



UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE
Corso di Laurea Triennale in Informatica

**Physiradio:
Uno studio sulla Data
Physicalization tramite lo
sviluppo di un device IoT**

Tesi di Laurea di:
Simone Scaravati
Matr. 870883

Relatore:
Prof. Andrea Trentini

Anno Accademico 2019-2020

Indice

1	Introduzione	3
1.1	Data Physicalization	4
1.2	Analisi degli studi	7
1.3	Open Data	10
1.3.1	OpenWeatherMap e la scelta delle condizioni meteorologiche	12
1.4	“Musicalization”: il processo di physicalization tramite la musica	13
1.4.1	Aggiunta del colore alla fisicalizzazione	14
1.5	Il mapping attuato	15
1.6	IoT	20
1.6.1	La scelta di sfruttare l’IoT come base per la realizzazione di una fisicalizzazione	20
2	Physiradio: ideazione e realizzazione del device	22
2.1	Componenti Hardware	23
2.1.1	Scheda embedded	23
2.1.2	Decodificatore audio	25
2.1.3	Speaker e striscia led	26
2.1.4	Schema cablaggio	27
2.2	Software implementati e utilizzati	30
2.2.1	Il protocollo MQTT	30
2.2.2	Libreria TaskScheduler	34
2.2.3	Descrizione dell’interazione con il device	37

2.2.4	Physiradio MQTT Interface: applicazione per interagire col device	41
3	Field testing e analisi feedback	44
3.0.1	Somministrazione con questionario	44
3.0.2	Analisi dei feedback	48
4	Conclusioni e futuri sviluppi	52
A	Codice implementazione protocollo MQTT	58
B	Codice implementazione cooperative multitasking	61

Capitolo 1

Introduzione

La Data Physicalization è stata definita come

“Un’area di ricerca che esamina in quale modo, le rappresentazioni fisiche dei dati, supportate da computer, possano aiutare la cognizione, la comunicazione, l’apprendimento, il problem-solving e il decision making ” [18]

Nel corso degli anni, grazie alla pervasività, e alla differenziazione, delle tecnologie che ognuno di noi ha a disposizione, la rappresentazione dei dati ha assunto un ruolo sempre più importante in diversi campi di ricerca, come l’informatica, l’educazione o la comunicazione.

La *data physicalization*, essendo un’area di ricerca molto recente, sta acquisendo sempre più rilevanza in questi campi. Questo la porta ad essere aperta a nuove idee e spunti per essere arricchita ed ampliata, portando a sperimentare nuovi modi per poter rappresentare dei dati, di varia natura, attraverso metodi non convenzionali e che, rendendoli tangibili, non sfruttino solamente la vista come mezzo di cognizione ed interpretazione del dato.

Il lavoro svolto in questa tesi, è stato quello di studiare i vari aspetti riguardanti la “fisicalizzazione” dei dati, e ha portato conseguentemente allo sviluppo di un prototipo che potesse presentare uno spunto interessante per la *data physicalization*. Nello specifico si tratta di un device IoT, che prende il nome di Physiradio, il quale riproduce specifici flussi musicali secondo una

mappatura che trasla le condizioni meteorologiche, accessibili tramite API aperte (open data), in generi musicali e colori; processo che viene nominato *musicalization*.

L’obiettivo principale di questo esperimento è verificare se un tale dispositivo può essere uno spunto per aumentare la curiosità sul mondo della *data physicalization*, dei dati aperti, e in secondo luogo verificare se il genere musicale può essere utilizzato in una mappatura dei dati. Quest’ultima fase è stata eseguita tramite un *field testing*, dove il prototipo è stato sottoposto ad un gruppo di utenti, che hanno compilato un questionario, da cui è stato possibile estrarre dati utili all’analisi dell’efficacia del device e della curiosità generata.

1.1 Data Physicalization

La *data physicalization*, che nel corso di questo elaborato verrà a volte tradotta (senza alcun riferimento a lavori altrui) in “fisicalizzazione dei dati”, è un’area di ricerca multidisciplinare moderna, basata sulla rappresentazione fisica di dati, di qualsiasi natura e tipo.

Le rappresentazioni dei dati tramite supporti fisici esistono da secoli, ricercabili persino nell’antichità. Ad esempio nella Figura 1.1 possiamo osservare come già nella Mesopotamia del 5500 a.c., venissero utilizzati oggetti di varia natura (denominati successivamente, dagli archeologi, “token” [1]), con un ordine ben specifico, per avere un’idea chiara e immediata del conteggio di merci o tasse.

Come questo esempio, ce ne sono molti altri, consultabili sul sito ufficiale dedicato alla *data physicalization* [10], gestita da Yvonne Jansen, che è la referente principale a livello internazionale per quanto riguarda questa branca di ricerca.

Entrando nello specifico della fisicalizzazione dei dati. Essa potrebbe essere confusa con la *data visualization*, ovvero la mera visualizzazione dei dati, cosa a cui tutti noi siamo abituati quotidianamente, basti pensare a quanti grafici, animazioni e numeri vediamo ogni giorno sui nostri smartphone o

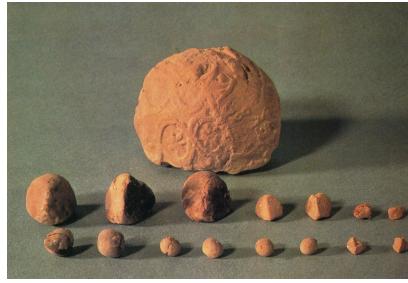


Figura 1.1: Token mesopotamici, fonte [1]

computer. In effetti, entrambe si riferiscono a modi per rappresentare dei dati. La *data visualization* cerca un modo di comunicare i dati attraverso l’uso di una mappatura sistematica tra segni grafici e valori dei dati nella creazione della visualizzazione. Questa mappatura stabilisce come i valori dei dati verranno rappresentati visivamente, determinando come e in che misura una proprietà di un segno grafico, come la dimensione o il colore, cambierà per riflettere le variazioni del valore di un dato.

Partendo da questo concetto, la *data physicalization* si spinge oltre[18], domandandosi se esistano altri tipi di rappresentazioni, che non si limitino solamente alla raffigurazione grafica.

Il punto chiave insita nel fatto che, nelle visualizzazioni, viene sfruttata solamente la vista come mezzo di trasmissione e cognizione dell’informazione, mentre potrebbe essere ancora più utile e di maggiore interesse cercare di sfruttare anche altri sensi del corpo umano.

A tal proposito, si possono citare Emerge [41] e TastyBeats [21] (Figura 1.2). Il primo è un insieme di barrette posizionate verticalmente, sopra un sistema di pistoni motorizzati, che permettono di vedere e interagire, tramite quindi l’uso della vista e del tatto, con dei dataset di varia natura. Il secondo invece, consta di un sistema di piccole pompe idrauliche, tramite cui è possibile creare un energy drink le cui componenti variano dinamicamente a seconda degli allenamenti fisici che vengono svolti, rappresentando di conseguenza dati relativi ad attività motorie attraverso il gusto.

Le recenti creazioni di interfacce tangibili, tecnologie pervasive avanza-



Figura 1.2: Esempi di *data physicalization*. Emerge (sinistra) e TastyBeats (destra), fonti [41] e [21]

te e la distribuzione sempre più diffusa di sistemi e componenti integrati, hanno portato allo sviluppo di creazioni “moderne” per rappresentare i dati, attraverso strumenti informatici associati a sensori, attuatori, ecc... , che permettono di avere rappresentazioni di carattere dinamico e interattivo; caratteristiche non molto presenti in progetti sviluppati precedentemente.

Secondo [18] la *data physicalization* può:

- “Aiutare le persone a esplorare, comprendere e comunicare i dati, utilizzando le rappresentazioni fisiche, basate sui computer, dei dati”;
- Rendere i dati più accessibili/raggiungibili a tutti;
- Favorire benefici cognitivi;
- Democratizzare i dati nel mondo reale;
- Coinvolgere le persone.

L’aspetto della democratizzazione è proposto anche in [45] dove i “widget domestici” vengono utilizzati per supportare con successo la creatività domestica e la co-creazione delle rappresentazioni di dati.

In [42] “DIY, hacking and craft” sono suggeriti come potenti strumenti di democratizzazione. I dispositivi IoT (Internet of Things) sono sfruttati in [14] in riferimento a “Il potenziale per democratizzare l’accesso, l’uso, e appropriazione dei dati”, poiché “la maggior parte dei dati è una ‘scatola nera’ in natura: gli utenti spesso non sanno come accedere ai dati o interpretarli”. Questi dispositivi “integriti nelle case”, possono essere utilizzati

per coinvolgere utenti non tecnici, il che è stato uno spunto importante per lo sviluppo del prototipo.

Può la creazione di un dispositivo fisico familiare e non minaccioso (ad esempio una radio dall'aspetto vintage) essere un veicolo di trasferimento delle conoscenze? Potrebbe, un oggetto del genere, essere in grado di suscitare curiosità sulla fonte e la disponibilità di alcuni dati (ad esempio il meteo atmosferico) e le tecniche alla base dell'accesso, dell'estrazione e dell'analisi dei dati?

1.2 Analisi degli studi

Il primo step del lavoro è stato di quello di effettuare un'analisi della *state of the art*. In particolare, si è scelto di focalizzarsi sui paper riguardanti prototipi/prodotti fisici effettivamente funzionanti, che potessero conferire una reale implementazione del concetto di *data physicalization*, incentrandosi sul tipo di messaggio che viene trasportato e tramite quali sensi.

Come fonte principale dell'analisi, è stata utilizzata la lista di documenti elencati nella bibliografia del sito ufficiale per la *data physicalization* [17].

Sono stati analizzati tre principali fattori per ogni prototipo/progetto presentato:

1. Quali sensi umani sono stati sfruttati
2. Il livello di interazione con l'oggetto
3. Il livello di dinamicità dei dati

Riguardo all'analisi svolta: sebbene vi sia una consapevolezza sul fatto che questa analisi possa essere soggettiva, in riferimento all'atto di leggere e interpretare gli articoli, si è cercato di renderla il più “meccanica” possibile, adottando i seguenti criteri:

1. SENSES: valore booleano per indicare quali sensi sono stati sfruttati (SIGHT=vista, TOUCH=tatto, HEARING=udito, TASTE=gusto,

SMELL=olfatto) dove 1 = il senso è stato sfruttato e 0 = il senso non è stato sfruttato.

2. INTERACTION: il livello di interazione da parte dell'utente con l'oggetto, espresso con tre valori numerici interi distinti:

- 0 = nessuna interazione, la creazione si limita ad essere osservata.
- 1 = l'interazione col prototipo cambia i parametri di fisicalizzazione, cioè come vengono rappresentati i dati, ma questi ultimi rimangono invariati.
- 2 = l'interazione col prototipo cambia anche il dataset di riferimento.

3. DYNAMICITY: flag booleano tradotto in STATIC(falso/0) e DYNAMIC(vero/1), dove DYNAMIC indica che vi è una costante connessione tra il device e i dati, ovvero che se i dati cambiano, cambia anche la fisicalizzazione (*real time*). STATIC indica che i dati di riferimento sono statici.

Da questa analisi è stato prodotto un dataset, pubblicato su Zenodo[35], da cui è possibile estrarre informazioni utili.

I risultati (Figura 1.3) mostrano che i sensi più utilizzati sono la vista e il tatto; mentre l'udito, il gusto e l'olfatto vengono raramente sfruttati.

Questo è stato uno spunto molto importante per la realizzazione del device, perché ci si è chiesto se esistesse un modo per trarre vantaggio da questi sensi “minori”.

Riconoscendo le difficoltà tecnologiche e creative nello sfruttare l'olfatto e il gusto come sensi per una fisicalizzazione, il senso rimanente era l'udito: in questo modo nacque l'idea di usare la musica come mezzo di comunicazione del dato.

In effetti, la musica fa parte della vita di tutti: lo si voglia o meno, tutti noi la ascoltiamo. Ed essa riesce a stimolare emozioni e stati d'animo, che possono essere utili per canalizzare dei dati. Sebbene la percezione di ogni canzone, di qualsiasi genere e artista, sia diversa per ognuno di noi, il pensiero di usare la musica come mezzo per fisicalizzare i dati, ovviamente attraverso l'udito, era convincente.

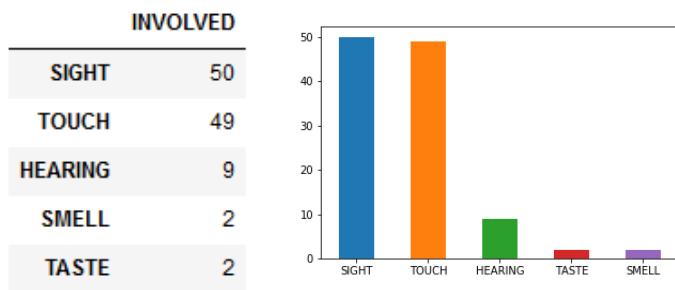


Figura 1.3: Sensi utilizzati nei dispositivi analizzati inerenti alla *Data Physicalization*. Casi totali: 51

Naturalmente, la musica può portare soggettività nell’interpretazione e introduce diversi problemi nella classificazione, dovuti a diverse variabili. Tuttavia, lo scopo principale del device non sarà quello di creare il miglior modo in assoluto per poter fisicalizzare dei dati, ma sarà quello di proporre una soluzione che stimoli la curiosità degli utenti. Di conseguenza, anche se il metodo utilizzato per la *physicalization* non sarà perfetto, comunque potrebbe essere sufficientemente valido per il suo scopo.

Successivamente, sono stati considerati gli altri due fattori: l’interattività con gli oggetti e la dinamicità dei dati usati dai dispositivi. L’analisi dei documenti, mostra livelli bassi (Figura 1.4) per entrambi gli aspetti. La maggior parte dei dispositivi utilizza dati statici (circa l’80%), mentre l’interazione fisica con l’oggetto risulta spesso inesistente e si limita alla mera osservazione (interazione= 0 in circa il 57% dei casi), il 23% propone un primo livello di interazione e solo il restante 20% presenta un’interazione tale da riuscire a cambiare i dati alla fonte.

Pertanto, partendo dai risultati di questa ricerca, per la creazione del device si è optato per una soluzione in grado di gestire i dati in tempo reale (quindi dinamici), e dove l’esperienza di interazione con esso sia concreta, tale che non ci si limiti solamente ad osservare il dispositivo.

In seguito sono stati individuati due punti critici per la creazione del device: quali dati usare e come usarli in funzione alla musica.

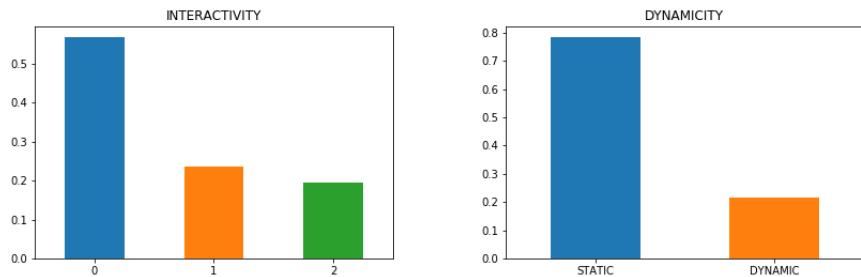


Figura 1.4: Livelli di interattività con i device e dinamicità dei dati. Casi totali: 51

Vedremo nei prossimi capitoli come, per il primo caso, ci si sia indirizzati verso il mondo dei dati aperti, per aumentare ulteriormente il grado di curiosità dell’esperienza e per cercare di avvicinare, al mondo della fruizione dei dati liberi, anche i non addetti al settore.

Per il secondo punto invece, l’utilizzo della musica si focalizzerà sull’utilizzare i generi musicali per eseguire la mappatura dei dati per la *physicalization*. Si elencheranno i pregi e i difetti di questa scelta, e il perché è stata presa.

1.3 Open Data

Gli open data sono diventati un movimento mondiale che coinvolge attori governativi e non governativi. La Open Knowledge Foundation (OKF, [27]) è stata una delle prime organizzazioni a definire “apertura” in questo contesto, la sua definizione può essere citata come: “Dei dati sono aperti se qualcuno è libero di usarli, riutilizzarli e ridistribuirli, con al più l’attribuzione della creazione” (vale a dire che i dati possono essere utilizzati apertamente in termini di licenze). Il “HM Government’s Open Data White Paper” [13] afferma che il Open Government Data consta di “Informazioni del settore pubblico che sono state rese disponibili al pubblico come dati aperti ” e definisce le Informazioni del Settore Pubblico (PSI) come “dati e informazioni prodotti, raccolti o detenuti da autorità pubbliche, nell’ambito del loro compito pub-

blico”, dati che dovrebbero essere accessibili (idealemente via Internet) a costi marginali (se non nulli) e senza discriminazioni, disponibili in un formato digitale e leggibile da una macchina, e fornito senza restrizioni sull’uso o sulla ridistribuzione.

Gli open data sono un concetto tecnico, il quale sta anche diventando rilevante dal punto di vista politico, poiché i dataset (su ambiente, imprese, ecc...) possono essere utilizzati per verificare o falsificare le politiche governative (*ex ante* ed *ex post*): i dati aperti possono essere un mezzo per la “responsabilità civica”. Qualsiasi cittadino, con sufficiente conoscenza, è in grado di recuperare dati (che possono essere ritenuti più affidabili se provengono da fonti di terze parti) da server pubblici e studiare l’effetto di leggi come, ad esempio, un cambiamento della tassazione del tabacco sul numero di fumatori, il divieto di alcuni tipi di veicoli in base alla qualità dell’aria [44], l’introduzione di una legislazione specifica per ridurre il numero di persone disoccupate, ecc...

Tuttavia, “l’analisi dei dati non è per le masse” [31]: il cittadino medio non è a conoscenza dell’esistenza di dati aperti e/o non è in grado di estrarre informazioni da un dataset.

Considerando gli obiettivi del device che verrà creato, ossia aumentare l’interesse e la curiosità nel frutto della fisicalizzazione di un dato, il fatto di poter usufruire di dati aperti e liberi può essere un valore aggiunto, specialmente per i non addetti ai lavori.

Anche se alla fine l’utente deciderà di non imparare come analizzare i set di dati, almeno avrà avuto la possibilità di ragionare sull’occasione di sfruttare il suo diritto alla “responsabilità civica”, con l’aiuto di una persona specializzata, o un oggetto, che gli permetta farlo al posto suo; strumenti come Physiradio sarebbero molto calzanti.

1.3.1 OpenWeatherMap e la scelta delle condizioni meteorologiche

Una volta analizzati gli open data e le loro potenzialità, ci si è chiesto quali dati effettivamente prelevare.

Il problema maggiore era legato al fatto che, per un primo prototipo da sviluppare, era necessario scegliere dei dati aperti che fossero di facile comprensione, per avere un esempio di funzionamento intuitivo anche da persone non provenienti da studi scientifici/tecnichi.

Dopo alcune ricerche e analisi, le condizioni meteorologiche sono risultate la scelta più naturale e utile allo scopo, perché sono qualcosa alla portata di tutti e di cognizione immediata.

Per quanto possano sembrare un’informazione inutile¹, tuttavia, nel contesto di questo esperimento, è stato un buon punto di partenza per conversazioni e stimoli a suggerimenti che sono arrivati da parte degli utenti (vedere nelle conclusioni, sezione 4). Inoltre, come vedremo nei capitoli successivi (legati ai software implementati), comunque il prototipo non si limiterà a descrivere il tempo di una sola città, ma è comunque interrogabile per analizzare tutte le città registrate nei server della piattaforma utilizzata.

Si è deciso così che Physiradio si basi su OpenWeatherMap[22], una piattaforma di dati aperta che fornisce molti servizi meteorologici. Per interagire con i suoi dati, essa fornisce delle API (Application Programming Interface) di tipo REST, che consentono l’accesso, via software, ai dati meteorologici in tempo reale.

In particolare, i dati restituiti sono sotto forma di file JSON, il che è molto utile: non solo sono di facile interpretazione da parte di moltissimi sistemi, ma numerosi dataset di open data utilizzano questo formato. Questo implica che Physiradio potrà essere agevolmente modificato/migliorato per sviluppi futuri, sfruttando altri dati da rappresentare.

¹Citando Robin Williams nel film “Good Morning Vietnam”: “Com’è il tempo? Hai una finestra? Aprila!”

1.4 “Musicalization”: il processo di physicalization tramite la musica

Successivamente, ci si è concentrati sull'utilizzo della musica come mezzo principale per questo esperimento di *data physicalization*.

La musica è un mondo molto vasto, con una semantica tutta propria, complessa ed articolata. Essa può trasmettere una vasta gamma di emozioni, sentimenti e stati d'animo. Quando si tenta di analizzare un brano, ci sono molti parametri da considerare: tempo (bpm), modo (maggiore/minore), *pitch*, *loudness*, ecc... Inoltre, è complicato valutare quale canzone è più adatta ad un particolare contesto come presentato in [12], dove viene descritto un approccio alla musica rispetto alla classificazione delle emozioni. Un altro aspetto fondamentale è che i testi di una canzone possono trasmettere informazioni all'ascoltatore (se le parole sono comprese, ovviamente), ma la musica e i testi possono essere discordanti per l'umore che la canzone vuole canalizzare², causando problemi a qualsiasi sforzo di classificazione.

Considerando tutti questi aspetti della musica, è evidente che vi sono delle difficoltà non irrilevanti nel tentativo di creare una classificazione tale che sia percepita in maniera totalmente oggettiva e uguale da tutti. Tuttavia, prendendo atto che l'obiettivo principale dell'esperimento non è quello di creare un mapping perfetto, si è deciso di perseguire comunque questa strada, adottando i criteri che seguono.

Esistono tecniche specifiche per trasferire informazioni tramite la semplice generazione di singoli suoni, come la cosiddetta *sonificazione* [34]. Abbiamo degli esempi in [6] e [23], dove vengono presentate tecniche di tracciamento del volto e di sintesi del suono per “sonificare” le espressioni facciali al fine di aiutare le persone con problemi visivi e un sistema di riferimento per in-

²ad esempio, “Some Nights” dei Fun è famosa per questa caratteristica, avendo un arrangiamento musicale dai toni allegri e vivaci, mentre il testo narra di tragedie avvenute in tempi di guerra.

interpretare modelli già esistenti di sonificazione e propone modelli futuri. Un suono semplice e costante, però, non può essere ascoltato per un lungo periodo di tempo, perché può risultare molto fastidioso. Di conseguenza, quando si deve usare una tecnica più morbida e sopportabile (per l'ascoltatore), entra in gioco la *musificazione*.

La musificazione è stata definita come la rappresentazione musicale dei dati. È pensata per andare oltre la sonificazione diretta: essa include elementi di tonalità e l'uso di scale modali per costruire composizioni musicali (vedere [8]), che generano strutture musicali (spartiti) con delle caratteristiche di livello superiore, come la polifonia o la modulazione tonale, per intrattenere l'ascoltatore di più, e per più tempo, rispetto al caso della sonificazione.

La musificazione e la sonificazione hanno una caratteristica in comune, ovvero il fatto di essere deterministici nei risultati: dato lo stesso input, l'output, sia esso un suono, uno spartito o una traccia musicale, sarà sempre lo stesso. Questo determinismo può risultare a sua volta “noioso”, se applicato in un contesto di vita quotidiana.

Physiradio cerca di ridurre al minimo questa potenziale sensazione di noia, ampliando l'associazione “dato → musica” introducendo l'idea di “*musicalizzazione*”; il termine è stato introdotto in questa tesi senza alcun riferimento a lavori altrui. Invece di generare suoni/musica ex novo, Physiradio sceglie e riproduce flussi musicali in streaming disponibili sulla rete, i quali sono etichettati in base al genere musicale (maggiori dettagli a riguardo nella sezione 1.5). In sostanza, il termine musicalizzazione si riferisce all'associazione tra valori di dati e brani musicali, che cambiano nel tempo, introducendo nel sistema un grado (potenzialmente alto, a seconda del flusso) di non-determinismo “anti-noia”.

1.4.1 Aggiunta del colore alla fisicalizzazione

Durante lo sviluppo del prototipo, analizzando lavori fatti da altri studenti/ricercatori, è stato pensato di aggiungere un altro senso sfruttato dalla fisicalizzazione, ossia la vista. La sua introduzione è dovuta a molteplici

fattori: in primis, parliamo del senso più comunemente usato nella rappresentazione dei dati (come anticipato nella sezione 1.2), dovuto anche alla vasta presenza delle *visualization* in molti contesti a cui noi tutti siamo abituati. Di conseguenza l’utente medio è più abituato, molto probabilmente, a ricorrere alla vista, rispetto ad altri sensi, nelle prime fasi di interpretazione di qualsiasi dato. In secondo luogo, la vista, è stata ritenuta un utile sostegno all’udito perché, con una giusta legenda/didascalia (o comunque una conoscenza pregressa del mapping), rende più veloce ed intuitiva l’associazione del dato. Perciò questo fattore, combinato con i generi musicali, potrebbe portare semantica e aiutare l’utente ad interagire meglio con il dispositivo, nell’interpretazione dei dati, aumentandone la leggibilità.

Per inserire la vista come ulteriore senso impiegato da Physiradio si è scelto di sfruttare i colori, in particolare una striscia di LED RGB che emette un singolo specifico colore che, a seconda della condizione atmosferica trovata, in combinazione con uno specifico genere musicale, formano una coppia ben precisa di dati, usata nel mapping (paragrafo 1.5).

Per introdurre i colori, è stato fatto affidamento sul modello di Robert Plutchick [29], in particolare è stata presa come riferimento la sua ‘ruota delle emozioni’ (Figura 1.5), perché è considerato uno dei più importanti studi psicologici sulle emozioni umane mappate sui colori. In combinazione con [7], uno studio che propone un abbinamento tra emozioni (*mood* nello specifico) e generi musicali, crea la giusta base per poter sviluppare un mapping adeguato per il nostro device IoT.

1.5 Il mapping attuato

Una volta prese le scelte riguardo a quali dati fisicalizzare (tempo atmosferico) e come fisicalizzarli (generi musicali e colori), si è dovuta affrontare la problematica del mapping da utilizzare.

Lo schema di funzionamento di Physiradio è il seguente: ottiene i valori delle condizioni meteorologiche (tramite le API di OpenWeatherMap) di una città configurata via software, li elabora estraendo solamente la descrizione prin-

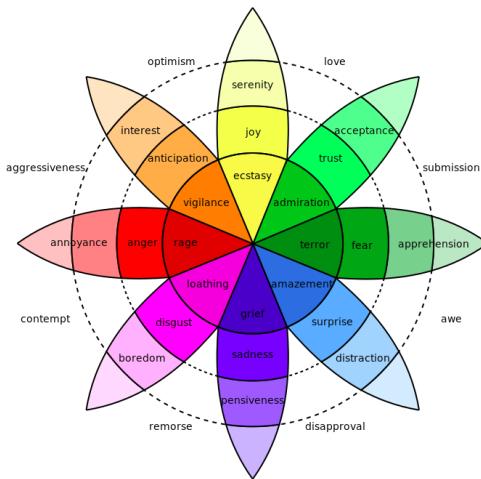


Figura 1.5: Ruota delle emozioni di Plutchik, fonte [48]

cipale e il livello di umidità relativa, quindi li mappa su una combinazione di genere musicale e colore. La funzione di mappatura è così fatta:

$$\text{map}(\text{Descrizione Condizioni Metereologiche}, \text{Umidità relativa}) \rightarrow (\text{Genere musicale}, \text{Colore})$$

Di seguito, vengono spiegate le motivazioni principali del perché sono stati scelti questi valori.

Il concetto base da cui si è partiti, è stato quello di avere come filo conduttore tra tempo atmosferico (umidità inclusa), genere musicali e colori, le emozioni/*mood* che ognuno di essi poteva scaturire nell'utente medio. Di conseguenza, ci si è indirizzati verso lavori che cercassero una corrispondenza con le emozioni per ognuno dei tre componenti.

Per implementare questa funzione di mappatura sperimentale, è stato esaminato uno studio di riferimento [20]. In esso viene descritto il modello di un selettore musicale basato sulle condizioni meteorologiche, che è ciò che più si avvicina al device che si è voluto implementare. Nello specifico vengono presi in considerazione molti parametri sia per quanto riguarda la musica, che

per il tempo atmosferico: per la prima si considerano modo, tempo, tonalità, ritmo, armonia e dinamica; mentre si considerano temperatura, umidità, pressione, vento, sole, nuvolosità e precipitazioni per il tempo atmosferico. Tuttavia, per la funzione di mapping implementata da Physiradio è molto difficile, se non impossibile, tenere in considerazione tutti questi fattori. Per giunta non è questo il fulcro della ricerca, avendo come obiettivo quello di aumentare la curiosità sulla *data physicalization* e sui dati aperti tramite un device IoT *user friendly*, non è necessario avere un mapping perfetto. Si è dovuta perciò effettuare un scrematura.

Per il tempo atmosferico, oltre alla descrizione della condizione corrente, si è considerato unicamente il valore dell'umidità relativa, poiché era l'unico valore estraibile che potesse essere utile a livello “superficiale” (inteso come immediatezza di ricezione da parte dell'utente), come descritto in [33], si è deciso di settare una soglia tale per cui il livello di umidità rende confortevole o meno il clima, in particolare si è deciso di applicare questa soglia ai tempi atmosferici ‘Sereno’ (soleggiato piacevole contro afa) e ‘Pioggia/Pioggerella’ (pioggia leggera contro pioggia abbondante). Basandosi sia sull'articolo poc'anzi citato che su test empirici, si è deciso di settare questa soglia a 85% di umidità relativa.

Per quanto riguarda la musica invece, una prima implementazioni si era pensata tramite l'utilizzo di app di streaming musicali (come Spotify o Apple Music) per poter avere una scelta specifica delle tracce da selezioni (con un tagging potenzialmente maggiore). Tuttavia, ovviamente, quei servizi a pagamento non permettono di accedere direttamente a flussi audio per ovvi motivi commerciali.

Di conseguenza si è dovuti ripiegare su una soluzione effettivamente implementabile. La scelta è ricaduta sui generi musicali come macro classificazione per la “musicalizzazione”, poiché vengono usati streaming di web radio liberamente utilizzabili, alcune delle quali sono estremamente selettive nel catalogo musicale (ci sono stazione che trasmettono solo Classica, altre solo Metal, ecc...), perciò permettono effettivamente di fare una selezione per genere.

La scelta dei colori invece come anticipato nella sezione precedente 1.4.1, si basa sullo studio delle emozioni di Plutchick[29].

Per conciliare genere musicali e colori, e quindi ottenere la coppia di valori usata nella funzione di mapping, i mezzi utilizzati sono appunto il *mood* e le emozioni che entrambi trasmettono all'uomo. Studi precedenti, come [7], analizzano come i generi musicali ispirino stati d'animo specifici alle persone.

Ciò nonostante, nello sviluppo del prototipo, sono stati necessari alcuni interventi, soggettivi e scarsi di fonti, riguardo alla scelta dei generi musicali usati.

In generale, i generi musicali moderni, come LoFi, ChillOut, Smooth Jazz e vari tipi di metal estremo, non sono stati menzionati in precedenti lavori accademici, tra quelli trovati ed esaminati durante lo svolgimento di questo studio. Tuttavia, questi generi si ritiene che siano molto suggestivi ed estremamente specifici: la maggior parte delle canzoni appartenenti a quei generi suoneranno molto simili tra loro, il che è utile se vogliono trasmettere le stesse informazioni ma con canzoni diverse.

Di conseguenza, qui di seguito, vengono riportati i principali criteri per poter associare i tempi atmosferici a determinati generi musicali. In particolare, vi sono due casi di condizioni atmosferiche che sono stati implementati secondo criteri personali: Il primo riguarda la condizione “Neve”, la quale sembrava utile e funzionale, quasi in modo naturale, rappresentarla con canzoni natalizie, sebbene sia ovvio che non esista un *mood* specifico legato alle “festività natalizie”.

Il secondo, di spettro più ampio, è collegato a condizioni meteorologiche pericolose, come “Tempolare”, “Tornado” oppure “Acquazzone”, che possono, secondo un criterio puramente soggettivo, essere rappresentate con canzoni appartenenti al genere ‘Metal estremo’, perché si è pensato che fosse il genere più appropriato per definire una situazione caotica e pericolosa, e molte canzoni appartenenti a questo genere sono davvero simili tra loro.

Per quanto riguarda le altre condizioni atmosferiche, ci si è basati sull'analisi

N#	UMORI	GENERE MUSICALE	TEMPO ATMOSFERICO	COLORE
1	Noia, Frustrazione, Irritazione	Musica Classica	Fumo, Caligine, Sabbia	Magenta
2	Rabbia, Aggressività	Metal	Sereno con molta umidità*	Rosso
3	Stanchezza, Sonnolenza	Smooth Jazz	Pioggia, Nuvoloso , Nebbia, Foschia, Pioggerella	Blu Viola
4	Felicità, Positività	Hit Estive (musica commerciale estiva, Pop, Rock, R&B, HipHop)	Sereno	Giallo
5	Paura	Heavy/Extreme Metal	Temporale, Tornado, Acquazzone, Cenere, Polvere	Verde Scuro
6	Tristezza, Depressione	Lo-Fi	Pioggia, Nuvoloso , Nebbia, Foschia e Pioggerella, con molta umidità*	Blu Scuro
7	Festività	Canzoni Natalizie	Neve	Bianco

Tabella 1.1: Tabella rappresentante il mapping, usato da Physiradio, tra umore, generi musicali, tempo atmosferico e colore.

* umidità relativa $\geq 85\%$

si sopra menzionata[7], poiché le condizioni meteorologiche più comuni sono anche quelle con interpretazione più difficile, di conseguenza avere una base statistica, sebbene piccola, può aiutare.

Il mapping finale testato sul campo è presentato nella tabella 1.1. N# si riferisce al numero di ascolto riprodotto durante gli esperimenti (N stazioni radio differenti).

1.6 IoT

Internet of Things, o IoT, è un termine che è diventato sempre più comune nel linguaggio di tutti i giorni per riferirsi ai nuovi dispositivi e tecnologie. Tuttavia non è semplice trovare una definizione precisa del suo significato; l’ Institute of Electrical and Electronics Engineers (IEEE), utilizza la seguente definizione: “*Un IoT è una rete che connette “cose” univocamente identificabili. Le “cose” hanno delle capacità di percezione/attuazione e potenzialmente di programmazione. Attraverso l’utilizzo di identificazione univoca e percezione, possono essere raccolte informazioni sulla “cosa”, e lo stato della cosa può essere cambiato da ovunque, in qualunque momento, da qualunque cosa.*” [25]

L’IoT dunque coinvolge numerose tecnologie situate a diversi livelli di astrazione: dalle componenti hardware che costituiscono sensori e attuatori degli oggetti collegati in rete alle applicazioni finali su computer e smartphone destinate a utenti e aziende.

1.6.1 La scelta di sfruttare l’IoT come base per la realizzazione di una fisicalizzazione

Una domanda che ci si potrebbe chiedere è: perché usare l’IoT invece di sviluppare altri tipi di fisicalizzazioni? La risposta insita nell’analisi degli *use case* legati al tipo di *data physicalization* che si aveva intenzione di creare. Riflettendo sui casi d’uso di questa specifica fisicalizzazione, si è capito che l’idea migliore fosse quella di creare qualcosa con cui ogni persona, indipendentemente dall’età e dalle competenze, potesse giocare e rappresentare i dati nel comfort della propria casa, spazio di lavoro, ufficio, in macchina, ecc... Una sorta di dispositivo, pensato per una *smart home*, “*blended*” con l’ambiente (vedere Figura 1.6), che, nei futuri sviluppi, potrebbe essere adattato a diverse situazioni. Da questo punto di vista, il prototipo presentato in [14] è stato un utilissimo riferimento.

Inoltre, come già anticipato nel paragrafo sull’analisi dello stato dell’arte (sezione 1.2), tra gli obiettivi, allo scopo di proporre un prototipo che fosse

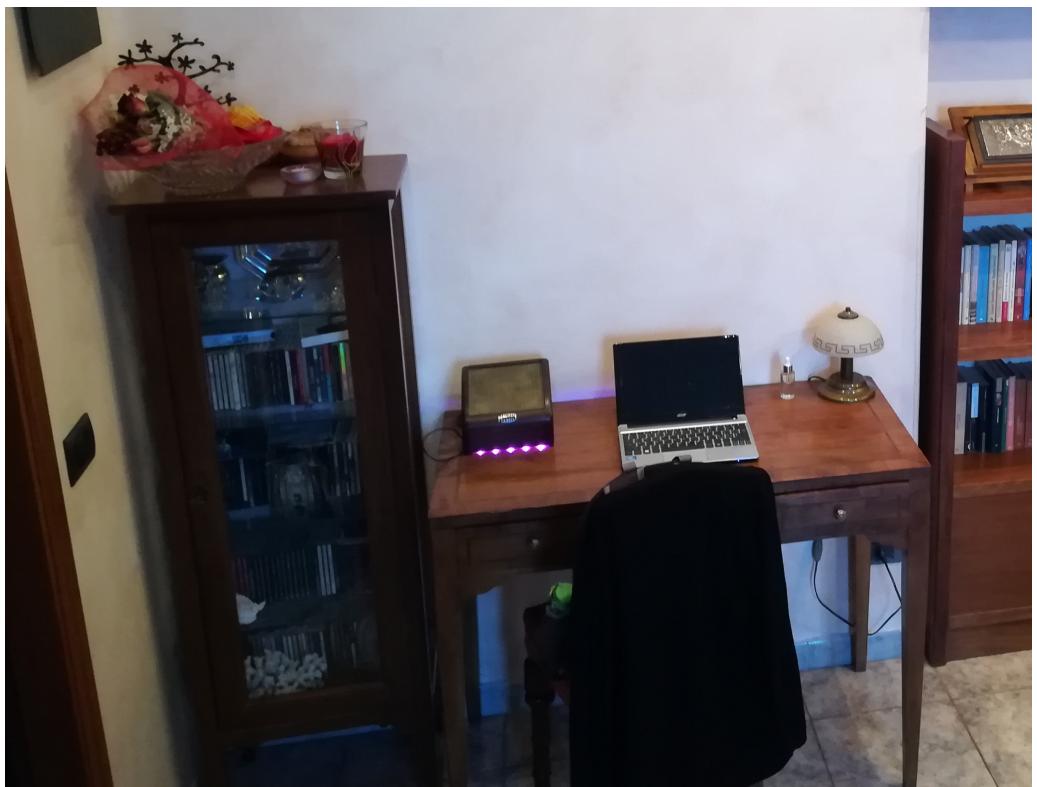


Figura 1.6: Physiradio “*blended*” nell’ambiente, come un comune device IoT pensato per una smart home (suggerimento: cercare i led)

innovativo, vi erano la capacità di creare una *physicalization* che sfruttasse dati di natura dinamica (possibilmente in *real time*) e che fosse interattivo. Perciò un device IoT è risultato estremamente utile anche da questo punto di vista, grazie alla sua capacità di interfacciarsi con la rete.

Capitolo 2

Physiradio: ideazione e realizzazione del device

Una volta capiti quali erano i dati da voler fisicalizzare, con quali criteri e come rappresentare la fisicalizzazione, si è giunti al momento di creare il prototipo. Il device è osservabile nel dettaglio alla Figura 2.1



Figura 2.1: Il prototipo Physiradio visto: all'esterno (sinistra) e all'interno (destra)

Esso è composto dalle seguenti componenti:

- Un cabinato vintage Magneti Marelli (anni '40 ca), con un piccolo speaker montato all'interno
- Una scheda embedded Wemos D1 Mini
- Un codec audio VS1053
- Una striscia led WS2801

Nelle prossime sezioni verranno descritti nel dettaglio i componenti hardware utilizzati, come interagiscono tra di loro, e come i software implementati permettano le interazioni tra le varie parti.

2.1 Componenti Hardware

Il fulcro di Physiradio è la board, una Wemos D1 mini. Ad essa sono connessi tutti gli altri componenti (eccezione fatta per lo speaker). Il funzionamento di tutto il dispositivo segue questa logica: il software che gira sulla scheda embedded riceve i dati riguardanti il tempo atmosferico, li interpreta e, successivamente, vengono *in parallelo*: settati i led, per poter emanare uno specifico colore, e bufferizzati i byte relativi ai flussi audio, per poi essere mandati al codec audio, che li concretizza in segnale audio emettendo musica dallo speaker.

2.1.1 Scheda embedded

“Con la terminologia Sistema Embedded (tradotto testualmente “sistema incorporato”) si tende ad indicare l’insieme composto da hardware e software (occasionalmente definito firmware) dedicato a specifici scopi (“*specific purpose*”) i cui elementi siano tutti quanti integrati ed incorporati.” [2]

Per la creazione del device, si è optato, sin dalle prime fasi, di scegliere una scheda embedded piccola di dimensioni, ma che avesse abbastanza potenza di calcolo e feature che potessero garantire tutte le funzionalità necessarie al corretto funzionamento di Physiradio. Di conseguenza, sia per scelte legate al budget, che per le appena citate motivazioni progettuali, la scelta è caduta su un Wemos D1 mini (Figura 2.2).

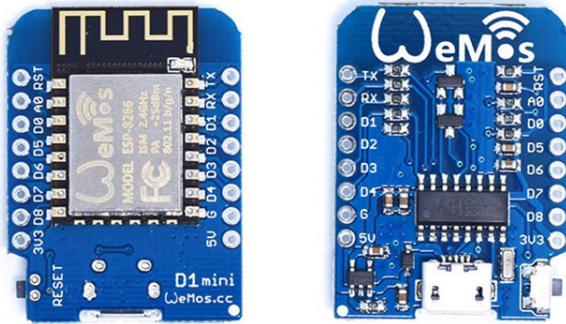


Figura 2.2: Scheda Embedded WeMos D1 mini (V1.0.0), fonte [24]

Essa è una piccola board, sia per peso che per dimensioni (34.2x25.6mm , 3g ca), creata da Wemos.cc [46]. È basato sul chip ESP-8266EX, dotato di un microprocessore Tensilica L106 a 32 bit, che presenta un consumo di energia estremamente basso, la cui velocità di clock della CPU può variare tra gli 80 e i 160 MHz. Una grande dote del Wemos D1 mini, è il modulo ESP-12F: un modulo Wi-Fi 2.4 GHz IEEE 802.11 b/g/n, con supporto WPA/WPA2, che permette la realizzazione di numerosi progetti nell'ambito dell'IoT tramite l'accesso a reti locali. Dispone inoltre di: 11 pin I/O digitali che gestiscono interrupt, pwm, I2C e ISP (tutti funzionanti con una tensione di alimentazione di 3,3 Volt), 1 ingresso analogico, una memoria flash interna di 4MB e un connettore mini USB, da cui viene anche prelevata un'alimentazione di 5 Volt (fondamentale per il funzionamento del codec audio).

Grazie a queste sue caratteristiche, sommate ad un basso prezzo di acquisto, la sua facile accessibilità e alla sua possibile programmazione attraverso l'IDE Arduino[3], questa board è divenuta molto popolare negli ultimi anni tra i “maker”, sia hobbysti che professionisti.

Per implementare Physiradio è risultata la candidata ideale. Si ringrazia Michele Maffucci [24] per i credit dell'immagine e per le specifiche della versione V1.0.0, poiché attualmente la scheda è alla versione v.3.1.0 e sul sito ufficiale vi sono solo le specifiche dell'ultima versione.

2.1.2 Decodificatore audio

Per poter implementare il concetto di *musicalization*, e di conseguenza riprodurre flussi audio per poter ascoltare le varie stazioni webradio, ovviamente è necessario un dispositivo che permetta di decodificare le informazioni ricevute dalla rete (dal Wemos D1 mini) e che, una volta bufferizzate, vengano riprodotte da una sorgente sonora. Per questo motivo, è risultato fondamentale l'utilizzo di un decodificatore audio, che potesse essere di dimensioni contenute e di basso costo, ma che potesse svolgere adeguatamente il lavoro. La scelta perciò si è riversata su un VS1003/1053, in particolare il modello costruito dalla LC Technology, Figura 2.3 .



Figura 2.3: LC Technology VS1003/1053 MP3 Codec audio, fonte [16]

Il modulo di codec audio VS1053 è sia un decodificatore audio MP3/MP3 + V/WMA/WAV/MIDI/SP-MIDI completo, che un codificatore ADPCM, di piccole dimensioni (50x40.5 mm) il quale presenta un DAC stereo (convertitore da digitale ad analogico), un ADC regolabile da 16 bit (convertitore da analogico a digitale), un driver cuffie stereo (con uscita jack da 3.5mm) , e tante altre feature importanti, consultabili direttamente sul sito del produttore [16].

Delle caratteristiche di questa scheda, ci soffermeremo solo su quelle fondamentali per il funzionamento di Physiradio.

L'alimentazione è di 5 Volt, a corrente continua, ed è erogata dal Wemos D1 mini (pin 5v). Utilizzando il driver per cuffie, si può alimentare in maniera

congrua lo speaker, ottenendo un sufficiente volume d’ascolto.

Inoltre il modulo è munito di un’interfaccia SPI (Serial Peripheral Interface), ovvero un sistema di comunicazione seriale tra un microcontrollore e altri circuiti integrati, o tra più microcontrollori, fondamentale per il prototipo.

Nello specifico, la board Wemos funge da master e il VS1053 da slave, in modo tale che dalla board venga emesso il segnale di clock per sincronizzare la comunicazione. Conseguentemente, il traffico viene gestito tramite un buffer di 32 byte (poiché è la capienza massima gestibile dal codec audio) presente in entrambe le schede. Viene riempito prima il buffer del Wemos D1 mini con lo stream acquisito dalla rete, trasferito al buffer del decodificatore audio, che lo interpreta, trasformandolo in un segnale analogico, ossia la musica.

Quest’ultimo passaggio è reso possibile grazie alla libreria creata da Marcin Szałomski [40], senza la quale sarebbe stato molto complesso ottenere una comunicazione tra le due componenti.

2.1.3 Speaker e striscia led

Le ultime componenti hardware da descrivere sono già state introdotte con la presentazione del prototipo (Figura 2.1), ovvero lo speaker e la striscia led. Per quanto riguarda la prima, è uno speaker realmente vintage della Magneti Marelli. Si specifica tuttavia che solamente il cabinet in legno è originale, risalente circa agli anni ’40, poiché lo speaker vero e proprio è moderno, di dimensioni ridotte; con impedenza 4 Ohm e potenza massima iniettabile di 5 W, il che lo rende adatto alle caratteristiche del decodificatore audio descritto nella sezione precedente.

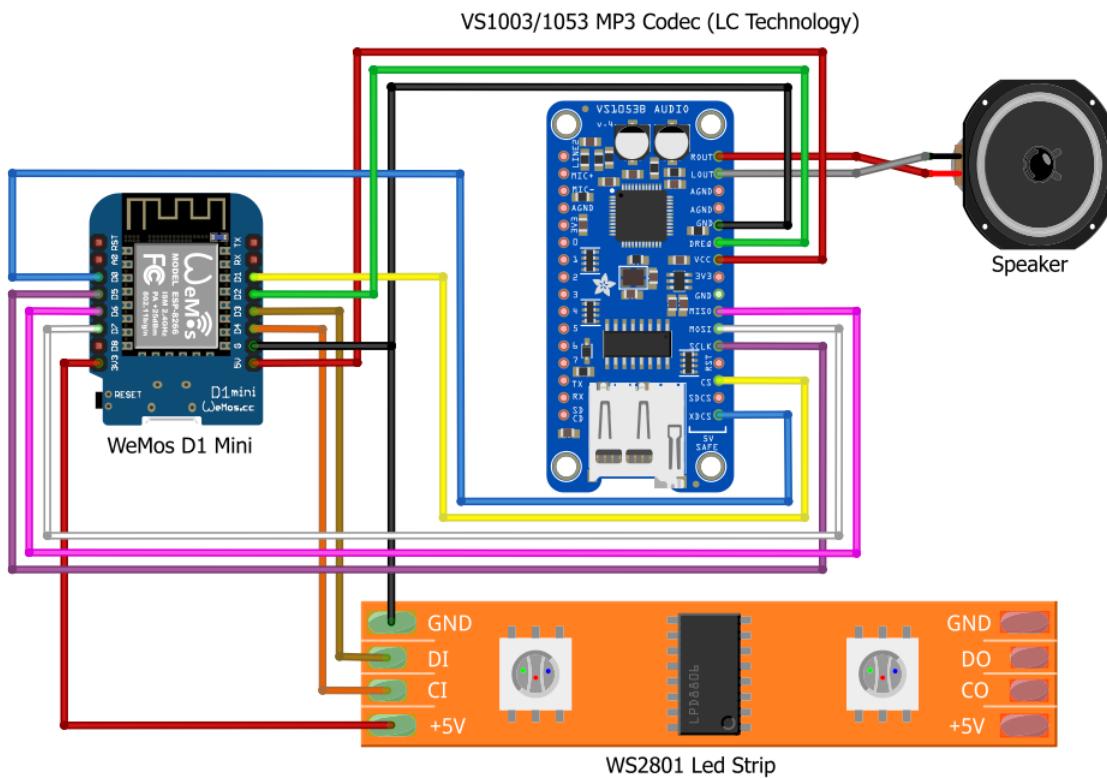
La striscia led invece, è una WS2801, ovvero una led strip digitale a quattro pin: alimentazione (5V), ground (GND), clock (CLK) e data (SD). Per adattarsi comodamente al cabinet in legno, con un *cable management* semplice, sono stati utilizzati solamente cinque led, posizionati nella parte frontale della struttura (sotto il logo ‘Magneti Marelli’) che puntano verso il basso. Questo garantisce una visione sufficiente del colore.

Una nota da specificare è relativa al fatto che, sebbene sia digitale, quindi

che ogni led è controllabile singolarmente via codice (trattandolo come un vero e proprio array), per questa prima versione del prototipo la *led strip* è utilizzata come se fosse una striscia led analogica, senza quindi animazioni o controlli particolari dei led.

2.1.4 Schema cablaggio

Nella Figura 2.4, viene riportato un semplice schema elettronico per dare un’idea di quali siano i collegamenti tra le quattro componenti hardware utilizzate per la creazione del prototipo, che sono state esplicitate in modo più chiaro sulla tabella 2.1. Lo schema è stato realizzato con Fritzing [30], un software libero pensato per la Electronic Design Automation (EDA), ovvero la progettazione di componenti elettroniche.



fritzing

Figura 2.4: Schema Fritzing del cablaggio delle varie componenti hardware all'interno di Physiradio

Occorre però specificare alcuni dettagli implementativi, in quanto si è dovuto ricorrere ad un *cable management* ottimale per poter far funzionare correttamente tutte le componenti e farle coesistere all'interno della struttura di Physiradio :

- È stato necessario saldare appositamente un cavo per poter avere il ground sia per il codec audio che per la striscia led, visto che il Wemos D1 mini possiede un solo pin per il ground (vedere Figura 2.2);
- Come si può notare nella tabella 2.1, si è deciso di alimentare la striscia led con una tensione di 3.3 Volt e non 5 Volt come suggerito,

Wemos D1 mini	VS1053	WS2801
D0	DCS	-
D1	CS	-
D2	DREQ	-
D3	-	SD
D4	-	CLK
D5	SCK	-
D6	MISO	-
D7	MOSI	-
5V	+5V	-
3V3	-	+5V
G	GND	GND

Tabella 2.1: Schema cablaggi interni a Physiradio

semplicemente per comodità pratica; visto e considerato che anche un'alimentazione più scarsa è sufficiente per generare luce da dei led;

- Il canale sinistro e il canale destro dello speaker, sono saldati ad un jack 3.5mm che è inserito direttamente nell'ingresso jack del VS1053, poiché è sufficiente l'amplificatore interno al codec audio per produrre del suono ad un volume accettabile.

Inoltre vanno specificati alcuni dettagli riguardanti lo schema. Il modulo audio non è lo stesso usato nel dispositivo reale: Il codec nello schema è un Adafruit VS1053, poiché purtroppo il modello della LC Technology (Figura 2.3) non esiste per Fritzing, e questo è il più simile trovato in rete. Lo stesso discorso vale per la striscia led, dove nella WS2801, utilizzata nel dispositivo, il pin CI è CLK (clock) e DI è SD (data).

2.2 Software implementati e utilizzati

Dovendo programmare una board come il Wemos D1 mini, la maggior parte dello sviluppo del codice si è incentrata su uno script per Arduino IDE [3], un'applicazione multipiattaforma open source, che offre all'utente un editor di testo per permettere la stesura di codice sorgente. Oltre al supporto di tutte le board Arduino, permette l'installazione di librerie atte a supportare gran parte dei sistemi in commercio, come il NodeMCU ESP8266, l'ESP32, i Wemos e molti altri, la cui scelta si basa sul tipo di utilizzo previsto.

In questi paragrafi verranno trattati quali protocolli e quali librerie sono stati utilizzati per il funzionamento del prototipo. Nello specifico verrà trattato il protocollo MQTT, la libreria multitasking e le funzionalità introdotte per interagire col device.

È utile precisare che, all'interno dello script, viene utilizzata la libreria ArduinoJSON [5], che è stata fondamentale al fine di poter interpretare i file di tipo JSON, ricevuti tramite le API di OpenWeatherMap, per poter ottenere le condizioni atmosferiche della città interrogata.

In un secondo momento è stata sviluppata un'applicazione su Unity[43] per dare un esempio di come si può interagire con Physiradio in contesti più realistici, rispetto allo stato attuale del device che è comunque prototipale.

2.2.1 Il protocollo MQTT

MQTT [39] è l'acronimo di Message Queue Telemetry Transport; le cui versioni v3.1.1 e v5.0 sono standard OASIS dal 2014 (la v3.1.1 è stata anche ratificata dalla ISO).

È un protocollo di messaggistica di pubblicazione/sottoscrizione (*publish/-subscribe*), estremamente semplice e leggero, progettato per dispositivi con risorse limitate e/o poco prestanti, e per reti a bassa larghezza di banda, alta latenza o inaffidabili.

I principi di progettazione sono quelli di minimizzare sia la banda utilizzata della rete che i requisiti delle risorse del dispositivo, cercando allo stesso tempo di garantire affidabilità e un certo grado di garanzia della consegna

dei messaggi. Questi principi, si sono rivelati col tempo ideali per rendere il protocollo molto utile e sfruttato dal mondo emergente dei dispositivi interconnessi “*Machine-to-Machine*” (M2M) e del “*Internet of Things*” (IoT), e per le applicazioni mobile, in cui la larghezza di banda e la durata della batteria sono fondamentali.

Prendendo come descrizione di riferimento [2]: Il protocollo si appoggia principalmente al protocollo TCP/IP, o in alternativa ad altri protocolli di rete che forniscono connessioni ordinate, senza perdita di dati e bidirezionali. Esso si basa sul pattern *publish-subscribe*¹: dei messaggi informativi vengono inviati da una sorgente (un nodo “client” della rete MQTT) verso un *dispatching broker* che li ridistribuisce a tutti i nodi che si erano dichiarati interessati a quel tipo (topic) di messaggio.

Ogni messaggio MQTT è composto da:

- un *topic*, che servirà al broker per la distribuzione: è una stringa il cui contenuto è libero, ma viene specificata la forma sintattica da utilizzare per costruirlo (parole separate dal carattere “/”, vedere documento di markdown nella sezione 2.2.3 per degli esempi);
- un *payload*, il corpo del messaggio: è semplicemente una stringa UTF-8, la cui semantica è affidata alle convenzioni stabilite tra chi manda e chi riceve, non è definita nel protocollo.

Il broker gestisce una lista di nodi client interessati ai vari topic, ogni messaggio che riceve viene inviato solo ai nodi che si erano “iscritti” a quel particolare topic. I nodi che si “iscrivono” (*subscribe*) possono indicare come “interesse” anche una *wildcard*, una stringa contenente caratteri speciali che rappresenta un insieme di stringhe effettive (si veda la funzione `reconnect()` nel Codice 1.1, Appendice A, per avere un esempio), invece di un singolo topic per esteso.

È perciò un modello che fornisce la distribuzione dei messaggi uno-a-molti e il disaccoppiamento delle applicazioni.

Un’altra caratteristica specifica del protocollo MQTT, sono i livelli di Quality of Service (QoS), letteralmente qualità di servizio. È un accordo

¹Inteso come Design Pattern, sul modello Observer-Observable

tra il mittente e il destinatario del messaggio, che ne definisce la garanzia di consegna. I livelli QoS sono una funzionalità chiave del protocollo MQTT, perché offre al client la possibilità di scegliere un livello di servizio che corrisponda all'affidabilità della rete e alla logica dell'applicazione. Poiché MQTT gestisce la ritrasmissione dei messaggi e garantisce la consegna (anche quando il trasporto sottostante non è affidabile), i livelli QoS rendono molto più semplice la comunicazione con reti inaffidabili.

I QoS sono 3 livelli, e sono:

- “*At most once*” - QoS 0, in cui i messaggi vengono recapitati in base ai migliori sforzi dell’ambiente operativo. La perdita del messaggio può verificarsi, ma è il metodo più veloce per inviarlo.
- “*At least once*” - QoS 1, in cui è garantito l’arrivo dei messaggi tramite un ACK specifico (PUBACK) inviato dal broker. Possono verificarsi duplicati.
- “*Exactly once*” - QoS 2, dove è garantito che il messaggio arrivi esattamente una volta, tramite un four-part handshake tra client e broker. È ovviamente il metodo più lento, ma anche il più sicuro.

Tutti i messaggi inviati con QoS 1 e 2 vengono messi in coda per i client offline, fino a quando il client non sarà nuovamente disponibile. Tuttavia, questo è possibile solo se il client ha una sessione persistente.

È necessario specificare che, in tutti i software sviluppati per questo progetto, si è sempre utilizzato il livello Quality of Service 0. La ragione principale di questa scelta, è legata al fatto che le componenti utilizzate per creare il device hanno risorse molto limitate e poco prestanti, specialmente la scheda embedded, che è di fatto il nucleo del funzionamento del prototipo. Di conseguenza, con un QoS 0, possiamo comunque ottenere un sistema rapido ed efficiente per quello che è il suo scopo; senza però avere meccanismi di sicurezza e di check del messaggio inviato. Viene dunque tenuto in considerazione, che un messaggio potrebbe non essere recapitato a Physiradio, il che non è problema considerando il tipo di dato che viene trasferito sulla rete:

privo di dati sensibili e che se, anche sniffato e/o alterato, non causerebbe rischi eccessivi a livello di sicurezza.

Nelle varie implementazioni di MQTT (è disponibile per molti linguaggi, sistemi operativi e su piattaforme sia embedded che standard) le funzioni principali le due:

- `Subscribe(String subTopic, Function callback)`: permette (una volta connessi ad un broker) la dichiarazione di interesse verso uno o più topic, quando il broker riceve un messaggio il cui topic corrisponde con subTopic lo invia al client che lo riceve e scatena l'invocazione della funzione callback che va implementata a cura del programmatore del nodo “*subscriber*”;
- `Publish(String topic, String message)`: permette (una volta connessi ad un broker) l'invio di un messaggio associato ad un topic, il broker analizzerà il topic e inoltrerà il messaggio ai nodi il cui subTopic corrisponde.

In questo progetto, il protocollo è stato implementato grazie alla libreria (per Arduino IDE) PubSubClient [26], creata da Nick O’Leary. Il codice implementato (Codice 1.1) è visionabile all’Appendice A. Nello specifico si può notare che sono state utilizzate le seguenti funzioni:

- `setServer (server, port)` : indirizzo e porta del server a cui connettersi;
- `setCallback (callback)` : un puntatore a una funzione di callback del messaggio, chiamata quando arriva un messaggio per la sottoscrizione a uno(o più) topic creato dal client. La funzione di callback ha questa forma:

```
void callback(const char[] topic, byte* payload, unsigned int length)
```

e i parametri sono:

`topic const char []` - il topic sul quale è arrivato il messaggio;

```
payload byte[] - il payload del messaggio;  
length unsigned int - la lunghezza del payload.
```

- **subscribe (topic, [qos])** : sottoscrive il client ai messaggi pubblicati dal server su uno specifico topic. Si può decidere opzionalmente con quale livello di QoS sottoscriversi al server;
- **connected ()**: verifica se il client è connesso ad un server;
- **connect (clientID, [username, password], [willTopic, willQoS, willRetain, willMessage], [cleanSession])**: cerca di connettersi al server settato precedentemente (con la funzione setServer). Si può opzionalmente conferire username e password, se il server li richiede per istanziare la connessione. Inoltre si possono settare i parametri legati al Last Will and Testament (LWT), per informare gli altri *subscriber* dello stesso broker che il client si è disconnesso (di solito in modo anomalo). Per ultimo, si può settare opzionalmente se il client richiede una connessione persistente(non-clean) oppure no.

Occorre specificare che non è mai stata usata alcuna funzione di *publish*, poiché Physiradio, allo stato attuale di sviluppo, è pensato solo per ricevere, analizzare e interpretare i messaggi che vengono inviati dai client *publisher*.

2.2.2 Libreria TaskScheduler

Un altro punto fondamentale del funzionamento di Physiradio è la sua implementazione del *multitasking*.

Quando si parla di sistemi embedded, i cui software sono stati sviluppati su Arduino IDE, vi sono numerose librerie per gestire, in modo elegante, più flussi paralleli all'interno del programma utente; nella logica di pulizia del codice, senza di esse, spesso si ricorre a tecniche naïf².

Durante le prime fasi di sviluppo del device, è subito sorto il problema di dover gestire le tre principali funzionalità del software:

- Flusso continuo dello streaming audio, reindirizzato al codec audio;

²In gergo si parla di “gestire a mano a colpi di millis” i vari task

- Interrogazione del sistema OpenWeatherMap, per avere il dato da mappare in *real-time*;
- Ricezione di messaggi MQTT che arrivano dal broker.

E che, di conseguenza, fossero gestite in modo ordinato e il più deterministico possibile, tramite delle politiche di scheduling.

A tal proposito, si è deciso di sfruttare una libreria per Arduino IDE, sviluppata da Anatoli Arkhipenko, chiamata TaskScheduler [4].

È un’implementazione, di dimensioni contenute, di un *cooperative multitasking*, chiamato anche *multitasking non-preemptive*. Consiste sostanzialmente in una modalità di *multitasking*, in cui il sistema operativo (nel nostro caso, con un ESP8266 compilato con le SDK Arduino, parliamo solamente di firmware [47]) non avvia mai un *context switch* da un processo in esecuzione ad un altro processo. Sono invece i processi stessi che volontariamente lasciano il controllo: periodicamente, o quando entrano in idle o quando vengono esplicitamente bloccati da qualche logica del programma. Questo tipo di *multitasking* è chiamato “cooperativo” perché tutti i processi devono cooperare per l’intero *workflow* di scheduling (Figura 2.5) al fine di garantire l’esecuzione dell’intero programma.

Nel caso della libreria TaskScheduler, avremo un unico programma utente che gira all’interno della CPU, nel quale potremo creare dei processi (cosa che nativamente non è possibile fare su Arduino Ide), che per lo specifico si chiamano *task*, e saranno tutti aggiunti ad una lista di task eseguibili all’interno di un oggetto Scheduler (vedere il Codice 1.2, Appendice B, per avere un esempio). Di default, i *task*, vengono eseguiti ciclicamente a periodicità definite nel *setup*, ma possono essere definiti molteplici parametri che ne possono variare notevolmente il ciclo di esecuzione.

In particolare, la libreria è in grado di garantire: l’esecuzione periodica dei *task* espressi in millisecondi o microsecondi (se esplicitamente abilitato), il numero esatto delle iterazioni dei singoli *task*, la modifica dinamica dei parametri di esecuzione (frequenza, numero di iterazioni, metodi di callback) e molte altre feature (vedere la documentazione ufficiale per info più specifiche [4]).

Nel dettaglio, per l’implementazione di Physiradio è stata sfruttata prin-

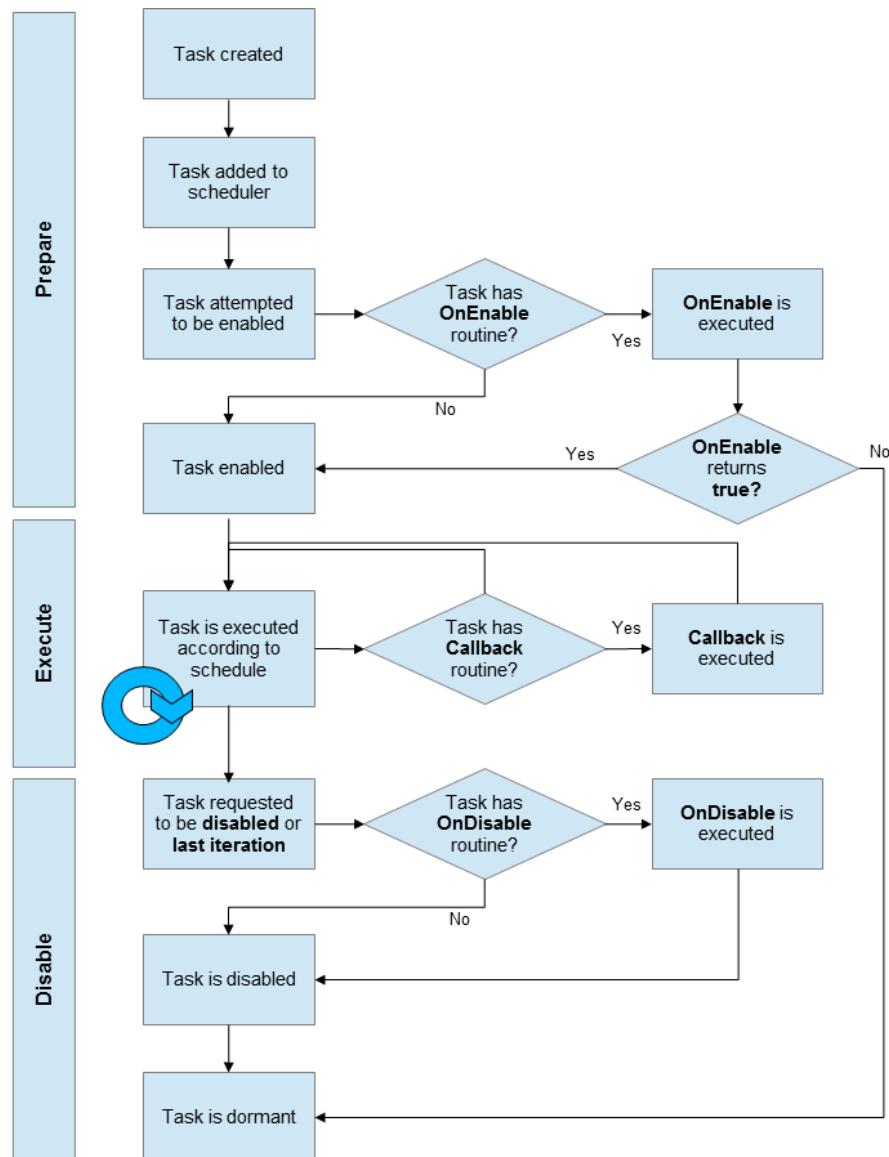


Figura 2.5: Workflow di TaskScheduler, fonte [4]

cipalmente la prima feature sopracitata, ovvero quella di definire l'esecuzione periodica dei singoli task, lasciando l'ordine di esecuzione con i parametri di default. È stata definita tramite un costruttore della classe Task:

```
Task(unsigned long aInterval, long aIterations, void (*aCallback)(),  
Scheduler* aScheduler, bool aEnable,  
bool (*aOnEnable)(), void (*aOnDisable)());
```

definendo *aInterval*, l'intervallo di tempo ogni quanto eseguire il task, *aIterations*, per quante volte eseguirlo, e assegnando **aCallback*, la funzione di callback da eseguire. Gli altri parametri, opzionali, non sono stati settati all'interno del costruttore, ma fuori da esso, per semplice comodità di debugging.

La sezione di codice sviluppata usufruendo della libreria TaskScheduler è visionabile all'Appendice B: da notare che, le tre principali funzionalità di Physiradio sopracitate, sono state tradotte in tre task specifici.

2.2.3 Descrizione dell'interazione con il device

Ci sono principalmente due modi per interagire con Physiradio: il primo metodo è quello di utilizzare l'app Physiradio MQTT Interface, che verrà descritta nel dettaglio successivamente (sezione 2.2.4). Il secondo metodo consiste nell'utilizzare un qualsiasi device capace di eseguire il comando (del protocollo MQTT) *publish*, interagendo con, o tramite, un broker MQTT.

In particolare, di seguito viene riportato integralmente il documento di mark-down (tradotto in italiano), disponibile sulla repository dedicata unicamente ai software di Physiradio [37], dove vengono riportate le istruzioni, e degli esempi, per interagire con il device, nei quali viene utilizzato Mosquitto [11] su un normale terminale Linux.

Topic e payload per interagire con PhysiRadio

ATTENZIONE: Tutte le chiamate MQTT devono essere eseguite da un dispositivo capace di istanziare almeno un comando di MQTT *publish* (per esempio un Mosquitto client o simili)

Forma generica di istruzione:

```
mosquitto_pub -h server_name -t PhysiRadio/topic -m "payload"
#nota: il "payload" deve essere scritto tra virgolette
```

Comandi eseguibili, divisi per topic:

Volume

- Volume +1
- Volume -1
- Volume mute
- Volume unmute

Esempi:

```
mosquitto_pub -h server_name -t PhysiRadio/Volume -m "+1"
mosquitto_pub -h server_name -t PhysiRadio/Volume -m "mute"
```

City (nota bene: la città deve esistere sulla piattaforma OpenWeatherMap, e il nome deve essere scritto possibilmente in lingua inglese)

- City “nome_di_una_città” (non è case sensitive, grazie alle API di OpenWeatherMap)
- City “nome_di_una_città, PAESE” (per quelle città che hanno lo stesso nome, ma che sono in paesi differenti)

Esempi:

```
mosquitto_pub -h server_name -t PhysiRadio/City -m "London"
mosquitto_pub -h server_name -t PhysiRadio/City -m "New York"
mosquitto_pub -h server_name -t PhysiRadio/City -m "Venice, IT"
#(Venice si trova anche negli USA)
```

Station (per motivi legati al testing, è stato utile forzare il cambio di stazione web radio)

- Station enable (Abilita il task per le API di OpenWeatherMap)
- Station disable (Disabilita il task per le API di OpenWeatherMap, di default è abilitato)
- Station station1 (Classical → Smoke/Mist)
- Station station2 (Metal → Clear(sun) + High humidity)
- Station station3 (SmoothJazz → Rain/Drizzle/Cloud/Fog + low humidity)
- Station station4 (SummerHits → Sunny + low humidity)
- Station station5 (ExtremeMetal → Thunderstorm/Tornado...)
- Station station6 (LoFi → Rain/Drizzle/Cloud/Fog + High humidity)
- Station station7 (Xmas Songs → Snow)

Esempi:

```
mosquitto_pub -h server_name -t PhysiRadio/Station -m "station3"
mosquitto_pub -h server_name -t PhysiRadio/Station -m "enable"
```

Possibili futuri sviluppi

Topic con distinzione del client ricevente

Allo stato attuale dello sviluppo, l'interazione tra i client “*publisher*” (device che usano broker di messaggi, come Mosquitto [11], o che usano Physiradio MQTT Interface) e Physiradio (o più di uno) è completamente broadcast: qualsiasi Physiradio creato(o creabile) è “iscritto” (inteso come *subscriber* MQTT) a tutti i topic sotto “Physiradio/#”, e tutti i messaggi pubblicati, dai device “*publisher*” prima citati, sotto quei topic, sono inviati attraverso il broker a tutti i Physiradio (attivi in quel momento) simultaneamente e senza nessun tipo di filtro.

Un'implementazione possibile per risolvere parzialmente questa situazione, potrebbe essere quella di utilizzare il meccanismo delle liste di controllo accessi(ACL). Questo permetterebbe di selezionare quale end-device (ovvero quale/i Physiradio) deve ricevere determinati messaggi, quindi non per forza tutti, aggiungendo uno (o più) livelli di topic, creando una divisione per classi.

Ad esempio, potremmo avere molteplici Physiradio situati in varie stanze della casa. Di conseguenza potremmo creare i topic in questo modo:

- Physiradio/allrooms/... : per mandare il messaggio a tutti i Physiradio (che saranno tutti sottoscritti a questo topic)
- Physiradio/room1/... : per mandare il messaggio solamente ai Physiradio nella room1
- Physiradio/room2/... : per mandare il messaggio solamente ai Physiradio nella room2

Altre implementazioni

- data2physical <json> : settare un mapping customizzato (tramite json)
- <data> <urlstream> : settare un stream audio diverso da quello mappato attualmente

Sottoscrizione a parametri, da parte dei device “*publisher*” a

- Volume attuale...
 - Dato fisicalizzato...
 - Url corrente...
 - Utilizzo della memoria
 - ecc...
-

Occorre sottolineare che, per quanto concerne le sezione sopracitata della selezione del topic con distinzione del client ricevente: è sicuramente un’implementazione futura che potrebbe rendere Physiradio un device IoT ancora più adatto ad uno scenario reale, conferendo anche un margine di customizzazione da parte dell’utente che ne usufruisce.

Tuttavia, nulla vieta di utilizzare il prototipo così com’è, avendo più device IoT che trasmettono lo stesso stream di una specifica webradio ed emettono tutti lo stesso colore dalla striscia led; tutti controllati da remoto da molteplici sorgenti, in concorrenza.

2.2.4 Physiradio MQTT Interface: applicazione per interagire col device

Come anticipato in precedenza, un metodo per interagire con Physiradio è utilizzare l’app Physiradio MQTT Interface.

Il suo sviluppo, è dovuto alla volontà di dare un esempio di una user experience il più realistica possibile, e far accrescere ancora di più la curiosità e l’interesse nell’utente fruitore verso questo dispositivo.

Essa è stata programmata in C#, sull’ambiente Unity [43], un motore grafico (sia 2D che 3D) multipiattaforma sviluppato da Unity Technologies, che negli ultimi anni è divenuto molto popolare nel mondo del *game development* per la sua semplicità e flessibilità di utilizzo. Tuttavia, proprio per queste sue caratteristiche, il motore si presta molto bene non solo per lo sviluppo di videogiochi, ma anche per altri contenuti interattivi, quali visualizzazioni architettoniche, animazioni 3D in tempo reale o lo sviluppo app general purpose. Quest’ultimo esempio è il caso di Physiradio MQTT Interface.

Uno dei motivi principali per cui è stato scelto Unity come ambiente di sviluppo, è la possibilità di poter esportare qualsiasi progetto (esclusi quelli per VR) in formati eseguibili per la maggior parte dei device sul mercato, siano essi desktop o mobile, senza dover fare alcuna modifica lato codice (se scritto nel modo corretto). Di conseguenza, l’app sviluppata per questo progetto è disponibile, sulla repository dedicata ai software implementati [37], per Windows, Linux, MacOS e Android.

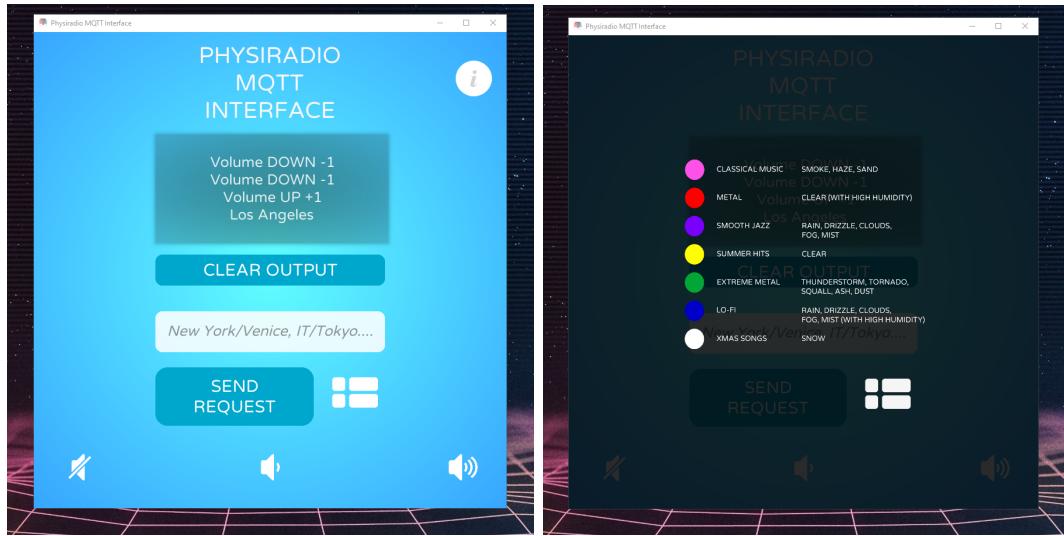
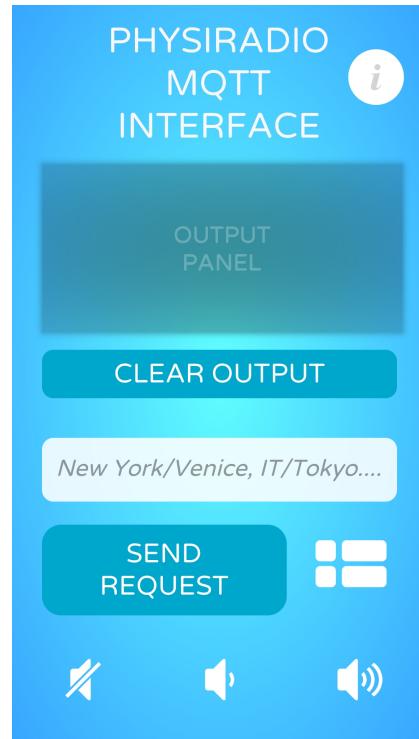


Figura 2.6: Physiradio MQTT Interface su Android(sotto) e Windows 10 (sopra)



L'applicazione (Figura 2.6) consiste di una singola schermata, da cui è possibile inviare messaggi MQTT a Physiradio, dello stesso formato descritto nel paragrafo precedente 2.2.3, e non solo. In particolare è possibile:

- Controllare il volume : mute/unmute, volume -1 e volume +1;
- Inviare una città da interrogare tramite le API di OpenWeatherMap: si compila il riquadro centrale bianco e si preme il bottone ‘SEND REQUEST’;
- Visualizzare gli ultimi quattro comandi inviati al device IoT (utilizzando sempre QoS 0, di conseguenza non avremo mai la certezza dell’arrivo del messaggio);
- Visualizzare il riquadro delle informazioni (pulsante rotondo con la ‘i’);
- Visualizzare una legenda sul mapping utilizzato (descritto nella sezione 1.5) con colore visualizzabile della striscia led, genere musicale e tempo atmosferico. Visibile nel riquadro in alto a destra della Figura 2.6.

Sull’ultimo punto descritto, è necessario specificare che è una feature implementata al fine di aumentare la facilità di comprensione del device e del dato che in quel momento viene fisicalizzato, poiché, conferendo una legenda, l’utente non è più obbligato a interpretare la musica e i colori che vede, in funzione del dato che potrebbe rappresentare, ma gli basterà usufruire del device e, nel dubbio interpretativo, interrogare la legenda. Purtroppo, per limiti temporali e logistici, l’utilizzo dell’applicazione non è stata testata sugli utenti che si sono offerti per il *field testing* del device.

Una nota importante da fare, è relativa alla libreria utilizzata per implementare il protocollo MQTT. Si tratta della libreria M2Mqtt[9] sviluppata da Ian Craggs, la quale permette di istanziare un client MQTT, il quale può connettersi ad un broker ed eseguire comandi sia di *publish* che di *subscribe*. Nel nostro caso, l’applicazione si limita solamente a pubblicare, poiché Physiradio non pubblica messaggi, perciò non vi è la necessità di sottoscrivere l’app a nessun topic.

Capitolo 3

Field testing e analisi feedback

Finita la preparazione del prototipo, lo step finale è stato quello di testarne l'esperienza di fruizione su un campione di persone, 59 per l'esattezza, al fine di raccogliere dati sull'esperienza provata nell'utilizzare Physiradio, e per capire se ha svolto il compito per il quale è stato creato. Ossia quello di poter far concepire il dato, proveniente da una sorgente *open* (in particolare, il tempo atmosferico), tramite una fisicalizzazione che sfrutta il senso dell'udito, stimolato dalle canzoni, e la vista, tramite i colori dei led; e se gli utenti ne sono stati incuriositi.

3.0.1 Somministrazione con questionario

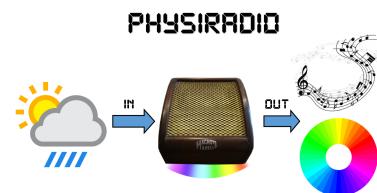
Nel dettaglio, per testare questa idea di musicalizzazione, è stato creato un questionario anonimo. Sono state organizzate numerose sessioni di ascolto in cui studenti universitari, colleghi, amici, membri di hacklab locali e alcuni parenti, sono stati sottoposti alla fisicalizzazione di Physiradio, dalla quale hanno potuto suggerire molti spunti e conferire feedback importanti.

Di seguito viene riportato il questionario per intero:

Physiradio è un dispositivo IoT (*Internet of Things*) dimostrativo: recupera dalla rete i dati relativi alle condizioni meteo di un luogo scelto in fase di configurazione, li elabora e li rappresenta attraverso una combinazione di musica (genere musicale) e colore. Realizza cioè una possibile “tangibilizzazione” (liberamente tradotto dall’inglese *physicalization*) di un dato.

Lo scopo di questa demo è raccogliere impressioni su:

- l’idea di rappresentazione del dato atmosferico mediante musica e colori
- l’efficacia dello strumento nel sollevare curiosità sulla tecnica di estrazione del dato stesso



Nota bene: non cerchiamo l’associazione perfetta, ci interessa valutare il procedimento di rappresentazione.

Ti chiediamo di compilare il questionario (anonimo) che segue, man mano che procediamo con la dimostrazione, per fornirci un *feedback* sul lavoro che stiamo realizzando.

1. Anagrafica (anonima):

- Età [.....] - Sesso F[] M[] - Città di residenza [.....]
- Occupazione [.....]
- Titolo di studio: Obbligo[] Superiori[] Laurea[] PhD[]
- Ascoltatore abituale di musica: Sì[] No[]
- Se “Sì”:
 - (a) Su quale *device*?
 - Radio[] Webradio[] CD[] Vinile[] MP3 player (o smartphone offline)[]
 - App streaming[]
 - (b) In che contesto?
 - Casa[] Lavoro[] Auto[] Mezzi pubblici[]
 - Bagno[] Letto[] Allenamento[] Passeggiando[]
 - Altro[.....]

(c) Quali generi musicali ascolti?

[.....]
[.....]

2. Ora ti faremo “ascoltare e vedere” alcune condizioni meteo senza dirti a cosa corrispondono, ti chiediamo per ogni ascolto di inserire uno a o più X in corrispondenza delle condizioni meteo che meglio lo rappresentano.

Ascolto	Pioggia	Pioggerella	Sereno/Soleggiato	Nuvoloso	Temporale	Uragano/Nubifragio	Tornado	Neve	Fumo	Nebbia	Foschia	Caligine	Sabbia	Cenere	Polvere	Umidità (Rel.)\geq85%
Ascolto 1																
Ascolto 2																
Ascolto 3																
Ascolto 4																
Ascolto 5																
Ascolto 6																
Ascolto 7																

3. Quale genere musicale e/o colore associeresti alle seguenti condizioni meteo? (puoi anche compilarla parzialmente: inserisci almeno le associazioni che ti sembrano più calzanti; per aiutarti, puoi anche partire da un genere musicale e associare una condizione meteo)

- Nebbia [.....]
- Pioggia [.....]
- Pioggerella [.....]
- Sereno/Soleggiato [.....]
- Nuvoloso [.....]
- Temporale [.....]

- Uragano/Nubifragio [.....]
 - Neve [.....]
4. Ritieni efficace/utile la rappresentazione di questo tipo di dato tramite genere musicale e colore?
 Poco Abbastanza Molto Moltissimo
5. Ci suggeriresti un altro tipo di dato interessante da tangibilizzare con questo strumento?
[.....]
[.....]

6. Open Data

I dati meteo usati in questo prototipo sono recuperati in tempo reale via rete da fonti cosiddette “open data”, cioè fonti che forniscono informazioni liberamente utilizzabili tramite programmi (quindi non solo attraverso pagine web). Dentro Physiradio gira infatti un programma che recupera dati dalla rete e sceglie uno *stream* musicale e un colore in base al *mapping* che hai appena sperimentato.

Lo scopo primario del processo di tangibilizzazione è quello di rendere fruibile i dati attraverso vari sensi, ma nel caso particolare di Physiradio ci interessa anche stimolare la curiosità sugli aspetti tecnologici del mondo dei dati aperti e della trasparenza, cioè:

- Come vengono resi disponibili i dati e come si possono recuperare?
- Che forma hanno questi dati e che tipo di elaborazioni/visualizzazioni sono possibili?

Ti abbiamo incuriosito? Poco Abbastanza Molto Moltissimo

Commenti liberi:

[.....]
[.....]
[.....]
[.....]
[.....]

GRAZIE!

3.0.2 Analisi dei feedback

Tutti i questionari sono stati analizzati singolarmente e sono stati registrati in un dataset. Esso, disponibile tramite Zenodo [36], è stato elaborato tramite un notebook Jupyter[19] sul quale è stato creato uno script Python 3 che sfrutta principalmente la libreria Pandas[28] per eseguire l'analisi dei dati.

Di seguito sono riportate le statistiche più rilevanti estratte dal dataset:

- Proporzione Uomini/Donne: 78/22%;
- Età media: 30.5 anni (dev. std: 15.4);
- ‘Studente’ è la classe più rappresentata con 64%, seguito da ‘Impiegato’ 21%. ‘Pensionato’, ‘Ricercatore’, ‘Libero professionista’ e altre classi non superano il 3%;
- Il 34% vive a Milano, il resto del campione vive prevalentemente nella provincia di Milano, tutti in Lombardia tranne qualche sporadica eccezione che abita in Piemonte;
- Titoli di studio: Diplomati 67.8%, Laureati 22.1%, PhD 8.4%, Scuola dell’obbligo 1.7% ;
- Il 96.6% del campione è un ascoltatore abituale di musica, dei quali:
 - Contesto di ascolto: ‘Casa’ 22%, ‘Auto’ 18%, ‘Mezzi pubblici’ 18%, ‘Passeggiando’ 12%, ‘Allenamenti’ 8.3%, ‘Letto’ 8%, ‘Bagno’ 6.5% , ‘Lavoro’ 4.6%, gli altri contesti non superano il 4%.
 - Dispositivo di ascolto: ‘App streaming musicale’ 33.3%, ‘Lettore MP3’ 23.9%, ‘Radio’ 18.8%, ‘CD’ 12.8%, ‘Vinile’ 6.8% e ‘Webradio’ 4.2%.

La Figura 3.1 mostra i grafici a torta relativi alle risposte date dagli utenti sottoposti al questionario riguardanti l’efficacia soggettiva, rispetto al prototipo presentato, e a quanto sono stati incuriositi dall’oggetto stesso.

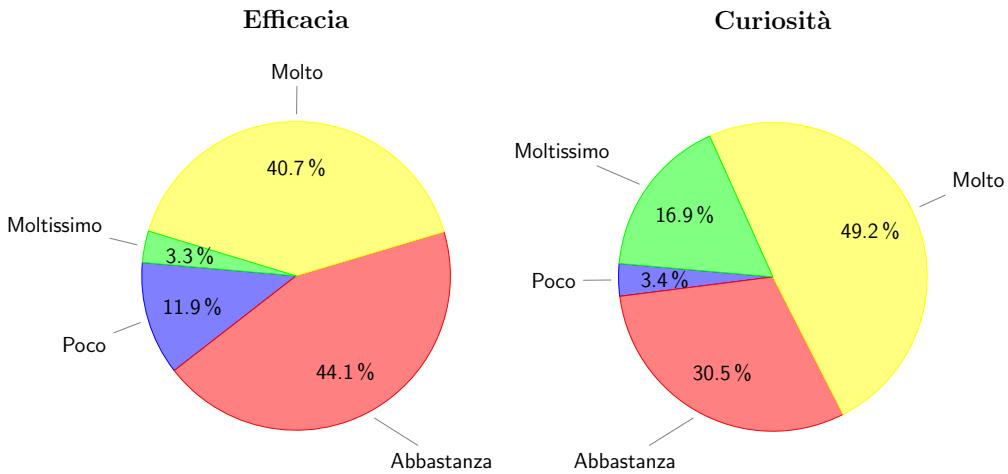


Figura 3.1: Risposte date riguardanti l’efficacia (sinistra) dell’attuale prototipo e la curiosità instillata (destra). In basso invece si possono analizzare le statistiche relative ai due dati sopra citati (per poter generare una statistica sulla valutazione (a risposta chiusa) di ogni utente, è stato eseguito un mapping, dell’intervallo Poco→Moltissimo, con numeri interi da 1 a 4)

	Poco (1)	Abbastanza (2)	Molto (3)	Moltissimo (4)	Media	Dev.Std
Efficacia	11.9%	44.1%	40.7%	3.3%	2.35	0.73
Incuriosito	3.4%	30.5%	49.5%	16.9%	2.79	0.76

L’ indice di correlazione tra *Incuriosito* e l’*Efficacia* è di 0.47 (con covarianza 0.26), tra *Età* e *Incuriosito* è di -0.0039 (con covarianza -0.46), tra *Età* e *Efficacia* è di 0.012 (con covarianza -0.14).

Possiamo notare che la correlazione tra fattore di curiosità accresciuta e l’efficacia del dispositivo è di circa 0.5. Questo ci suggerisce che i due valori sono abbastanza legati fra di loro da una leggera relazione lineare: una buona parte di chi è stato convinto dalla funzionalità dell’oggetto è stato anche

genuinamente incuriosito da come funzionasse.

Inoltre, dalle statistiche estratte, si evince che, il fatto di ritenere il device efficace e di esserne stati incuriositi, non ha pressoché alcuna rilevanza rispetto all'età degli utenti. Da questo punto di vista il campione è omogeneo.

Oltre a chiedere l'opinione degli utenti, l'efficacia della mappatura è stata anche misurata verificando le corrispondenze tra l'ascolto e le condizioni meteorologiche associate. La Figura 3.2 elenca e traccia le frequenze corrispondenti¹ per ogni ascolto riprodotto. I valori mostrano che gli ascolti 4, 5, 6 e 7 sono stati ‘indovinati’ correttamente molto spesso, mentre gli ascolti 1, 2 e 3 sono stati associati correttamente di rado.

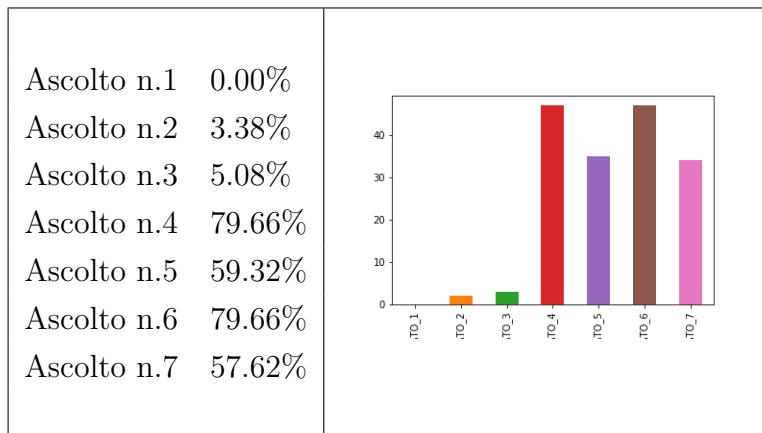


Figura 3.2: Conteggio delle corrispondenze trovate tra il valore dato dagli utenti per ogni ascolto e l'effettivo valore associato dal mapping (ricordiamo che il campione è di 59 utenti)

Questi dati in realtà non sorprendono molto, da come si poteva già notare nella tabella 1.1, i tempi atmosferici legati agli ascolti n.1, 2, e 3, sono quelli di cui ci si aspettava una più difficile interpretazione. In primo luogo, i generi musicali assegnati sono, purtroppo, molto ampi come spettro di sonorità.

¹I valori vanno da 0% (nessuno ha ‘indovinato’) a 100% (tutti hanno ‘indovinato’).

Secondariamente, quegli specifici tempi atmosferici erano i più complessi da rappresentare. Analizzando i due paper di riferimento [20] e [7] si nota innanzitutto che le condizioni ‘Fumo’, ‘Caligine’ e ‘Sabbia’ non sono neanche stati citati, mentre potrebbero comunque essere ricevuti da OpenWeatherMap, di conseguenza la scelta del genere è stata complessa e, purtroppo, fallimentare. Successivamente, per quanto riguarda l’ascolto n.2, il tempo atmosferico ‘Sereno con molta umidità’, assimilabile come afa, non è quasi mai stato preso in considerazione dal campione di utenti, forse perché poco comune e non molto intuitivo rispetto al semplice ‘Sereno’. Lo stesso principio vale per l’ascolto n.3, ma al contrario; probabilmente l’utente medio percepisce i tempi come ‘Pioggia’, ‘Pioggerella’, ‘Foschia’ ecc... direttamente con un alto tasso di umidità invece che considerare anche il caso in cui sia bassa.

Capitolo 4

Conclusioni e futuri sviluppi

I risultati mostrano che Physiradio è ben accettato dall’utenza che ne ha usufruito (letteralmente “che bell’oggetto!” è stato affermato frequentemente) e che stimola efficacemente la curiosità sui funzionamenti interni e sui dati aperti: il 66.1% (Figura 3.1) dei soggetti ha indicato *Molto* o *Moltissimo* sulla sezione riguardo l’ “aumento della curiosità”, e non vi è correlazione per età a riguardo.

Coloro che lo hanno trovato interessante, hanno anche assegnato un valore “relativamente alto”¹ per l’efficacia (correlazione: 0,47).

Il 44% dei soggetti ha dichiarato che il prototipo è molto efficace e che trova coerente la mappatura tra condizioni meteorologiche e la coppia genere musicale-colore. Questa valutazione è stata confermata analizzando le corrispondenze effettive (Figura 3.2) tra il mapping utilizzato e il mapping ipotizzato (musica e colore → tempo atmosferico) dagli utenti. In quattro ascolti (ovvero 4, 5, 6, 7) su sette, le corrispondenze sono state spesso “indovinate” (tra 59.32% e 79.6% delle volte), al contrario dei restanti tre ascolti (vale a dire 1, 2, 3) dove quasi mai (tra lo 0% e 5.08% delle volte) sono state individuate correttamente.

Un problema emerse molto presto: il genere musicale è una definizione

¹+0.5 è la soglia per cui iniziare ad a tenere in considerazione la correlazione, citando <https://www.dummies.com/education/math/statistics/how-to-interpret-a-correlation-coefficient-r/>

tropo ampia ² da usare *invertita*, ovvero estrapolando le condizioni meteorologiche originali partendo dal genere. Inoltre, l'associazione tra musica, colori, e clima è, ovviamente, molto soggettiva; specialmente se come nesso hanno l'umore/*mood* indotto nell'utente.

Sebbene il processo di *musicalization* soddisfi il presupposto di essere una “sonificazione” formale (cioè, stessa condizione meteorologica → stesso genere musicale), è anche vero che il genere potrebbe non essere sufficiente, per tutti i fruitori, al fine di associare una specifica condizione meteorologica, anche con l'aiuto del colore ³.

Difatti, tutte le webradio utilizzate sono incentrate su un genere specifico, ciò nonostante risulta comunque vasta la gamma di brani appartenenti allo stesso genere. Perciò le sessioni sperimentali effettive, sono state in qualche modo influenzate dal brano riprodotto in quel preciso lasso di tempo, e molto probabilmente, ripetendo i test in altri momenti, i risultati sarebbero potuti essere differenti.

Un suggerimento molto importante ricevuto da un collega è stato il seguente: “invece di utilizzare una webradio incentrata sul genere per ogni canale di streaming, sarebbe meglio creare playlist specifiche, meglio se definite dall'utente”. Ad esempio, ogni utente di Physiradio a lungo termine, dovrebbe essere in grado di personalizzare la sua configurazione in termini di:

- Che tipo di dato potrebbe essere preso in input;
- Creare playlist customizzabili;
- Associare quelle playlist a dei valori enumerabili.

In realtà, questa opzione è stata prevista nella progettazione: l'adozione di MQTT come linguaggio di interfaccia per controllare il dispositivo, è stata scelta per facilitare l'integrazione con sistemi come IFTTT [15], con un Connnettore MQTT, oppure Node-RED [38], pronto per MQTT. Physiradio può essere completamente controllato tramite un'API MQTT con comandi co-

²Ad esempio, un webradio “latino” potrebbe suonare “salsa”, “bachata”, “reggaeton”, “chacha”, ecc... che sono molto diversi tra loro. In effetti, in termini musicali, non esiste una definizione universalmente accettata di generi specifici.

³Non tutti i soggetti hanno notato / utilizzato i LED colorati, a volte perché l'ambiente non aveva un'illuminazione adeguata (ad es. troppa luce ambientale) e talvolta perché il soggetto ha deciso (e dichiarato) di non prestare attenzione a entrambe, musica e luce.

me ‘volume’, ‘stazione’, ‘città’(per ottenere dati meteorologici specifici), ecc. In questo modo, anche un utente semi-tecnico⁴ potrebbe implementare una configurazione adatta.

Naturalmente, la soluzione più semplice al “problema della soggettività” sarebbe quella di descrivere la mappatura nella documentazione (o tramite un piccolo display) in modo che una volta assimilata, nessun utente debba impegnarsi ad interpretare il dispositivo in seguito. Problema che è stato in parte risolto introducendo una legenda all’interno dell’app sviluppata con Unity (Figura 2.6 in alto a destra).

Altri consigli ricevuti, per quanto riguarda la diminuzione della soggettività a favore dell’oggettività, sono stati:

- Utilizzare musiche per balli molto più canonizzate (ad esempio ‘salsa’, ‘tango’, ‘can can’, ‘tarantella’ ecc...) probabilmente abbasserebbero il grado di soggettività o limiterebbero il problema della difficoltà nel definire un genere musicale.
- Sfruttare meglio le luci e i colori, per esempio far pulsare i LED a ritmo, seguendo il tempo del brano.

Durante le sessioni sperimentali, molti soggetti sono stati ispirati dal dispositivo e hanno iniziato a suggerire altre mappature basate sul loro lavoro e sulla loro esperienza di vita, come:

- Carico complessivo della CPU (in una server farm);
- Traffico di rete, non solo in termini di volume ma anche in termini di tipo di traffico (attacchi denial of service, spamming della posta con molti messaggi ripetuti, ecc...);
- Condizioni del traffico cittadino;
- Orari dei treni e ritardi (ad esempio “Se devi uscire di casa al momento giusto, ma ti stai radendo/facendo il bagno/etc.”);
- Tempi di cottura (ad esempio “Se hai voglia di un timer da forno musicale”);

⁴Sia IFTTT che NodeRED sono molto facili da usare e intuitivi.

- Tempi di attesa del call center (musica che rappresenta il tempo di attesa di un operatore);
- Vari contesti in cui sono coinvolti dei bambini, i quali sono mediamente più sensibili alla musica e ai colori rispetto ad un’utenza adulta.

In generale, qualsiasi situazione in cui vi è necessità di monitorare continuamente una ‘variabile’(dato), risulta utile sfruttare il potere di discernimento “superiore” (in termini di riconoscimento del cambio di stato) dell’ascolto rispetto alla vista [32]. Inoltre, risulterebbe fastidioso e/o noioso rappresentare la variazione di stato con un tono/suono semplice, proprio per questo Physiradio risulta essere un mezzo utile ed interessante.

Come accennato precedentemente nella sezione 2.2.3 e in altri paragrafi, le implementazioni future possibili riguardanti il software sono molteplici:

- Avendo creato il prototipo affinché interpreti i dati sul meteo in formato JSON, è possibile prendere in input dati/dataset differenti nello stesso formato, senza dover modificare troppo il codice. Magari anche dando la possibilità di scegliere dinamicamente quali fisicalizzare;
- Il mapping scelto è totalmente modificabile e aperto a chi volesse variarlo e migliorarlo;
- Il protocollo MQTT implementato potrebbe essere migliorato da più punti di vista. In prima istanza si possono implementare funzioni di *subscribe*, sia da parte dello script che gira sul device sia per l’applicazione, al fine di migliorare l’interazione tra le due parti. Successivamente si possono pensare migliorie per quanto riguarda la suddivisione in *subtopic* più specifici, divisi in classi, in modo da evitare una comunicazione “*broadcast*” poco selettiva. O ancora, si potrebbe modificare il codice affinché acquisisca un livello di sicurezza maggiore, utilizzando QoS di livelli più alti dello 0 (il più basso).

Riassumendo, gli esperimenti sul campo sono stati molto soddisfacenti: i soggetti sono diventati molto loquaci e hanno posto molte domande.

La mappatura non è risultata molto efficace, ma rimane un dato relativo, poiché l'obiettivo principale dell'esperimento era creare un prototipo di device IoT che, fisicalizzando dei dati (open data nello specifico), tramite una combinazione di musica (genere musicale) e colore, stimolasse curiosità e immaginazione nell'utente, cosa che riesce a fare con successo. Ultimo ma non meno importante, il design vintage del cabinet di legno scelto per il prototipo, si è rivelato una chiave di successo per l'accettazione del dispositivo da parte degli utenti.

Ringraziamenti

Per prima cosa, vorrei ringraziare il Prof. Andrea Trentini per avermi aiutato nello sviluppo di questo progetto; è raro trovare professori così disponibili, umanamente buoni e competenti allo stesso tempo.

Un ringraziamento speciale va a Claudio, Cristina e Chiara, la mia famiglia, a cui devo tutto. Senza il loro sostegno e il loro affetto, specialmente nei momenti di sconforto e difficoltà, probabilmente non sarei riuscito a completare questo percorso di studi. Sono orgoglioso di essere vostro figlio e fratello.

Ci terrei inoltre a ringraziare i miei compagni di studi Carlo, Noah, Stefano, Marco (D.N.), Alberto, Daniele e Marco (V.V.), con i quali ho condiviso momenti importanti della mia vita e della mia formazione.

Un saluto speciale va a tutti i componenti degli Elkir, la mia band nonché seconda famiglia, con cui ho portato avanti una splendida esperienza di vita in parallelo all'università e che spero duri il più possibile.

Infine ringrazio tutti i miei amici, vecchi e nuovi, che mi hanno sempre sostenuto e mi vogliono bene, a cui io stesso tengo moltissimo.

Grazie

Appendice A

Codice implementazione protocollo MQTT

Codice 1.1: libreria PubSubClient

```
1 #include <PubSubClient.h>
2 ...
3 WiFiClient mqttClient; //ESP8266WiFi network client
4 PubSubClient psclient(mqttClient);
5
6 //Callback function for MQTT protocol
7 void mqttCallback(char* topic, byte* payload, unsigned int length);
8 ...
9 void setup () {
10 ...
11 //MQTT setup
12 psclient.setServer("mqtt.atrent.it", 1883); //server name, port
13 psclient.setCallback(mqttCallback);
14 reconnect();
15 ...
16 }
17
18 void mqttCallback(char* topic, byte* payload, unsigned int length) {
19     //Parsing the payload
20     char msg[length];
21     for (int i = 0; i < length; i++) {
22         msg[i] = (char)payload[i];
23     }
24     //Adding EOF to cast msg to a string
25     msg[length] = '\0';
26
27     if(strcmp(topic, "PhysiRadio/Volume") == 0) {
28         if(strcmp(msg, "mute") == 0) {
29             Serial.println("Muting volume");
30             player.setVolume(0);
31             Serial.println("VOLUME : 0");
32         }
33     }
34     if(strcmp(msg, "unmute") == 0) {
```

```

36     Serial.println("UnMutting volume");
37     player.setVolume(TMP_VOLUME);
38     Serial.print("VOLUME : ");
39     Serial.println(TMP_VOLUME);
40 }
41
42 if(strcmp(msg, "+1") == 0) {
43     Serial.println("Volume +1");
44     TMP_VOLUME+=1.0;
45     if (TMP_VOLUME <= 100.0) {
46         player.setVolume(TMP_VOLUME);
47     } else {
48         TMP_VOLUME = 100.0;
49     }
50     VOLUME = TMP_VOLUME;
51     Serial.print("VOLUME : ");
52     Serial.println(TMP_VOLUME);
53 }
54
55 if(strcmp(msg, "-1") == 0) {
56     Serial.println("Volume -1");
57     TMP_VOLUME-= 1.0;
58     if (TMP_VOLUME >= 0.0) {
59         player.setVolume(TMP_VOLUME);
60     } else {
61         TMP_VOLUME = 0.0;
62     }
63     VOLUME = TMP_VOLUME;
64     Serial.print("VOLUME : ");
65     Serial.println(TMP_VOLUME);
66 }
67 }
68
69 if(strcmp(topic, "PhysiRadio/Station") == 0) {
70
71     if(strcmp(msg, "enable") == 0) {
72         t_api.enable();
73         Serial.println("API task Enabled");
74     }
75
76     if(strcmp(msg, "disable") == 0) {
77         t_api.disable();
78         Serial.println("API task Disabled");
79     }
80
81     if(strcmp(msg, "station1") == 0) {
82         Serial.println(msg);
83         Serial.println("Radio 1 - Classical -> Smoke/Haze");
84         connect_radio(&radioClassical);
85     }
86
87     if(strcmp(msg, "station2") == 0) {
88         Serial.println(msg);
89         Serial.println("Radio 2 - Metal -> Clear(sun) + High humidity");
90         connect_radio(&radioMetal);
91     }
92
93     if(strcmp(msg, "station3") == 0) {
94         Serial.println(msg);
95         Serial.println("Radio 3 - SmoothJazz -> Rain/Drizzle/Cloud/Fog
+ low humidity");
96         connect_radio(&radioSmoothJazz);
97     }

```

```

98
99     if(strcmp(msg, "station4") == 0) {
100        Serial.println(msg);
101        Serial.println("Radio 4 - SummerHits -> Sunny + low humidity");
102        connect_radio(&radioSummer);
103    }
104
105    if(strcmp(msg, "station5") == 0) {
106        Serial.println(msg);
107        Serial.println("Radio 5 - ExtremeMetal -> Thunderstorm/Tornado... ");
108        connect_radio(&radioExtremeMetal);
109    }
110
111    if(strcmp(msg, "station6") == 0) {
112        Serial.println(msg);
113        Serial.println("Radio 6 - LoFi -> Rain/Drizzle/Cloud/Fog
+ High humidity");
114        connect_radio(&radioLoFi);
115    }
116
117    if(strcmp(msg, "station7") == 0) {
118        Serial.println(msg);
119        Serial.println("Radio 7 - Xmas Songs -> Snow");
120        connect_radio(&radioXmas);
121    }
122}
123
124 //MQTT dynamic city change for API request
125 if(strcmp(topic, "PhysiRadio/City") == 0) {
126     cityRequested = String(msg);
127 }
128 }
129
130 void reconnect() {
131
132     //If it's not connected, try to connect
133     if (!psclient.connected()) {
134         Serial.print("Attempting MQTT connection... ");
135
136         //Attempt to connect
137         if (psclient.connect(clientId.c_str())) {
138             Serial.println(clientId + " connected to MQTT server");
139
140             //Subscribe to all subtopics
141             psclient.subscribe("PhysiRadio/#");
142
143         } else {
144             Serial.print("failed, rc=");
145             Serial.print(psclient.state());
146         }
147     }
148 }
```

Appendice B

Codice implementazione cooperative multitasking

Codice 1.2 : libreria TaskScheduler

```
1 #include <TaskScheduler.h>
2 ...
3 //One WiFi client for each task, to guarantee more connection stability
4 WiFiClient api;
5 WiFiClient music;
6 WiFiClient mqttClient;
7 ...
8 //Scheduler and tasks definition
9 Scheduler scheduler;
10 void task_apiCallback();
11 void task_audioStreamCallback();
12 void task_mqttCallback();
13 ...
14 //Call weather api function, every 10s
15 Task t_api(10 * TASK_SECOND, TASK_FOREVER, &task_apiCallback);
16 //Call audio stream task, always(the minimum interval possible)
17 Task t_stream(0, TASK_FOREVER, &task_audioStreamCallback);
18 //Call mqtt callback to get mqtt messages, every 100ms
19 Task t_mqtt(100, TASK_FOREVER, &task_mqttCallback);
20 ...
21 void setup () {
22     ...
23     //Scheduling tasks
24     scheduler.init();
25     Serial.println("Initialized scheduler");
26
27     scheduler.addTask(t_api);
28     Serial.println("added api task");
29     scheduler.addTask(t_stream);
30     Serial.println("added stream task");
31     scheduler.addTask(t_mqtt);
32     Serial.println("added mqtt task");
33
34     t_api.enable();
35     Serial.println("Enabled api task");
```

```

36     t_stream.enable();
37     Serial.println("Enabled stream task");
38     t_mqtt.enable();
39     Serial.println("Enabled mqtt task");
40     ...
41 }
42
43 //Just runs the scheduler
44 void loop(){
45     scheduler.execute();
46 }
47
48 //Task for http get : OpenWeatherMap API -> returns JSON
49 void task_apiCallback() {
50     if (!api.connect("api.openweathermap.org", 80)) {
51         Serial.println("Connection failed");
52         api.stop();
53         return;
54     }
55     Serial.println("Connected to API domain!");
56     //Send HTTP GET request
57     String httpMsg =
58     String("GET /data/2.5/weather?q=" +
59     cityRequested +
60     "&APPID=caaaf9c4e90fc42fe0baa94b8f1f6928&units=metric HTTP/1.0");
61     Serial.println(httpMsg);
62     api.println(httpMsg);
63     api.println("Host: api.openweathermap.org");
64     api.println("Connection: close");
65     if (api.println() == 0) {
66         Serial.println("Failed to send request");
67         api.stop();
68         return;
69     }
70
71     //Check HTTP status
72     char status[32] = {0};
73     api.readBytesUntil('\r', status, sizeof(status));
74     if (strcmp(status, "HTTP/1.1 200 OK") != 0) {
75         Serial.print("Unexpected response: ");
76         Serial.println(status);
77         api.stop();
78         return;
79     }
80
81     //Skip HTTP headers
82     char endOfHeaders[] = "\r\n\r\n";
83     if (!api.find(endOfHeaders)) {
84         Serial.println("Invalid response");
85         api.stop();
86         return;
87     }
88
89     //Allocate the JSON document
90     const size_t capacity = JSON_ARRAY_SIZE(4) +
91     2*JSON_OBJECT_SIZE(1) +
92     2*JSON_OBJECT_SIZE(2) +
93     4*JSON_OBJECT_SIZE(4) +
94     JSON_OBJECT_SIZE(5) +
95     JSON_OBJECT_SIZE(6) +
96     JSON_OBJECT_SIZE(14) +
97     1816;

```

```

98     // (1816, is an empirical byte value,
99     // to ensure enough space for the json document)
100    DynamicJsonDocument doc(capacity);
101
102    //Parse JSON object
103    DeserializationError error = deserializeJson(doc, api);
104    if (error) {
105      Serial.print("deserializeJson() failed: ");
106      Serial.println(error.c_str());
107      api.stop();
108      return;
109    }
110
111    //Extract values
112    JsonObject weather_0 = doc["weather"][0];
113    JsonObject main_0 = doc["main"];
114    const char* weather_0_main_char = weather_0["main"]; //eg "Clouds"
115    const char* city_name_char = doc["name"]; //eg "Milan"
116    city_name = String(city_name_char);
117    weather_0_main = String(weather_0_main_char);
118    humidity = main_0["humidity"];
119
120    Serial.println("Response:");
121    Serial.println(city_name);
122    Serial.println(weather_0_main);
123    Serial.print(humidity);
124    Serial.println(" % humidity");
125
126    //Execute weather-radiostation mapping
127    doMapping(weather_0_main);
128  }
129
130  void task_audioStreamCallback() {
131    if(!music.connected()) {
132      Serial.println("Reconnecting for audio stream...");
133      if(music.connect(httpHost, httpPort)) {
134        music.print(String("GET ") + httpPath + " HTTP/1.1\r\n" +
135        "Host: " + httpHost + "\r\n" +
136        "Connection: close\r\n\r\n");
137        Serial.println("connected to stream source");
138      }
139    }
140
141    //Play stream
142    if(music.available() > 0) {
143      uint8_t bytesread = music.readBytes(mp3buff, BUFF);
144      player.playChunk(mp3buff, bytesread);
145    }
146
147    //Code to control Physiradio easily by Serial input on Arduino IDE
148    if (Serial.available() > 0) {
149      // read the incoming byte for serial input control;
150      incomingByte = Serial.read();
151      switch(incomingByte) {
152        ... //VARI CASE PER IL CONTROLLO DEL DEVICE
153      }
154    }
155  }
156
157  void task_mqttCallback() {
158    //Connect if we're not already connected.
159    if (!psclient.connected()) reconnect();

```

```

160
161     //Process any events.
162     psclient.loop();
163 }
164
165 //Callback assigned to all mqtt clients
166 void mqttCallback(char* topic, byte* payload, unsigned int length) {
167     //VEDERE SEZIONE SUL PROTOCOLLO MQTT
168 }
169
170 void reconnect() {
171     //VEDERE SEZIONE SUL PROTOCOLLO MQTT
172 }
173
174 void doMapping(String weather) {
175     if(weatherMap->has(weather)) {
176         Serial.println("Doing mapping for " + weather + " weather");
177
178         radioStation tempRadio = weatherMap->get(weather);
179
180         //Cases for high humidity levels
181         if(humidity >= 85) { //
182             if( (tempRadio.name).equals(radioSummer.name) ) {
183                 tempRadio = radioMetal;
184             }
185
186             if( (tempRadio.name).equals(radioSmoothJazz.name) ) {
187                 tempRadio = radioLoFi;
188             }
189         }
190
191         Serial.print("Got: " + tempRadio.name + " --> ");
192
193         //If new radio is different from the actual, then connect
194         //look host OR port, because some radios use
195         //same host for multiple web radio switching on ports
196         if((strcmp((tempRadio.host), httpHost) != 0) | (tempRadio.port != httpPort))
197             connect_radio(&tempRadio);
198         } else {
199             Serial.println("Already on the same station");
200         }
201
202     } else {
203         Serial.println("Weather not mapped");
204     }
205 }
206
207 void connect_radio(radioStation *radio) {
208     fade_out_audio();
209
210     httpHost = radio->host;
211     httpPath = radio->path;
212     httpPort = radio->port;
213
214     if (!music.connect(httpHost, httpPort)) {
215         Serial.println("Radio Connection failed");
216         return;
217     }
218
219     Serial.print("Requesting stream: ");
220     Serial.println(httpPath);
221     music.print(String("GET ") + httpPath + " HTTP/1.1\r\n" +
222     "Host: " + httpHost + "\r\n" +

```

```
223     "Connection: close\r\n\r\n");
224     setLedsColor(radio->color);
225     fade_in_audio();
226     Serial.println(radio->name);
227
228 }


---


```

Bibliografia

- [1] Mesopotamian clay tokens - 5500 bc. <http://dataphys.org/list/mesopotamian-clay-tokens/>.
- [2] A. Trentini A. Carraturo. *Sistemi Embedded: Teoria e pratica.* Ledizioni.it, 2017.
- [3] Arduino.cc. Arduino ide. <https://www.arduino.cc/en/main/software>.
- [4] Anatoli Arkhipenko. Task scheduler: libreria cooperative multitasking per arduino ide. <https://github.com/arkhipenko/TaskScheduler>.
- [5] Benoît Blanchon. Arduinojson: a json library for embedded c++. <https://arduinojson.org/>.
- [6] D Bonafede, LA Ludovico, and G Presti. A proposal for the interactive sonification of the human face. In *International Conference on Computer-Human Interaction Research and Applications*, pages 163–169. SCITEPRESS, 2018.
- [7] Worlu Chijioke. Predicting listener’s mood based on music genre: An adapted reproduced model of russell and thayer. *Journal of Technology Management and Business*, 0:20, 06 2017.
- [8] Allan Coop. Sonification, musification, and synthesis of absolute program music. 07 2016.
- [9] Ian Craggs. M2mqtt : a mqtt client available for all .net platforms. <https://m2mqtt.wordpress.com/>.

- [10] Pierre Dragicevic and Yvonne Jansen. Official list of data physicalizations: List of physical visualizations and related artifacts. <http://dataphys.org/list>.
- [11] Eclipse Foundation. Eclipse mosquitto: An open source mqtt broker, sponsored by cedalo.com. <https://mosquitto.org/>.
- [12] Byeong-Jun Han, Seungmin Rho, Sanghoon Jun, and Eenjun Hwang. Music emotion classification and context-based music recommendation. *Multimedia Tools and Applications*, 47(3):433–460, 2010.
- [13] HM Government. Open data white paper - unleashing the potential. Technical report, 2012.
- [14] Steven Houben, Connie Golsteijn, Sarah Gallacher, Rose Johnson, Saskia Bakker, Nicolai Marquardt, Licia Capra, and Yvonne Rogers. Physiskit: Data engagement through physical ambient visualizations in the home. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1608–1619. ACM, 2016.
- [15] Ifttt.com. Ifttt. <http://ifttt.com>.
- [16] LC Technology Inc. Pagina ufficiale lc technology vs1003/1053. <http://www.lctech-inc.com/plus/view.php?aid=215>.
- [17] Yvonne Jansen. Data physicalization wiki: Bibliography. <http://dataphys.org/wiki/Bibliography>.
- [18] Yvonne Jansen, Pierre Dragicevic, Petra Isenberg, Jason Alexander, Abhijit Karnik, Johan Kildal, Sriram Subramanian, and Kasper Hornbæk. Opportunities and challenges for data physicalization. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3227–3236. ACM, 2015.
- [19] Jupyter.org. Jupyter, 2014. <https://jupyter.org>.
- [20] Debajyoti Karmaker, Md Al Imran, Niaj Mohammad, Mohaiminul Islam, and Md Nafees Mahbub. An automated music selector derived

from weather condition and its impact on human psychology. In *GSTF Journal on Computing (JoC), Global Science and Technology Forum*, volume 4, page 13, 2015.

- [21] Rohit Ashok Khot, Jeewon Lee, Deepti Aggarwal, Larissa Hjorth, and Florian 'Floyd' Mueller. Tastybeats: Designing palatable representations of physical activity. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2933–2942, 2015.
- [22] OpenWeather Ltd. Openweathermap: Open data platform that provides weather and satellite data worldwide. <https://openweathermap.org/>.
- [23] Luca A Ludovico and Presti Giorgio. The sonification space: a reference system for sonification tasks. *International Journal of Human-Computer Studies*, 85:72–77, 2016.
- [24] Michele Maffucci. Blog personale per descrizione e specifiche wemos d1 mini per la versione v1.0 (usata in physiradio). <https://www.maffucci.it/2017/05/13/iot-con-wemos-d1-mini-usando-arduino-ide-e-blynk/>.
- [25] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (iot). *IEEE Internet Initiative*, 1(1):1–86, 2015.
- [26] Nick O'Leary. Pubsubclient: libreria (per arduino ide) che fornisce un client per poter scambiare semplici messaggi di publish/subscribe con un server che supporta mqtt. <https://pubsubclient.knolleary.net/>.
- [27] Open Knowledge Foundation. History of the Open Definition, 2014.
- [28] sponsored by NumFOCUS opensource. Pandas, libreria analisi dati per python, 2009. <https://pandas.pydata.org/>.
- [29] Robert Plutchik. The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. *American scientist*, 89(4):344–350, 2001.

- [30] Interaction Design Lab Potsdam. Fritzing. <https://fritzing.org/home/>.
- [31] Aare Puussaar, Ian G Johnson, Kyle Montague, Philip James, and Peter Wright. Making open data work for civic advocacy. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):143, 2018.
- [32] David A. Rabenhorst, Edward J. Farrell, David H. Jameson, Thomas D. Linton Jr., and Jack A. Mandelman. Complementary visualization and sonification of multidimensional data. In Edward J. Farrell, editor, *Extracting Meaning from Complex Data: Processing, Display, Interaction*, volume 1259, pages 147 – 153. International Society for Optics and Photonics, SPIE, 1990.
- [33] Reference.com. What is a comfortable humidity level outdoors? <https://www.reference.com/science/comfortable-humidity-level-outdoors-a56b2ecf82004523>.
- [34] Robert.M.Candey@nasa.gov. Nasa definition of sonification. <https://spdf.gsfc.nasa.gov/research/sonification/>.
- [35] Simone Scaravati and Andrea Mario Trentini. Dataset creato ad hoc basato sull’analisi dati della state of the art. <https://doi.org/10.5281/zenodo.3612806>.
- [36] Simone Scaravati and Andrea Mario Trentini. Dataset derivante dal questionario sottoposto agli utenti. <https://doi.org/10.5281/zenodo.3627829>.
- [37] Simone Scaravati and Andrea Mario Trentini. Repository dedicata ai software sviluppati per physiradio. <https://github.com/simoneScaravati/Physiradio>.
- [38] IBM’s Emerging Technology Services and JS Foundation. Node-red: Low-code programming for event-driven applications. <http://nodered.org/>.

- [39] Andy Stanford-Clark (IBM) and Arlen (Cirrus Link Solutions) Nipper. Mqtt(message queue telemetry transport), 1999. <http://mqtt.org/>.
- [40] Marcin Szałomski. Una libreria per vs1053 mp3 codec breakout adattata per schede espressif esp8266 ed esp32. https://github.com/baldram/ESP_VS1053_Library.
- [41] Faisal Taher, John Hardy, Abhijit Karnik, Christian Weichel, Yvonne Jansen, Kasper Hornbæk, and Jason Alexander. Exploring interactions with physically dynamic bar charts. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3237–3246, 2015.
- [42] Joshua G Tanenbaum, Amanda M Williams, Audrey Desjardins, and Karen Tanenbaum. Democratizing technology: pleasure, utility and expressiveness in diy and maker practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2603–2612. ACM, 2013. non era tra quelli scaricati da dataphys, ma viene citato spesso.
- [43] Unity Technologies. Unity: Real-time development platform. <https://unity.com/>.
- [44] Andrea Trentini. Lombardy EPA *Obtorto Collo* data and anti-pollution policies fallacies. *Journal of e-Learning and Knowledge Society*, 10(2), 2014.
- [45] David Verweij, David Kirk, Kay Rogage, and Abigail Durrant. Domestic widgets: Leveraging household creativity in co-creating data physicalisations. 2019.
- [46] Wemos.cc. Pagina ufficiale wemos lolin d1 mini. https://docs.wemos.cc/en/latest/d1/d1_mini.html.
- [47] Wikipedia.org. Pagina wikipedia relativa al esp8266. <https://it.wikipedia.org/wiki/ESP8266>.

- [48] Wikiversity.org. Motivation and emotion/book/2014/plutchik's wheel of emotions. https://en.wikiversity.org/wiki/Motivation_and_emotion/Book/2014/Plutchik%27s_wheel_of_emotions.