Programmazione per l'IoT - Laurea Magistrale in Informatica Applicata - DiSPeA - Università degli Studi di Urbino Carlo Bo
Data di pubblicazione xx/xx/xxxx - DOI: xxxx/xxxxxxx

Dispositivo anti-collega rumoroso IoT

Francesco Rombaldoni[1]*, Emanuele Lattanzi[2]

Sommario
L'**Internet of Things** (IoT) rappresenta una delle rivoluzioni più significative della tecnologia contemporanea. Il termine, coniato negli anni '90 da Kevin Ashton, indica l'insieme di dispositivi fisici ("cose") — sensori, attuatori, microcontrollori, sistemi embedded — connessi tra loro e alla rete Internet, in grado di scambiare dati, interagire e prendere decisioni in modo automatico o assistito.

Già dagli anni '80 si sperimentavano reti di sensori per il monitoraggio industriale e ambientale, ma è con la diffusione della banda larga, dei protocolli wireless e dei microprocessori a basso consumo che l'IoT ha iniziato ad espandersi in modo capillare, dagli ambienti domestici (domotica, smart home) all'industria, dalla medicina alla mobilità urbana.

Dal punto di vista tecnologico, l'IoT integra componenti hardware (sensori, microfoni, telecamere, attuatori) con software dedicato per la raccolta, l'elaborazione e la trasmissione dei dati. La comunicazione può avvenire tramite protocolli standard (HTTP, MQTT, WebSocket) e i dati possono essere aggregati su piattaforme cloud per la visualizzazione e l'analisi.

Negli ultimi decenni, la crescita esponenziale di dispositivi smart ha portato alla nascita di nuove sfide e opportunità in termini di sicurezza, interoperabilità, gestione dei dati e privacy. L'IoT è oggi un elemento centrale nella cosiddetta "quarta rivoluzione industriale" (Industria 4.0) e nell'evoluzione degli ambienti di lavoro e di vita.

In questo contesto, il progetto presentato in questa relazione si propone di realizzare un dispositivo IoT per il monitoraggio del rumore in ambienti condivisi, sfruttando le tecnologie hardware e software tipiche dell'IoT: sensori audio, interfaccia web per la configurazione, automazione delle azioni correttive e cloud integration per la raccolta e l'analisi dei dati.

Keywords
Internet of Things — Open Space — RaspberryPi 3

[1] *Laurea Magistrale in Informatica Applicata, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
[2] *Docente di Programmazione per l'Internet of Things, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
***Corresponding author**: f.rombaldoni@campus.uniurb.it

Introduzione

Negli ultimi anni, l'**Internet of Things** (IoT) si è affermato come paradigma centrale per la digitalizzazione degli ambienti di vita e lavoro, integrando dispositivi intelligenti, sensori e attuatori in reti pervasive capaci di raccogliere, analizzare e condividere dati in tempo reale. L'IoT consente la creazione di ecosistemi interconnessi dove il monitoraggio ambientale, la risposta automatica agli stimoli esterni e la comunicazione tra oggetti fisici e sistemi digitali diventano elementi fondamentali per l'efficienza e la qualità della vita.

In parallelo, l'avanzamento dell'**Intelligenza Artificiale** (AI) ha rivoluzionato il modo in cui i dati provenienti dai dispositivi IoT vengono interpretati e utilizzati: algoritmi di analisi predittiva, riconoscimento di pattern e automazione decisionale permettono ai sistemi IoT di evolvere verso soluzioni "smart" capaci di adattarsi dinamicamente alle esigenze degli utenti e dell'ambiente circostante.

All'interno di questo scenario si colloca il progetto qui presentato, che affronta una problematica concreta e spesso trascurata negli ambienti di lavoro condivisi: il disturbo acustico generato da comportamenti troppo rumorosi di alcuni individui. In open space, uffici condivisi e laboratori, episodi di disturbo sonoro possono compromettere la concentrazione, la produttività e il benessere dei colleghi, generando un contesto lavorativo meno sereno e meno efficiente.

La crescente diffusione di dispositivi IoT, unita alle po-

tenzialità offerte da AI e automazione, offre la possibilità di affrontare questi problemi attraverso soluzioni tecnologiche innovative e non invasive. Il progetto "Dispositivo IoT Anti Collega Fastidioso" nasce da tale esigenza: realizzare un sistema intelligente, basato su Raspberry Pi e microfono USB, in grado di monitorare costantemente il livello di rumore ambientale, identificare situazioni di disturbo e intervenire in modo mirato quando necessario.

Il caso d'uso è chiaro: fornire un feedback acustico immediato e personalizzato tramite eco quando il livello di rumore supera una soglia configurabile, invitando il responsabile a moderare il proprio comportamento dandogli un feedback del proprio volume di voce. Gli eventi di superamento soglia vengono inoltre registrati e aggregati su una piattaforma cloud (ThingSpeak), permettendo l'analisi statistica e la visualizzazione storica degli episodi, utile sia per la gestione interna che per studi di ergonomia del lavoro.

Questo approccio si integra perfettamente nell'ecosistema dei dispositivi IoT:

- Il sistema è progettato per essere modulare, scalabile e facilmente integrabile con altri sensori o servizi.
- L'interfaccia web consente la configurazione remota dei parametri e la visualizzazione in tempo reale dello stato del dispositivo.
- L'invio automatico dei dati a un servizio cloud garantisce la persistenza e la fruibilità delle informazioni raccolte, aprendo la strada a possibili integrazioni con sistemi di AI per la profilazione degli ambienti, la predizione dei picchi di rumore o l'adozione di strategie di mitigazione personalizzate.

La motivazione alla base del progetto è quindi duplice: da un lato, rispondere a un'esigenza reale di miglioramento del benessere in ufficio tramite automazione e monitoraggio intelligente; dall'altro, sperimentare l'integrazione di tecnologie IoT e cloud per la creazione di soluzioni smart, adattabili e orientate all'utente.

Nei capitoli successivi verranno approfonditi gli aspetti architetturali, metodologici e implementativi del sistema, con particolare attenzione alle scelte tecniche adottate e alle potenzialità di estensione e personalizzazione offerte dalla piattaforma.

1. Methods

1.1 Architettura Hardware

Il dispositivo è basato su un **Raspberry Pi 3B+** dotato di microfono USB e altoparlante da 5W, alimentato tramite un comune powerbank USB. Questa soluzione garantisce bassi consumi energetici, piena portabilità e può essere installata in qualunque ambiente di lavoro dove sia presente una rete WiFi.

1.1.1 Motivazione della Scelta e Componenti

Perché Raspberry Pi?

- Consumo energetico ridotto: il sistema può funzionare per ore con un powerbank di media capacità.
- Supporto completo Linux per librerie audio Python (sounddevice, numpy).
- Porte USB per microfono e altoparlante.
- Gestione remota semplificata tramite SSH e SFTP (vedi sottosezione software).

Lista componenti:

- Raspberry Pi 3B+ (Raspberry Pi OS Legacy Desktop)
- Microfono USB PS3 Eye
- Altoparlante USB da 5W
- Powerbank USB

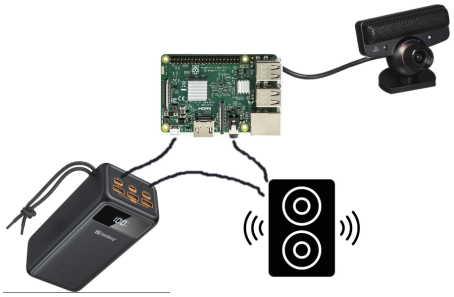

Figura 1. Setup hardware completo alimentato da powerbank USB.

Tutto l'hardware può essere gestito da remoto tramite accesso SSH per la shell e SFTP per lo scambio di file, permettendo aggiornamenti e manutenzione rapidi.

1.2 Architettura Software

Il software principale è sviluppato in Python e organizzato in diversi moduli chiave.

1.2.1 Monitoraggio Audio e Attivazione Eco

Lo script centrale, `audio_monitor_echo_service.py`, registra continuamente l'audio dal microfono USB, calcola i valori RMS e dBFS e attiva l'effetto eco se il volume supera la soglia configurata. Espone inoltre una API REST locale per l'interazione in tempo reale.

Estratto di codice significativo:

```
        # Ciclo di monitoraggio
        def monitor_thread():
        global current_volume, mic_enabled
        while True:
        if mic_enabled:
        audio = sd.rec(int(FRAME_DURATION *
            SAMPLE_RATE),
        samplerate=SAMPLE_RATE, channels=CHANNELS,
             dtype='int16')
```

```
        sd.wait()
        arr = audio.astype(np.float32)
        rms = np.sqrt(np.mean(arr**2))
        dbfs = rms_to_dbfs(rms)
        if dbfs > THRESHOLD_DBFS:
        log(f"Superata soglia! dbfs={dbfs:.1f}.
            Attivo eco...")
        trigger_echo(arr.flatten())
```

1.2.2 Effetto Eco: Simulazione del Delay a Nastro

L'algoritmo di eco simula il classico delay a nastro: il frame audio registrato viene riprodotto con tap (ripetizioni) configurabili, ritardo, feedback e fading del volume.

```
        def create_echo_buffer(audio, taps,
            delay_sec, feedback, start_vol,
            end_vol):
        n_samples = len(audio)
        delay_samples = int(delay_sec *
            SAMPLE_RATE)
        envelope = np.linspace(start_vol, end_vol,
             taps)
        out = np.zeros(n_samples + delay_samples *
             taps, dtype=np.float32)
        out[:n_samples] += audio.flatten()
        for i in range(taps):
        start = delay_samples * (i + 1)
        end = start + n_samples
        if end > len(out): break
        echo = audio.flatten() * envelope[i]
        if feedback > 0 and i > 0:
        echo += out[start-delay_samples:end-
            delay_samples] * feedback
        out[start:end] += echo
        return out
```

Parametri come ritardo, tap, feedback e fading sono regolabili dalla dashboard (vedi sottosezione dashboard). Questa implementazione riprende il comportamento delle unità echo vintage a nastro, dove tempo di delay, feedback e livelli di uscita erano fondamentali.

1.2.3 Avvio Automatico del Servizio

Lo script di monitoraggio viene avviato automaticamente all'accensione tramite un servizio systemd. **Esempio di unit file systemd:**

```
        [Unit]
        Description=Audio Monitor & Echo Service
        After=network.target sound.target

        [Service]
        Type=simple
        User=rombo
        ExecStart=/home/rombo/00_APPLICATION/
            start_audio_monitor.sh
        WorkingDirectory=/home/rombo/00
            _APPLICATION
        Restart=on-failure

        [Install]
        WantedBy=multi-user.target
```

Qualsiasi aggiornamento agli script Python o ai file di configurazione può essere effettuato da remoto via SSH/SFTP e attivato riavviando il servizio.

1.2.4 API REST e Configurazione

Il servizio audio espone endpoint REST per la lettura del volume, la modifica delle soglie e la gestione dei parametri dell'eco:

- `/volume` — Livello di rumore attuale (RMS, dBFS)
- `/threshold` — Leggi/imposta la soglia
- `/echo_params` — Leggi/imposta parametri eco
- `/lockout` — Leggi/imposta tempo di lockout
- `/status` — Stato completo del sistema
- `/start_echo` — Avvio manuale dell'eco

La configurazione è persistente tramite `threshold_config.json`.

1.3 Dashboard e Controllo Real-Time

La dashboard web, accessibile tramite l'IP statico del Raspberry Pi (protetta da Nginx e password hardcoded), offre controllo completo e visualizzazione in tempo reale del sistema.

1.3.1 Funzionalità Principali

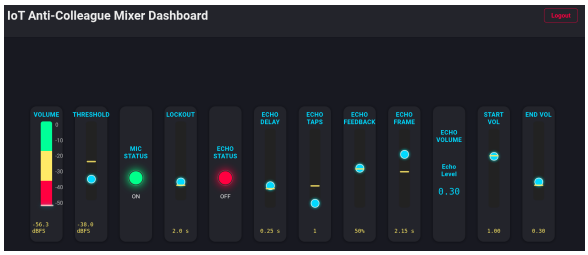

Figura 2. Dashboard web.

- **Vmeter:** Visualizzazione live del dBFS rilevato dal microfono.
- **Slider:** Regolazione soglia, lockout, delay eco, tap, feedback, volume iniziale/finale e durata frame.
- **Lampade di stato:** Indicazione di attività microfono ed eco.
- **Animazione eco:** Visualizzazione live del "livello eco" durante la riproduzione.

1.3.2 Aggiornamento Real-Time

La dashboard JavaScript (`mixer.js`) interroga l'API REST backend ogni 200ms e aggiorna istantaneamente tutti i controlli e indicatori.

```
        // Poll dello stato e aggiornamento UI
        async function pollStatus() {
                const resp = await fetch("/api/
                    status");
                const data = await resp.json();
                // Aggiorna Vmeter, slider,
                    lampade, timer, ecc.
        }
        setInterval(pollStatus, 200);
```

L'accesso alla dashboard è protetto da Nginx come reverse proxy e da una password (memorizzata in `password.json` e verificata nel frontend Flask).

1.4 Log degli Eventi e Analisi Cloud

Ogni volta che l'eco viene attivato automaticamente (superamento soglia), un evento viene registrato sia localmente (CSV) che su ThingSpeak cloud:

1.4.1 Registrazione e Invio Dati

- **Log CSV:** `event_log.csv` contiene timestamp e dBFS per ogni evento.
- **ThingSpeak:** Gli eventi sono inviati via HTTP REST API usando le credenziali in `config.json`.

```
def log_event_thingspeak(dbfs, ts_iso):
payload = {
        "api_key": write_api_key,
        "field1": dbfs,
        "field2": ts_iso,
}
url = "https://api.thingspeak.com/update.
    json"
requests.post(url, data=payload, timeout
    =6)
```

La dashboard ThingSpeak fornisce grafici storici e statistiche sugli eventi di rumore, essenziali per comprendere i pattern e migliorare l'ergonomia dell'ambiente di lavoro.

1.5 Deployment e Impatto

Il sistema è stato installato in uno spazio co-working e ha immediatamente ridotto comportamenti rumorosi e urla (soprattutto telefonate e esclamazioni ad alto volume), rendendo l'ambiente sensibilmente più tranquillo e produttivo.

2. Risultati

2.1 Risultati della sperimentazione

2.1.1 Analisi dei dati raccolti

Durante le due settimane di sperimentazione, il dispositivo è stato installato al centro dell'ufficio, alimentato tramite powerbank USB (ricaricato ogni sera). L'approccio iniziale era giocoso, ma ben presto il sistema ha stimolato una riflessione collettiva sul comportamento in open space, risultando non invasivo e accettato dalla maggior parte dei colleghi.

I parametri impostati tramite la dashboard web sono rimasti pressoché invariati per tutto il periodo, grazie a una taratura efficace fin dalla prima installazione.

Grafico 1: Volume dei trigger nel tempo Il primo grafico mostra l'andamento del livello di volume `noise_dbfs` che ha causato il trigger dell'eco, in funzione del tempo.

Come si osserva, il livello di volume che ha causato l'attivazione dell'eco è rimasto pressoché costante, a conferma che il comportamento indesiderato era sempre riconducibile a superamenti netti della soglia impostata.

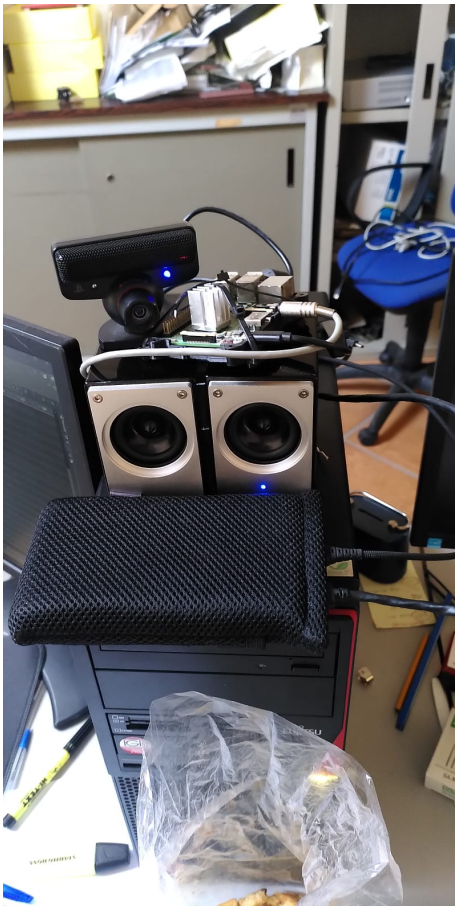

Figura 3. Dispositivo in azione in un vero ambiente di co-working.

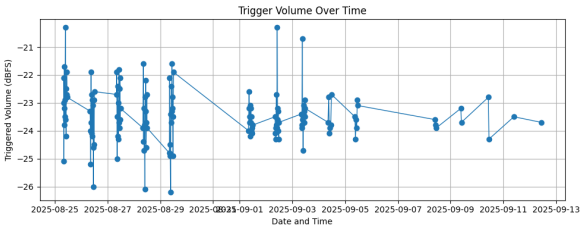

Figura 4. Livello di volume (dBFS) degli eventi di trigger nel tempo.

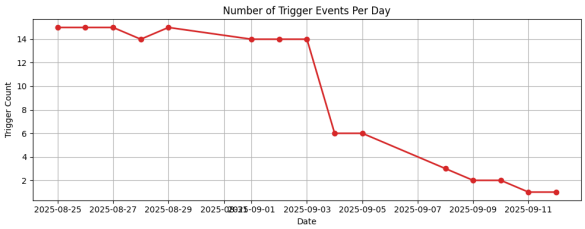

Figura 5. Numero di eventi di trigger registrati per ciascun giorno lavorativo.

Grafico 2: Numero di trigger giornalieri Il secondo grafico mostra il numero di attivazioni giornaliere del sistema.

Questo grafico evidenzia chiaramente il trend discendente degli episodi di disturbo: nei primi giorni il numero di trigger era elevato, ma progressivamente si è ridotto fino a stabilizzarsi su minimi fisiologici, segno che i colleghi hanno interiorizzato la necessità di mantenere un comportamento più rispettoso.

2.1.2 Considerazioni sull'impatto sociale e ambientale

L'esperienza in ufficio ha mostrato come la semplice presenza del dispositivo abbia sensibilizzato la maggior parte dei colleghi a tenere un tono di voce più consono e a spostarsi nella sala comune per rispondere alle telefonate. Su otto colleghi, solo uno ha manifestato una certa avversione percependo il sistema come un controllo invasivo, ma il clima generale è rapidamente migliorato e si è consolidato un comportamento più attento al benessere collettivo.

La sperimentazione, durata **due settimane lavorative** (escludendo sabato e domenica), ha raggiunto pienamente l'obiettivo del progetto: non solo ridurre i picchi di rumore, ma soprattutto favorire una cultura della collaborazione e del rispetto nei luoghi di lavoro condivisi.

2.1.3 Conclusioni operative

I dati raccolti e i grafici presentati dimostrano che l'approccio tecnologico, se ben calibrato e comunicato, può essere uno strumento efficace per la gestione del benessere in open space, favorendo l'autoregolazione e la consapevolezza dei comportamenti.

3. Conclusioni

L'esperimento condotto con il dispositivo IoT anti-collega rumoroso ha raggiunto pienamente il suo obiettivo: mitigare in modo efficace il problema del rumore eccessivo negli ambienti di lavoro condivisi. Fin dalla sua installazione, il sistema ha favorito una maggiore consapevolezza nei comportamenti dei colleghi, portando a un netto miglioramento del clima generale in ufficio e alla riduzione degli episodi di disturbo.

Tuttavia, come emerso durante la sperimentazione, una soluzione di questo tipo non può essere adottata in modo continuativo e permanente. La presenza costante dell'eco sonoro, seppur utile in fase iniziale per sensibilizzare e correggere le cattive abitudini, rischia a lungo andare di generare un senso di oppressione e controllo eccessivo. Questo aspetto è stato confermato dal feedback dei partecipanti, che hanno apprezzato l'efficacia del sistema ma hanno sottolineato l'importanza di un approccio equilibrato e non invasivo.

Qui emerge la vera forza dell'IoT e della raccolta dati automatizzata: grazie alla piattaforma online e al monitoraggio continuo, il datore di lavoro può scegliere di mantenere attivo il sistema in modalità "passiva", con gli altoparlanti spenti ma con la raccolta dati sempre attiva. In questo modo, è possibile tenere sotto controllo l'ambiente lavorativo senza intervenire direttamente, analizzando i trend e rilevando eventuali peggioramenti nel comportamento acustico. Solo in caso di necessità, e dopo aver affrontato la questione con i dipendenti, si può valutare di riattivare l'eco come strumento di deterrenza.

Per quanto riguarda gli sviluppi futuri, il progetto può essere ulteriormente ottimizzato sia in termini di consumi energetici che di ingombro, integrando soluzioni hardware più compatte e algoritmi di gestione più efficienti. Tuttavia, dal punto di vista operativo, la sperimentazione ha dimostrato che già l'attuale sistema svolge egregiamente il proprio lavoro, rappresentando un valido esempio di applicazione intelligente delle tecnologie IoT per il benessere negli ambienti di lavoro.

4. Risorse

Tutti i file sorgente e la documentazione del progetto sono disponibili nel repository GitHub (`https://github.com/tuo-username/tuo-repo`). Qui di seguito viene fornita una breve panoramica e descrizione di ciascun file presente nel progetto:

- **Application/audio_monitor_echo_service.py**: Script principale che esegue il monitoraggio continuo del livello di rumore tramite microfono USB su Raspberry Pi. Attiva l'effetto eco quando il volume supera la soglia impostata, gestisce il lockout del microfono e i parametri dell'effetto eco. Espone una REST API per la configurazione remota e registra gli eventi su ThingSpeak e in locale (CSV).
- **Application/webapp.py**: Web application Flask che fornisce la dashboard di controllo in tempo reale. Permette la visualizzazione dello stato del sistema, la modifica dei parametri (soglia, eco, lockout) e la gestione dell'autenticazione. Funziona da frontend e proxy verso la REST API del backend audio.
- **Application/static/mixer.js**: Script JavaScript che gestisce l'interfaccia utente della dashboard web. Aggiorna in tempo reale i controlli, visualizza il livello di rumore, lo stato del microfono e dell'eco, e invia le modifiche dei parametri alla REST API.
- **Application/static/style.css**: Foglio di stile CSS utilizzato dalla dashboard web per una visualizzazione moderna e responsive. Gestisce l'aspetto dei controlli, slider, lampade di stato, timer e layout generale.
- **Application/templates/login.html** & **templates/dashboard.html**: File HTML template per Flask. `login.html` mostra il form di autenticazione, `dashboard.html` visualizza la dashboard con tutti i controlli e indicatori del sistema.
- **Application/threshold_config.json**: File di configurazione contenente i parametri runtime del sistema: soglia

di trigger, lockout del microfono, parametri eco (delay, tap, feedback, volume iniziale/finale, durata frame). Modificabile tramite la dashboard o direttamente.

- **Application/config.json**: File di configurazione contenente le API Key e l'ID del canale ThingSpeak. Utilizzato dal backend Python per inviare i dati al cloud. Da mantenere privato (non va pubblicato su repository pubblici).
- **Application/event_log.csv**: File di log locale dove vengono salvati gli eventi di attivazione dell'eco: timestamp ISO8601 e livello di rumore (dBFS). Utile per analisi storiche e backup dei dati.
- **Application/audio_monitor_service.py** & **Application/start_audio_monitor.sh**: File di servizio systemd e script di avvio che permettono di eseguire il backend Python come servizio automatico all'accensione del Raspberry Pi.
- **Application/password.json**: File di configurazione che contiene la password per l'accesso alla dashboard di controllo. Da mantenere privato.
- **Report/Report/report.tex**: Il documento LaTeX principale del progetto, contenente la relazione tecnica, l'analisi dei risultati, le figure, i grafici e la descrizione dettagliata dell'implementazione.
- **Report/Report/sample.bib**: File BibTeX di esempio per la bibliografia del report LaTeX (se necessario).

Tutti i file sono documentati e commentati nel codice e nella repository GitHub. Per domande, richieste di estensione o segnalazione bug, consultare il README o aprire una issue direttamente sul repository.