensibilità,
- la telecamera memorizza dati a piena sensibilità.

La scelta dello stato sembra essere puramente casuale. Ripetere la procedura finché non ci si trova nello stato desiderato. Per evitare la scelta casuale dello stato, inserire un ritardo di 50ms tra ogni singola trasmissione.

Impostazioni di sensibilità

La sensibilità è controllata da due blocchi di configurazione, lunghi nove e due byte rispettivamente. Le impostazioni nella tabella 2.5 sono note e funzionanti.

Blocco 1	**Blocco 2**	**Note**
00 00 00 00 00 00 90 00 C0	40 00	
00 00 00 00 00 00 FF 00 0C	00 00	Massima sensibilità
00 00 00 00 00 00 90 00 41	40 00	Alta sensibilità
02 00 00 71 01 00 64 00 FE	FD 05	Livello wii 1
02 00 00 71 01 00 96 00 B4	B3 04	Livello wii 2
02 00 00 71 01 00 AA 00 64	63 03	Livello wii 3
02 00 00 71 01 00 C8 00 36	35 03	Livello wii 4
07 00 00 71 01 00 72 00 20	1F 03	Livello wii 5

Tabella 2.5: Blocchi di sensibilità

L'ultimo byte del blocco 1 è inversamente proporzionale alla sensibilità. Entrambi i byte del blocco 2 devono essere 0 per il massimo range di sensibilità.
Impostare la sensibilità al valore più alto possibile, senza che vengano rilevate fonti luminose non volute.

2.3.2 Formato dei dati

La telecamera IR può restituire differenti insiemi di dati che descrivono gli oggetti che sta tracciando.
La telecamera IR assegna l'oggetto identificato al primo slot possibile, se un oggetto sparisce dalla visuale, lo slot viene marcato come vuoto. Con più di quattro oggetti, la telecamera cambia velocemente da uno all'altro; questo permette di percepire più di quattro oggetti, ad una ridotta velocità di risposta e affidabilità.

Modalità	**Numero di modalità**
Basic	1
Extended	3
Full	5

Tabella 2.6: Modalità di formato dei dati

Vedremo qui in dettagli soltanto la modalità *extended* perché è quella utilizzata nel progetto.
La modalità *basic* non dà informazioni sulla dimensione dell'oggetto, mentre quella *full* prevede l'invio di due pacchetti per ogni rilevamento, il che rallenterebbe l'applicazione.

Modalità extended

	Bit							
Byte	7	6	5	4	3	2	1	0
0	**X**$< 7:0 >$							
1	**Y**$< 7:0 >$							
2	**Y**$< 9:8 >$		**X**$< 9:8 >$		**S**$< 3:0 >$			

Tabella 2.7: Formato dei dati

Nella modalità basic, la telecamera IR restituisce 10 byte di dati, corrispondenti alle cordinate X e Y di ognuno dei quattro punti. Ogni coordinata occupa 10 bit ed ha un range di 0-1023 per l'X e 0-767 per l'Y. Nella modalità extended, la telecamera IR restituisce gli stessi dati della modalità basic, più la dimensione di ogni oggetto. I dati occupano 12 byte, tre per ogni oggetto. La dimensione ha un range di 0-15. Il formato dei dati è illustrato in tabella 2.7.

Capitolo 3

La libreria libcwiid

Libcwiid è una libreria *multi-thread* che fornisce un'API per interfacciarsi con il Wiimote. Analizzeremo quindi la sua struttura, le funzioni di ogni thread, le *pipe* e le interazioni con il Wiimote. Dopodiché analizzeremo la sequenza di azioni che ogni thread compie.

3.1 Struttura di libcwiid

Libcwiid è composto da quattro thread (in verde nel diagramma in figura 3.1) che comunicano tra di loro tramite pipe (in arancione), i thread comunicano con il Wiimote tramite canali Bluetooth (in blu).

3.1.1 Thread

Main

Il thread main è quello che fa partire l'intera applicazione, si occupa di connettersi al Wiimote, di inizializzare tutte le strutture dati, di abilitare la telecamera IR e di rimanere in attesa di un input dall'utente. Nella fase di inizializzazione, come vedremo in seguito, richiede lo stato del Wiimote ed avvia la procedura di abilitazione della telecamera IR che abbiamo visto nella sezione 2.3.1, per entrambe queste operazioni deve comunicare con il

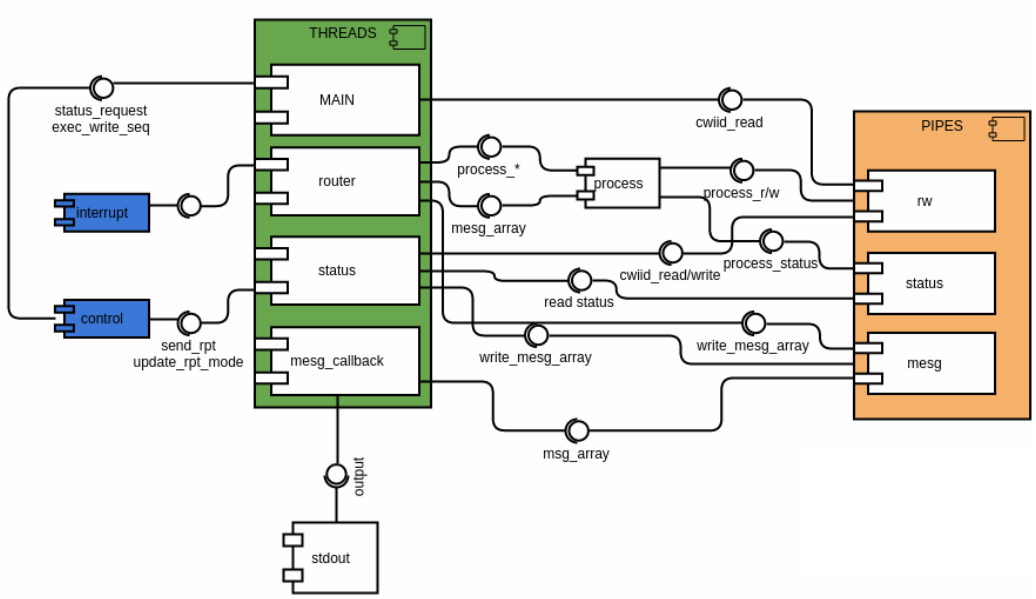

Figura 3.1: Struttura di libcwiid

Wiimote tramite il canale di controllo.
Si occupa anche di creare i canali Bluetooth e gli altri thread, nonché le pipe di comunicazione.

Router

Come dice il nome, questo thread si occupa di instradare i dati ricevuti dal Wiimote attraverso il canale Bluetooth interrupt, alle funzioni o ai thread che si occuperanno poi di analizzarli ed elaborarli.
Questo thread è l'unico che riceve i dati sul canale interrupt, quello riservato ai report di tipo `DATA` e `DATC`, creando così una struttura modulare ben organizzata. Router è il thread centrale durante l'esecuzione dell'applicazione, immette dati in tutte e tre le pipe, si occupa di inoltrare i messaggi alla funzione di callback e di mandare messaggi al thread status quando c'è bisogno di aggiornare lo stato del Wiimote.

Status

Questo thread è il meno utilizzato ed il più marginale, ma ricopre comunque un ruolo importante in quanto si occupa di aggiornare lo stato del Wiimote quando necessario.
Nonostante il thread main utilizzi il canale Bluetooth di controllo durante la fase di inizializzazione, esso rimane in attesa di un input utente per il resto dell'esecuzione.
Dopo l'inizializzazione sarà il thread status ad occuparsi del canale di controllo, leggendo i dati e inviando report quando necessario.
Anche il thread status scrive dati su tutte e tre le pipe, ma con meno frequenza.
Il thread status riceve i suoi comandi attraverso la pipe di stato e si occupa di gestire il cambiamento di stato aggiornando la struttura `wiimote_t` attraverso il modulo state.c.

Mesg_callback

Questo thread si occupa della funzione di callback definita dal programmatore.
È l'unico thread che può mandare in output delle informazioni (nessuno vieta di farlo anche negli altri thread, ma questo romperebbe la modularità della libreria, rendendo così il codice più difficile da leggere).
Il thread mesg_callback resta in attesa di un input da parte degli altri thread sulla pipe dei messaggi, quando riceve un messaggio chiama la funzione di callback passandoglielo come parametro insieme ad alcune informazioni aggiuntive.
Parleremo più avanti della funzione di callback, di cui si può trovare un esempio in wmdemo.c (all'interno di cwiid), una demo che utilizza la libreria libcwiid.

3.2 Analisi dinamica

I diagrammi 3.2 3.3 3.4 indicano le tre fasi principali del funzionamento di libcwiid quando viene utilizzato per leggere dati dalla telecamera IR:

1. l'inizializzazione,
2. l'abilitazione dei segnali IR,
3. la lettura dei segnali IR.

3.2.1 Inizializzazione

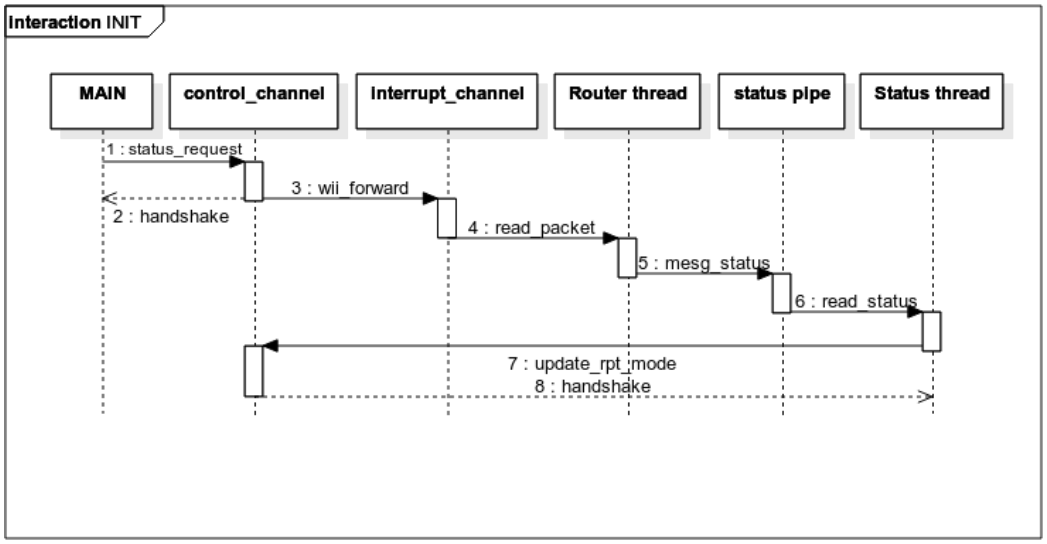

Figura 3.2: Diagramma di sequenza inizializzazione libcwiid

In questa fase, che inizia subito dopo la connessione, viene richiesto lo stato del Wiimote che servirà a popolare la struttura dati wiimote_t che vedremo in seguito, per far questo il thread main deve scrivere sul canale di controllo.
I thread sono già stati inizializzati, la risposta alla richiesta dello stato infatti arriverà sul canale di interrupt, verrà letta dal thread router che si occuperà di analizzarla e mandare un messaggio sulla pipe di stato in modo che venga

letto dal thread status e quindi elaborato.
Alla fine di questa fase il thread main si metterà in attesa di un input dall'utente.

3.2.2 Abilitazione dei segnali IR

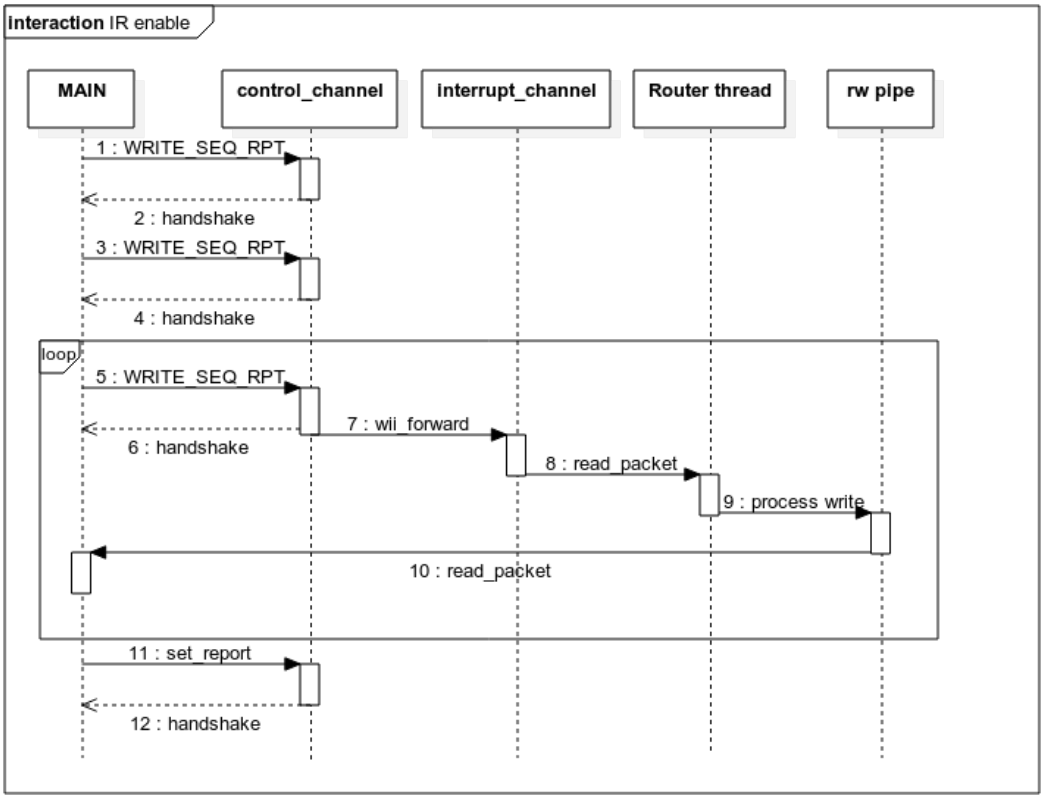

Figura 3.3: Diagramma di sequenza abilitazione libcwiid

Wmdemo.c permette all'utente, tramite terminale, di abilitare i segnali IR, lightboard lo fa in automatico durante la fase di inizializzazione.
In questa fase si attiva la procedura vista nella sezione 2.3.1 per attivare la ricezione dei dati della telecamera IR. Come abbiamo visto, la procedura richiede 7 passaggi, ma libcwiid usa soltanto i primi 6. Per fare questo il thread main manda i report necessari sul canale di controllo, attende i messaggi di handshake e, nel caso delle fasi 3-6, i dati ricevuti in risposta dal Wiimote

sul canale di interrupt vengono letti ed elaborati dal thread router.
Per fare questo il thread router utilizza una pipe dedicata, la pipe r/w (read/write).
Alla fine di questa fase inizieranno ad arrivare sul canale di interrupt i messaggi relativi ai dati IR raccolti dalla telecamera ad infrarossi del Wiimote.

3.2.3 Lettura dei dati IR

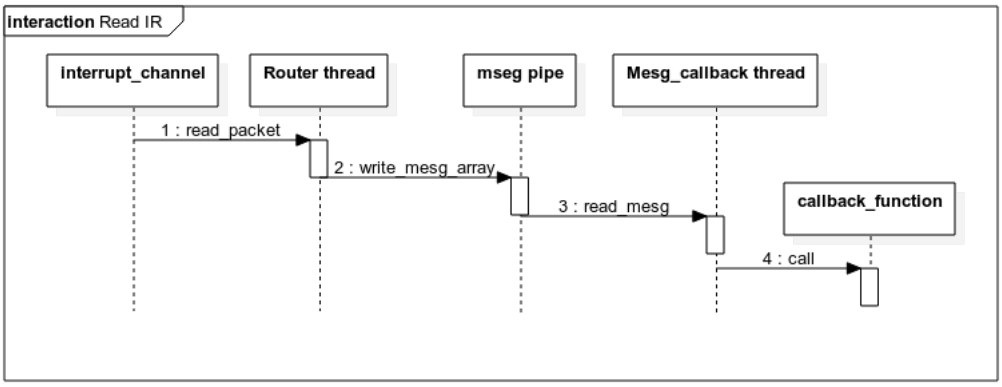

Figura 3.4: Diagramma di sequenza lettura libcwiid

In questa fase il thread status non fa nulla, ma può attivarsi in caso di un input da parte dell'utente, il thread main rimane in attesa ed il lavoro maggiore viene effettuato dal thread router e dal thread mesg_callback. Il thread router riceverà continuamente pacchetti, li analizzerà, li elaborerà tramite il modulo process.c ed infine li inoltrerà a mesg_callback tramite la pipe dei messaggi.
Questo permetterà a mesg_callback di occuparsi soltanto della chiamata alla funzione di callback definita dal programmatore.

Thread router in dettaglio

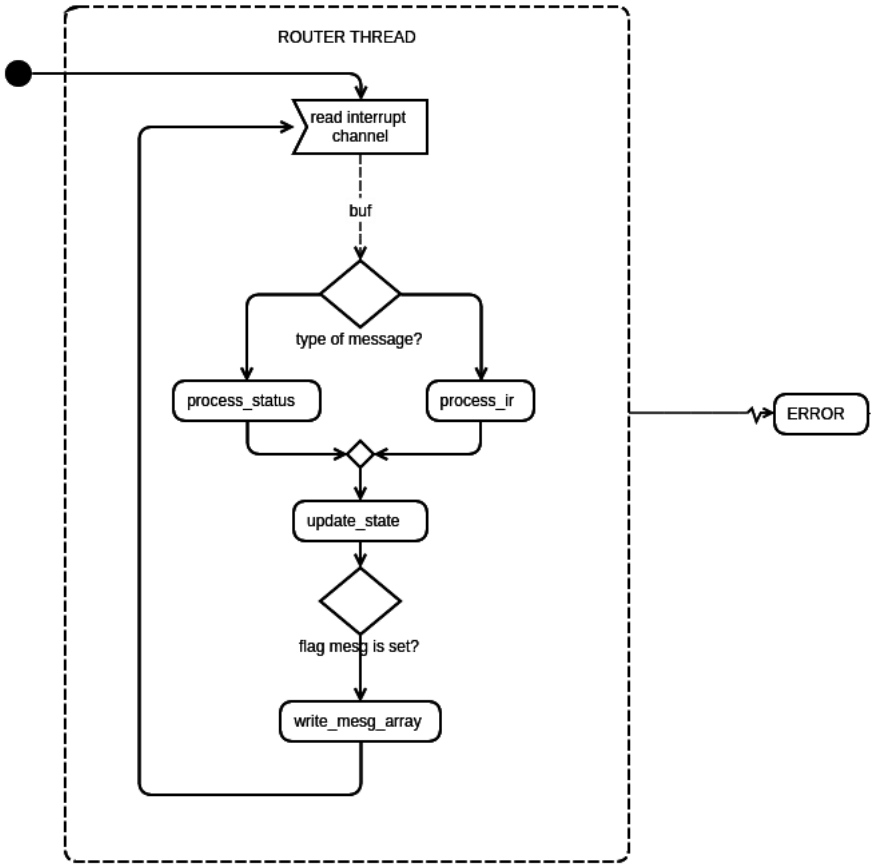

Figura 3.5: Diagramma delle attività del thread router

Nel diagramma in figura 3.5 vediamo il funzionamento del thread router. Come già anticipato, il thread router è composto da un ciclo che inizia con la ricezione di un messaggio sul canale di interrupt, in base al tipo di messaggio, chiamerà una funzione del modulo process.c adibita alla sua elaborazione. Dopodiché viene aggiornato lo stato interno, memorizzato nella struttura dati `wiimote_t`.
Infine, se si è abilitata la scrittura dell'output nel thread mesg_callback, verrà inviato un array contenente le informazioni analizzate, sulla pipe dei messaggi, al thread mesg_callback.

Thread status in dettaglio

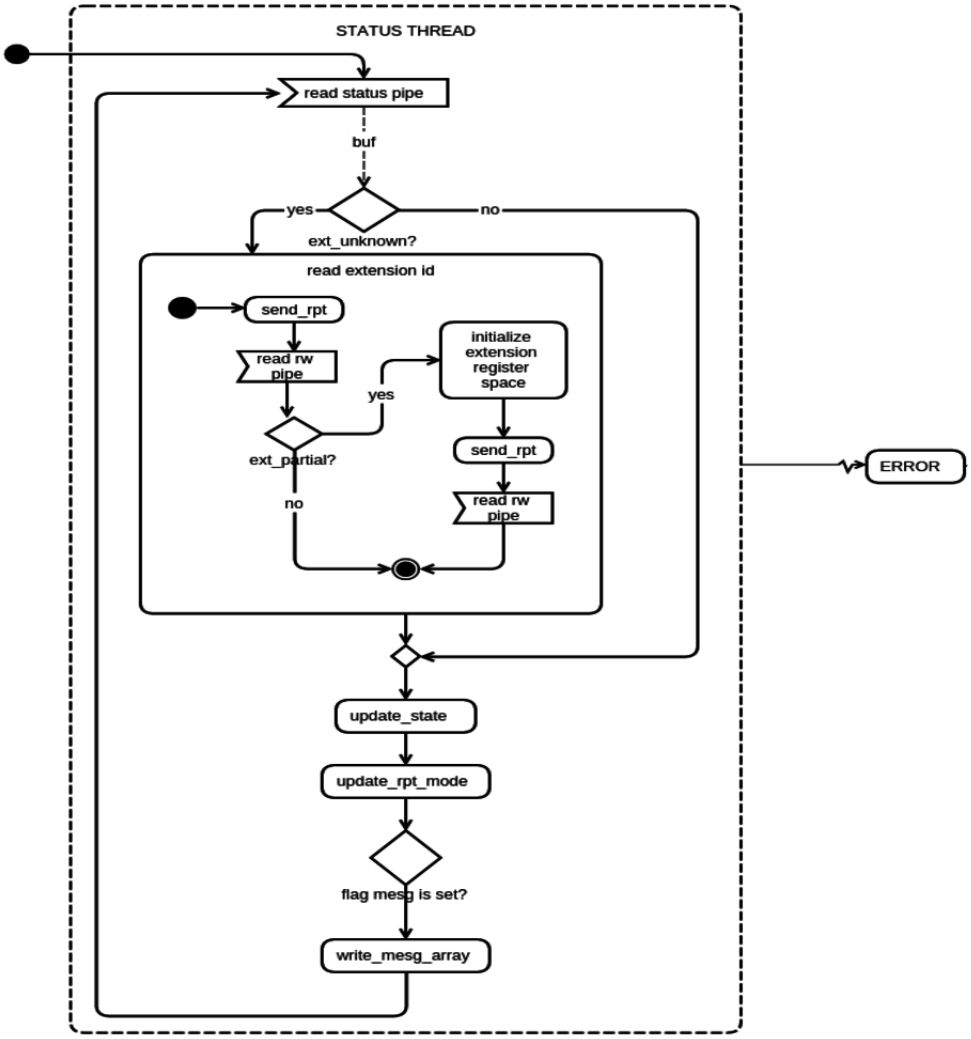

Figura 3.6: Diagramma delle attività del thread status

Nel diagramma in figura 3.6 vediamo invece il funzionamento del thread status. Il flusso è simile: viene letto un pacchetto, analizzato, elaborato, viene aggiornato lo stato e, se specificato, viene mandato un messaggio al thread mesg_callback.
In questo caso il messaggio viene letto dalla pipe status; come abbiamo visto nella sezione 2.2.1, il report `0x20` avrà nel bit 1 della maschera di bit `LF`, il

valore 1 se un'estensione è collegata, 0 altrimenti. Nel caso non ci sia nessuna estensione collegata, il messaggio ricevuto dal thread status avrà come tipo di estensione (`ext_type` nella struttura dati) la costante: `CWIID_EXT_NONE`.
Nel caso ci sia un'estensione collegata, il report `0x20` non dice nulla sulla natura di questa estensione, in tal caso il messaggio ricevuto dal thread status avrà come tipo di estensione la costante: `CWIID_EXT_UNKNOWN`. Il flusso continua controllando il tipo di estensione, se è sconosciuto (è collegata un'estensione, ma non si sa quale) si effettua una sotto-procedura per determinare il tipo di estensione, in caso contrario si salta questo passaggio.
Non si analizzerà questa fase in quanto non è di interesse per il progetto.
Non sono infatti necessarie estensioni e molte di esse hanno un'utilità solo quando è il Wiimote stesso ad essere utilizzato direttamente dall'utente, non quindi nel caso di lightboard (o di python-whiteboard). Dopodiché viene effettuato l'aggiornamento dello stato come nel caso del thread router e, se richiesto, l'aggiornamento della modalità di report (è qui che viene abilitata o disabilitata la telecamera IR).
Infine, se specificato, si può mandare un messaggio al thread mesg_callback in modo da poter riportare in output lo stato.

3.3 Strutture dati

Le strutture dati utilizzate dalla libreria libcwiid si trovano nei file `cwiid.h` e `cwiid_internals.h`.
Nel file `cwiid.h` sono presenti quelle strutture dati che servono a contenere i messaggi e gli stati inviati dal Wiimote, in `cwiid_internals.h` ci sono invece le strutture dati che servono alla libreria per le sue funzioni interne.
In entrambi i file sono definite le costanti, le union e le funzioni utilizzate nella libreria.

3.3.1 cwiid.h

In questo file le strutture interessanti sono due, la prima riguarda i dati IR e la seconda riguarda lo stato.

```
struct cwiid_ir_src {
    char valid;
    uint16_t pos[2];
    int8_t size;
};
```

Il campo size corrisponde ad una costante: `CWIID_IR_SRC_COUNT` che vale 4 (in lightboard modificata ad 1), che indica il numero di sorgenti IR da leggere dai report del Wiimote.
`valid` è un flag che indica se il dato è valido o meno, `pos` invece è un array che contiene le coordinate (`CWIID_X` e `CWIID_Y`) del punto.

```
struct cwiid_state {
    uint8_t rpt_mode;
    uint8_t led;
    uint8_t rumble;
    uint8_t battery;
    uint16_t buttons;
    uint8_t acc[3];
    struct cwiid_ir_src ir_src[CWIID_IR_SRC_COUNT];
    enum cwiid_ext_type ext_type;
    union ext_state ext;
    enum cwiid_error error;
};
```

Questa invece è la struttura relativa allo stato del Wiimote.
Qui viene utilizzata la struttura `cwiid_ir_src` per contenere l'ultimo dato IR letto, in particolare viene creato un array di `CWIID_IR_SRC_COUNT` strutture.
Gli altri campi indicano proprio quello che il loro nome suggerisce.

3.3.2 cwiid_internals.h

In questo file invece la struttura più importante è proprio quella che mantiene i dati riguardanti il Wiimote.

```
struct wiimote {
    int flags;
    int ctl_socket;
    int int_socket;
    pthread_t router_thread;
    pthread_t status_thread
    pthread_t mesg_callback_thread;
    int mesg_pipe[2];
    int status_pipe[2];
    int rw_pipe[2];
    struct cwiid_state state;
    cwiid_mesg_callback_t *mesg_callback;
    pthread_mutex_t state_mutex;
    pthread_mutex_t rw_mutex;
    pthread_mutex_t rpt_mutex;
    int id;
    const void *data;
};
```

Qui possiamo vedere che sono memorizzati i canali Bluetooth, i thread, le pipe, lo stato, l'id ed altre informazioni importanti per il programma.
Questa è senza dubbio la struttura dati più importante e centrale di tutta la libreria.

Capitolo 4

Python-whiteboard

Anche nel caso di python-whiteboard faremo un'analisi dinamica utilizzando un diagramma di sequenza per mostrare il flusso di attivazione degli oggetti ed i messaggi scambiati.

Prima di questo sarà necessaria un'analisi statica delle classi più importanti di python-whiteboard.

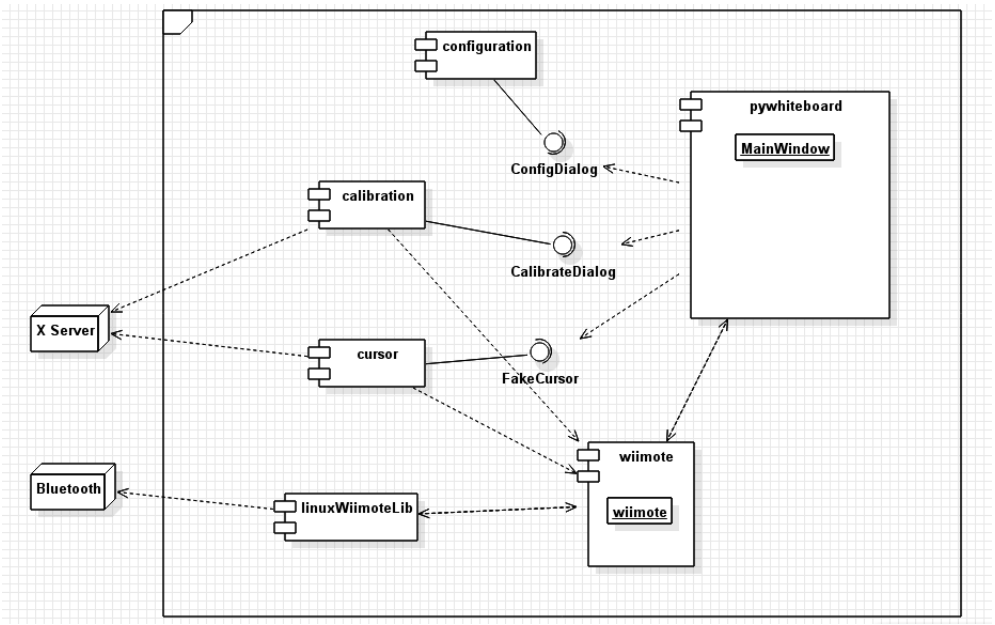

Figura 4.1: Struttura di python-whiteboard

Python-whiteboard si divide in moduli ben specificati, ogni modulo ha la sua funzione e le classi necessarie.

4.1 Analisi statica

pywhiteboard.py

Questo è il modulo principale, le sue classi sono:

- **MainWindow** - questa è la classe che gestisce la finestra principale del programma, è collegata ad un Wiimote ed in base al suo stato (connesso, calibrato, attivo) modifica l'interfaccia grafica. I suoi metodi permettono di gestire i profili e cambiare lo stato del Wiimote,
- **AboutDlg** - la schermata di informazioni generali sul programma,
- **PBarDlg** - la classe che gestisce la barra progressiva che permette di mostrare la connessione al Wiimote.

configuration.py

Questo modulo fornisce l'interfaccia di configurazione dell'applicazione, contiene due classi:

- **ConfigDialog** - la schermata di configurazione vera e propria, permette di accedere alle configurazioni tramite un'interfaccia grafica intuitiva,
- **Configuration** - implementata secondo il pattern *singleton*, questa classe contiene la configurazione corrente dell'applicazione e permette di leggere e scrivere valori.

calibration.py

Questo modulo si occupa di gestire la calibrazione del Wiimote, le sue classi sono:

- **CalibrateDialog** - questa è la schermata principale di calibrazione, gestisce i punti agli angoli dello schermo e la possibilità di avvicinarli o allontanarli,

- **SmallScreen** - questa piccola finestra si trova al centro della schermata di calibrazione e permette di vedere la posizione effettiva del punto luminoso nello schermo della telecamera IR,

- **SandClock** - si occupa dell'animazione di completamento dei punti agli angoli dello schermo.

cursor.py

Questo modulo contiene le classi necessarie al movimento ed al click del cursore sullo schermo, utilizza la libreria X e le sue classi sono:

- **FakeCursor** - il cursore vero e proprio, gestisce il movimento e la funzione di callback del Wiimote;

- **Click** - la funzione che permette il click del cursore, viene istanziata da FakeCursor all'occorrenza.

wiimote.py

Contiene un'unica classe: **Wiimote**, ed è l'unico punto di accesso alle informazioni sul dispositivo, permette di impostare una funzione di callback, calcolare i dati di calibrazione e salvarli nell'oggetto, recuperare informazioni sullo stato del Wiimote e creare i thread necessari all'esecuzione delle funzioni di callback.

linuxWiimoteLib.py

Questo modulo gestisce tutte le operazioni a basso livello della comunicazione con il Wiimote, le sue classi sono:

- **Wiimote** - manda e riceve report, permette la connessione del Wiimote, l'attivazione della telecamera IR e gestisce l'esecuzione della funzione di callback,
- **Parser** - comprende i report inviati dal Wiimote e restituisce oggetti ordinati con le informazioni ricevute,
- **WiimoteState** - memorizza lo stato del Wiimote e lo tiene aggiornato.

thread.py

Un modulo minore, estende il modulo QThread. I thread utilizzati in python-whiteboard si occupano delle seguenti azioni:

- la funzione di callback del Wiimote relativa alla ricezione dei dati IR,
- la connessione con il Wiimote,
- la gestione del cursore.

4.2 Analisi dinamica

L'applicazione comprende tre fasi: inizializzazione, calibrazione, lettura dati IR.

Inizializzazione

Come si può vedere dal diagramma in figura 4.2, l'oggetto centrale è di tipo `MainWindow`, esso crea un oggetto di tipo `Wiimote` ed uno di tipo `PBarDlg` per visualizzare la barra di caricamento della connessione con il Wiimote, viene quindi creato un thread apposito che si occupi della connessione. Quando il Wiimote è connesso viene distrutto l'oggetto `PBarDlg` ed il thread creato in precedenza attiva la telecamera IR per effettuare la calibrazione. Nel frattempo è stata creata la schermata di configurazione istanziando la classe `ConfigDialog`.

Calibrazione

A questo punto viene creato l'oggetto di classe `CalibrateDialog`, esso, oltre a creare la schermata di calibrazione, crea una funzione di callback apposita. Appena i punti rilevati sono quattro restituisce al chiamante e l'oggetto viene distrutto.
`MainWindow` allora chiama il metodo `calibrate` del Wiimote, che si occupa di effettuare i calcoli matriciali e memorizzare le informazioni di calibrazione nell'oggetto stesso.

Lettura dei dati IR

In questa fase viene creato un oggetto di tipo `FakeCursor`, questo oggetto si occuperà di creare un thread e la funzione di callback che esso dovrà eseguire, a questo punto il programma si mette in attesa di un dato IR e, quando lo riceverà, la funzione di callback si occuperà di calcolare le coordinate, passarle all'oggetto `FakeCursor` per spostare il mouse e, se specificato, effettuare il click istanziando la classe `Click`. Nel caso di lightboard, python-whiteboard è stato modificato il meno possibile, la fase di lettura dei dati IR è stata eliminata (viene gestita dall'engine in C) ed il bottone di attivazione (che nell'applicazione originale avvia proprio questa fase) scrive tutti i dati di configurazione nel file `.lbconfig.ini` contenuto nella cartella `ui/stuff`, poi chiude l'applicazione.

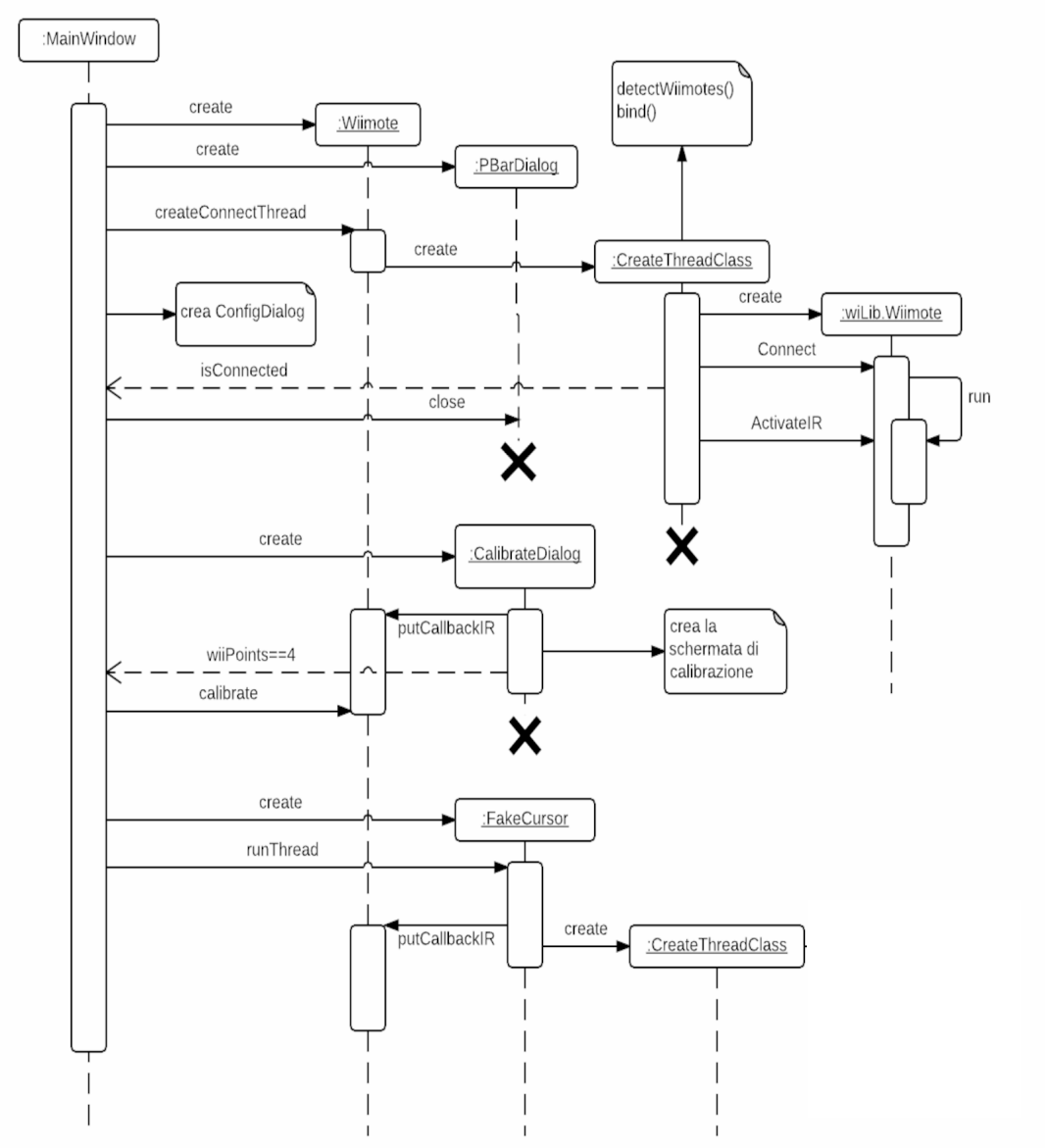

Figura 4.2: Diagramma di sequenza di python-whiteboard

Capitolo 5

Lightboard

Lightboard è il frutto di approfondite analisi delle prestazioni e studio delle applicazioni su cui si basa.
L'applicazione finale unisce l'interfaccia intuitiva di python-whiteboard, con la velocità di libcwiid, con alcune modifiche a quest'ultimo per ottimizzarlo ulteriormente.

5.1 Funzionamento

Come vediamo dal diagramma in figura 5.1, la prima cosa che fa lightboard è eseguire python-whiteboard. Per fare questo si avvale di una libreria sviluppata dal professor Renzo Davoli: *libs2argv* [9]. In questa libreria è presente un'API, `execs`, che ha lo stesso effetto di system, ma senza aprire una sotto-shell (che porterebbe ad alcuni problemi legati alla sicurezza). A questo punto si aprirà l'interfaccia grafica di python-whiteboard che permetterà all'utente di connettere il Wiimote ed effettuare la calibrazione. Effettuata la calibrazione si dovrà premere *Activate* per scrivere le configurazioni ed i dati di calibrazione su un file e ritornare a lightboard.
Il file creato avrà questo formato:

```
indirizzo Bluetooth del Wiimote
[[hxy]]
```

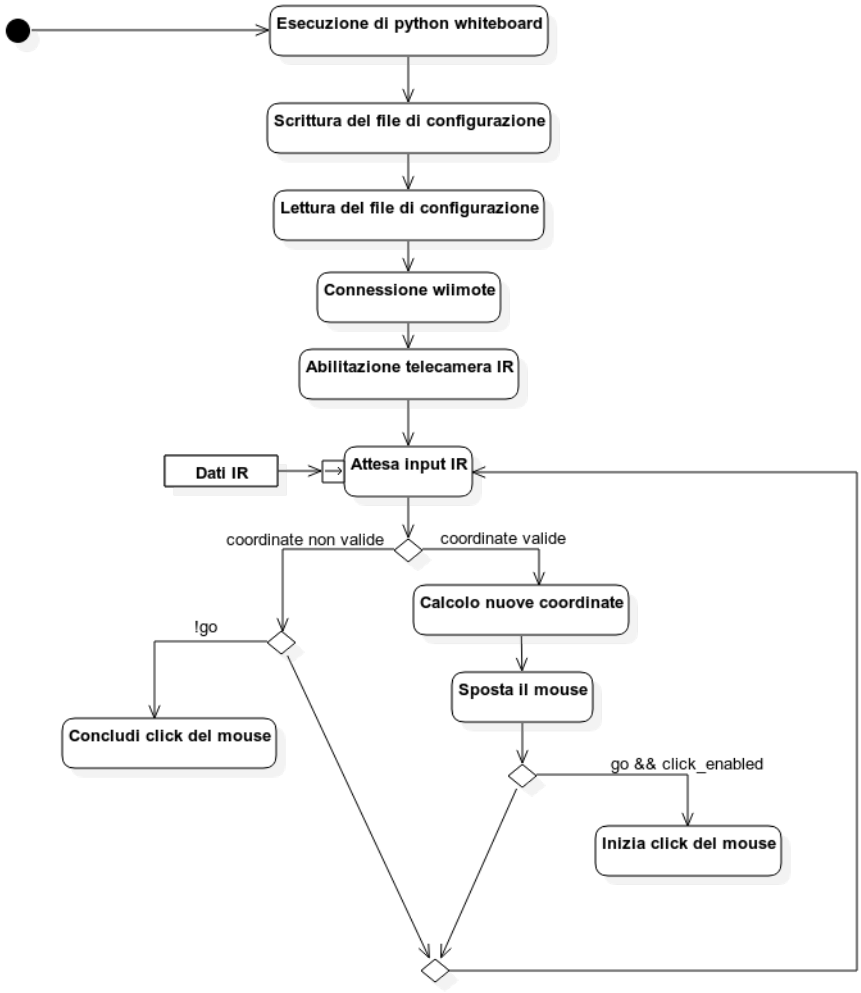

Figura 5.1: Diagramma delle attività di Lightboard

`click_enabled`

Dove `hxy` è l'elenco dei valori alla riga `x` e colonna `y` del risultato del sistema di equazioni calcolato durante la fase di calibrazione.
`click_enabled` è un valore binario, vale 0 se si è selezionata la modalità di puntamento, 1 per quella di click. Un esempio di file di configurazione potrebbe essere:

```
00:22:4C:9B:E7:88
[[ 1.08250275]]
```

```
[[-0.09788087]]
[[ 4.08139181]]
[[-0.25604773]]
[[-1.25427506]]
[[ 758.56401825]]
[[-0.00033507]]
[[-0.00018954]]
0
```

Lightboard si occuperà di leggere questo file e salvare i dati in apposite variabili, dopodiché si connetterà automaticamente all'indirizzo Bluetooth specificato ed abiliterà la telecamera IR.
A questo punto il programma funziona come definito nella sezione 3.2. Se le coordinate specificate nel report sono corrette, vengono calcolate le coordinate corrispondenti sullo schermo con queste espressioni:

```
new_x = ( (matr[0]*x) + (matr[1]*y) + matr[2] ) /
        ( (matr[6]*x) + (matr[7]*y) + 1 );
new_y = ( (matr[3]*x) + (matr[4]*y) + matr[5] ) /
        ( (matr[6]*x) + (matr[7]*y) + 1 );
```

Dove `matr[i]` corrisponde ad `hxy` in ordine per come sono stati presentati nel file di configurazione.
A questo punto, facendo uso della libreria XTest [10], si sposta il cursore nella posizione specificata.
Nel caso in cui il click non sia abilitato, la funzione di callback finisce qui e si aspetta il prossimo pacchetto.
Nel caso contrario invece l'idea è quella di iniziare il click (e mantenerlo) fintanto che arrivano pacchetti IR contenenti coordinate valide, appena arriva un pacchetto IR con coordinate non valide si rilascia il click del cursore.
Questo permette di emulare il funzionamento di un mouse nelle operazioni di click, doppio click e trascinamento.
In lightboard questo viene fatto con l'ausilio di una varibile chiamata `go`.

Bibliografia

[1] URL: http://wiildos.wikispaces.com.

[2] URL: https://learn.sparkfun.com/tutorials/bluetooth-basics.

[3] URL: http://pyqt.sourceforge.net/Docs/PyQt4/introduction.html.

[4] URL: https://gcc.gnu.org/onlinedocs/gcc-4.7.1/gcc/Optimize-Options.html.

[5] *Bluetooth*. URL: http://en.wikipedia.org/wiki/Bluetooth.

[6] Inc. Bluetooth SIG. *Baseband Architecture*.

[7] Inc. Bluetooth SIG. *Human Interface Device Profile*. URL: https://developer.bluetooth.org/TechnologyOverview/Pages/HID.aspx.

[8] Inc. Bluetooth SIG. *Logical Link Control*. URL: https://www.bluetooth.org/en-us/specification/assigned-numbers/logical-link-control.

[9] Renzo Davoli. *S2argv-execs: something was missing in libc*. URL: http://bit.ly/1uojZur.

[10] Kieron Drake. *XTEST Extension Library*. URL: http://bit.ly/16vBAVZ.

[11] HID_WG. *HUMAN INTERFACE DEVICE PROFILE*. 2012.

[12] Albert Huang. *An Introduction to Bluetooth Programming*. URL: http://people.csail.mit.edu/albert/bluez-intro/x148.html.

[13] Giovanni Incammicia. "Lightboard: un programma efficiente per costruire una LIM con Raspberry Pi e Wiimote". Università degli Studi di Bologna, 2015.

[14] Johnny Chung Lee. URL: http://johnnylee.net/projects/wii/.

[15] David Lippman. *Wiimotes and Robots*. URL: www.imathas.com/wiimotes/wiimotesrobots.ppt.

[16] *List of Bluetooth profiles*. URL: http://en.wikipedia.org/wiki/List_of_Bluetooth_profiles.

[17] *List of Bluetooth protocols*. URL: http://en.wikipedia.org/wiki/List_of_Bluetooth_protocols.

[18] Microsoft. *Introduction to HID Concepts*. URL: http://bit.ly/16vtnRN.

[19] *Qt (Software)*. URL: http://en.wikipedia.org/wiki/Qt_%28software%29.

[20] Deon Spengler. *Understanding and using htop to monitor system resources*. URL: http://bit.ly/1DKuxG6.

[21] *The X Window System*. URL: https://www.freebsd.org/doc/handbook/x-understanding.html.

[22] *USB*. URL: http://en.wikipedia.org/wiki/USB#Latency.

[23] *USB human interface device class*. URL: http://en.wikipedia.org/wiki/USB_human_interface_device_class.

[24] *Wii Remote*. URL: http://en.wikipedia.org/wiki/Wii_Remote.

[25] Wiibrew. *Wiibrew Wiimote*. URL: http://wiibrew.org/wiki/Wiimote.

[26] *Wiimote/Library*. URL: http://wiibrew.org/wiki/Wiimote_Driver.

[27] *X Window System*. URL: http://en.wikipedia.org/wiki/X_Window_System.