UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

STUDIO DI SOLUZIONI LOW-COST PER IL RILEVAMENTO DELLA PRESENZA

APPLICAZIONE AL CASO DI COMPLEMENTI D'ARREDO SMART

Relatore: Prof. Antonio Rodà

Laureando: Riccardo Selmin

Correlatore: Dott. Filippo Carnovalini

ANNO ACCADEMICO 2023 – 2024

Data di laurea 12/03/2024

Indice

Abstract

Questa tesi si occupa di studiare e progettare una soluzione per rilevare la presenza di persone alla scrivania o all'interno della stanza, col fine di integrare i risultati ottenuti in un progetto finalizzato al miglioramento del comfort ambientale in ambienti di ufficio. Partendo dai sensori più utilizzati per rilevare la presenza di persone e comprendendone il funzionamento fisico, passiamo poi a spiegare i componenti scelti per il progetto con il rispettivo funzionamento e le proprie caratteristiche. Successivamente, si analizzano i protocolli utilizzati e i principali blocchi di codice scritto per gestirne il funzionamento. Come ultima parte si vanno a discutere i risultati ottenuti mediante grafici e le prestazioni del prototipo.

Parole chiave: Presenza, Microcontrollore, Comfort Ambientale

Capitolo 1

Introduzione

1.1 Ambiti e dispositivi utilizzati per il rilevamento della presenza

In questo lavoro di tesi esponiamo la sperimentazione relativa all'inclusione di sistemi di rilevamento della presenza da usare in complementi d'arredo per interni (come uffici), in particolare, si intende fornire dei pannelli fonoassorbenti da ufficio di sensori per verificare se la persona è presente all'interno di una stanza o se è seduta alla scrivania. I dispositivi per il rilevamento della presenza sono utilizzati in una vasta gamma di settori per identificare la presenza o l'assenza di persone e/o oggetti in determinate aree, sfruttando differenti tecnologie. Per comprenderne al meglio il funzionamento è necessario conoscere la definizione di presenza. Con "rilevamento di presenza" di intende la capacità di rilevare la presenza o l'assenza di persone o oggetti in un determinato ambiente utilizzando sensori e tecnologie. In questa situazione i sensori sono utilizzati per rilevare e comunicare in tempo reale la presenza o l'assenza di individui.

Le applicazioni principali di questi dispositivi sono identificate negli ambiti riguardanti:

- Sicurezza (rilevamento di intrusi, controllo accessi);
- Automazione domestica, edifici intelligenti ed arredo smart (illuminazione, climatizzazione, sicurezza, insonorizzazione);
- Sanità (monitoraggio pazienti, prevenzione delle infezioni);
- Trasporti (sistemi di conteggio passeggeri, sicurezza veicolare);
- Industria (automazione, sicurezza sul lavoro);

Per creare sistemi utili a questi ambiti si utilizzano diversi tipi di sensori, che in base allo scopo, possiamo trovare quasi ovunque dato il vasto utilizzo. Un vantaggio

che possono offrire questi sensori è il loro basso costo e il basso consumo di energia, garantendo così un vantaggio in termini economici durante la fase di produzione.

1.1.1 Sensore ad infrarossi passivo - PIR

Figura 1.1: Sensore ad infrarossi passivo - PIR

Il **sensore ad infrarossi passivo** (detto più comunemente **PIR**, acronimo di *Passive InfraRed*) è un sensore elettronico che rileva la radiazione infrarossa (IR, tra 700nm e 1mm) irradiata dagli oggetti nel suo campo visivo. Sono sensori molto spesso utilizzati come sensori di movimento: infatti, ogni oggetto con temperatura superiore allo zero assoluto emette energia sottoforma di radiazioni luminose invisibili all'occhio umano perché di frequenza minore allo spettro della luce visibile, ma con questi sensori progettati appositamente per tale scopo, è possibile identificarla.
I PIR si dicono passivi perchè non emettono nessuna energia, ma lavorano solo rilevando l'energia sprigionata dagli oggetti, avendo così il vantaggio di avere un basso consumo elettrico.
Il suo funzionamento è semplice: non rileva automaticamente un movimento, ma rileva la variazione di temperatura memorizzata come riferimento alla sua attivazione. Quando un corpo passa davanti al PIR, esso rileva una rapida variazione di temperatura rispetto al riferimento preso e questo rapido cambiamento identifica la presenza di un corpo. Il PIR è composto principalmente da un nucleo fatto di materiale piroelettrico (materiale che genera energia quando esposto al calore), con una lente al di sopra di esso detta *lente di Fresnel* (fig. 1.2), che hanno le particolarità di diminuire la distanza focale, avere meno ingombro e pesare meno, a differenza delle lenti più comuni. Molto importante è tenerle pulite senza interperie sulla loro superficie, per evitare misurazioni fasulle e la limitazione del campo visivo.
Per coprire il sensore, viene posizionata una finestra di plastica di materiale traslucido per la luce visibile, e trasparente per le radiazioni ad infrarossi. Questa finestra serve per agire da filtro, per evitare l'ingresso nel sensore di polvere ma soprattutto per direzionare gli infrarossi verso il nucleo. Il tutto è completato da un circuito integrato che serve per elaborare le misure.

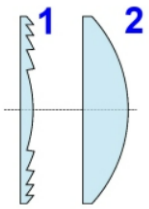


Figura 1.2: 1-Lente di Fresnel, 2-Lente comune

Questo sensore è progettato principalmente per captare le variazioni date dalla temperatura corporea, quindi se si vogliono identificare oggetti, è consigliabile l'utilizzo di altre tipologie di sensori. Il sensore PIR non può essere utilizzato in ambienti dove ci sono forti variazioni di calore. Le sue applicazioni più comuni le troviamo nei sistemi di sicurezza (per esempio di antifurto) e di illuminazione automatica.

1.1.2 Sensore ad ultrasuoni

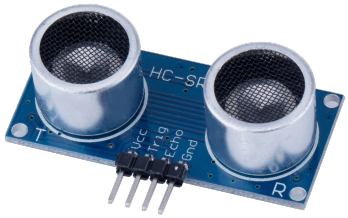


Figura 1.3: Sensore ad ultrasuoni HC-SR04

Il **sensore ad ultrasuoni** è un sensore che utilizza gli *ultrasuoni* per rilevare la presenza di oggetti e misurare distanze. Questo tipo di sensore utilizza delle onde meccaniche sonore alla convenzionale frequenza di 40kHz, quindi superiore a quella del suono udibile dall'orecchio umano, da cui deriva il nome ultrasuono. Questo avviene trasmettendo onde sonore ad alta frequenza (adatta a definirle ultrasuoni) ed analizzando i segnali dell'eco riflessi dagli oggetti presenti nel loro percorso. Calcolando il tempo di volo dei segnali di eco, i sensori ad ultrasuoni possono determinare la distanza dall'oggetto.
Di seguito vediamo un esempio numerico utilizzando il sensore ad ultrasuoni HC-SR04, dotato di due pin, rispettivamente di *trigger* (transmitter) ed *echo* (receiver).

Il sensore invia un impulso ad ultrasuoni tramite il pin di trigger verso un oggetto, il quale rimbalzerà su di questo tornando verso il sensore, che ne rileverà il ritorno tramite il pin di echo (fig. 1.4). Per calcolare la distanza, bisogna considerare la velocità del suono nell'aria

$$\nu = 343 m/s$$

ed utilizzare la formula

$$d = (pulseTime * \nu)/2$$

con d definita come distanza da misurare e pulseTime definito come tempo trascorso tra trasmissione e ricezione dell'impulso. Importante calcolare la metà della moltiplicazione, altrimenti si calcolerebbe la distanza percorsa doppia (andata e ritorno dell'onda).

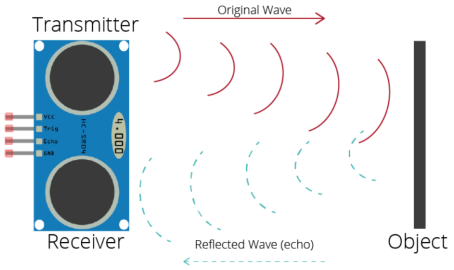


Figura 1.4: Illustrazione meccanismo rilevazione distanza

Gli utilizzi di questi sensori possiamo trovarli dall'automotive (parcheggio assistito, rilevazione di ostacoli, guida autonoma, ecc) ai sistemi di sicurezza, dalla robotica (per la navigazione) alla nautica (misurazione delle profondità), dalla medicina (per l'imaging) all'agricoltura.

1.1.3 Sensore di pressione

Figura 1.5: Sensore di pressione capacitivo ceramico dell'acqua WNK83MA

Il **sensore di pressione** è uno strumento analogico che consente di misurare la pressione meccanica esercitata da un liquido o un gas sul sensore, traducendola in un segnale elettrico. In base all'utilizzo necessario, si possono scegliere differenti tecnologie di misura, le più popolari sono:

- **Sensore di pressione capacitivo**: misura la **capacità elettrica**, cioè il rapporto tra la quantità di carica che il conduttore riesce ad accumulare e la differenza di potenziale tra due conduttori. Consiste in un condensatore variabile, composto da un materiale dielettrico tra una piastra fissa ed una mobile, che cambia la posizione in base alla pressione ad essa applicata. Cambiando la posizione, di conseguenza, cambia la distanza tra le due piastre e quindi la capacità del condensatore. Questi sensori sono limitati a capacità basse (circa 40 bar), quindi spesso è necessario usare un amplificatore per aumentare la sensibilità e rendere più precisa la misurazione.
- **Sensore di pressione piezoresistivo**: si basa sul principio fisico della **piezoresistenza**, cioè la resistenza che un materiale oppone al passaggio della corrente quando varia la sua resistività elettrica, cambiamento dovuto alla deformazione del materiale causato da una sollecitazione esterna. La sua caratteristica risposta-deformazione però non è lineare, ma tende ad essere quadratica su alti valori di misura, limitandone la quantità di applicazioni. In base al materiale usato, può arrivare fino a 1000 bar.

Questi sensori trovano un vasto utilizzo nell'automotive per controllare la pressione delle gomme, nei dispositivi medici, nell'aeronautica, nella meteorologia per informazioni utili alle previsioni del tempo e nella robotica.

1.1.4 Sensore a microonde

Figura 1.6: Sensore di movimento a microonde 180° SKU 5571 VT-8036

Il **sensore a microonde** è un dispositivo che emette onde elettromagnetiche con lo scopo di identificare la presenza o meno di un corpo nel suo campo visivo (per esempio il campo visivo del sensore qui sopra rappresentato è di 180°).
Il suo funzionamento è il seguente: il sensore emette la radiazione, la quale va incontro ad un corpo. Se quest'ultimo è fermo, l'onda ritorna con la stessa frequenza verso il sensore, se invece è in movimento, avviene un fenomeno chiamato *effetto Doppler*, definito come "cambiamento, rispetto al valore originario, della frequenza o della lunghezza d'onda percepita da un'osservatore raggiunto da un'onda emessa da una sorgente che si trova in movimento rispetto all'osservatore stesso". Perciò, quando si verifica un movimento lungo la congiungente tra corpo e sensore nel campo visivo dell'ultimo, le onde che ritornano al sensore hanno una frequenza o lunghezza d'onda

differente rispetto a quando sono state emesse, e questo fa attivare il sensore. Se il corpo si allontana, la frequenza sarà minore, viceversa se si avvicina sarà maggiore. Questi sensori sono molto più efficaci se si vogliono rilevare movimenti molto piccoli, inoltre hanno il vantaggio di non avere difficoltà ad operare in condizioni di calore. Il loro utilizzo lo ritroviamo in apparecchi di illuminazione automatici, in sistemi di sicurezza e allarme, sistemi di controllo dell'accesso e nei sistemi radar.

1.1.5 Sensore fotoelettrico

Figura 1.7: Fotocellula

Il **sensore fotoelettrico**, più comunemente conosciuto come *fotocellula*, è un dispositivo utilizzato per rilevare distanza o identificare la presenza o l'assenza di un corpo. Utilizza una sorgente ottica, tipicamente una lampada ad infrarossi o un diodo a led, e un ricevitore fotoelettrico, che può essere un fotodiodo o un fototransistor. Il sistema più comune è il *sistema a barriera*, che consiste in un ricevitore ed un emettitore posti l'uno di fronte all'altro. L'emettitore emette un fascio di luce infrarossa che va a terminare sul ricevitore. Se il fascio di luce tra i due viene interrotto, il ricevitore identifica la presenza di un corpo. Questa interruzione può essere utilizzata per un meccanismo utile al sistema dove la fotocellula è installata.
Un impiego tipico di questo sensore è l'utilizzo nelle aperture e chiusure automatiche; come ad esempio nei cancelli automatici, dove, nella sua fase di chiusura, se interrompo il fascio di luce della fotocellula, viene inviato un segnale dal ricevitore che ne blocca la chiusura. Altre applicazioni sono il controllo della sicurezza di garage e parcheggi, sistemi di controllo di accessi e ambiti industrali, come il controllo di produzione e la sorveglianza di aree sensibili.

Di seguito una tabella riassuntiva dei sensori elencati:

Tipo di sensore	Vantaggi	Svantaggi
PIR	Basso consumo elettrico	Non identifica oggetti Sensibile alle grandi variazioni di calore
Ultrasuoni	Identificazione degli oggetti	Distanza di misura limitata
Pressione	Range di misure alto	Limitato a capacità basse Caratteristica risposta-deformazione non lineare
Microonde	Molto sensibili ai movimenti Nessuna difficoltà ad operare in grandi condizioni di calore	Non efficaci nel rilevare grandi movimenti
Fotoelettrico	Rilevazione degli oggetti senza il contatto con essi	Sensibile alle condizioni ambientali

Tabella 1.1: Vantaggi e svantaggi dei sensori elencati.

1.2 Descrizione e scopo del progetto

Il progetto in esame deve essere integrato nei complementi d'arredo per interni, che essendo dotati di componentistica elettronica possono essere definiti *arredo smart*. Questi dispositivi saranno destinati principalmente ad ambiti lavorativi, come degli uffici per esempio, posti direttamente all'interno dell'arredo così da non intaccare il design dell'interno. Il prodotto finale sarà un pannello da arredo contenente l'elettronica necessaria per due principali obiettivi: mascherare il rumore ambientale e di rilevare la presenza all'interno della stanza, andando così a migliorare il comfort negli ambienti di co-working. Tutto avviene tramite l'utilizzo di due microfoni ed un sensore di prossimità (o distanza, a seconda dall'utilizzo necessario) che funge anche da sensore di luce ambientale. Il sistema viene comandato dal microcontrollore ESP32-WROOM-32E, dotato del proprio progetto firmware dedicato. Per l'eliminazione del rumore ambientale, saranno presenti degli altoparlanti che, utilizzando un algoritmo capace di distinguere i rumori ambientali dai rumori voluti (per esempio la voce di una persona) catturati dai microfoni, oltre a distinguere la direzione della provvenienza del rumore (destra o sinistra), emetteranno un rumore adatto a mascherare il rumore presente. Un compito secondario ma non meno importante è quello di controllare il risparmio energetico, per esempio verificando se la persona è seduta alla scrivania e in caso contrario spegnere il monitor e le luci.
L'argomento trattato di questa tesi è la progettazione e lo sviluppo del **rilevamento della presenza**.

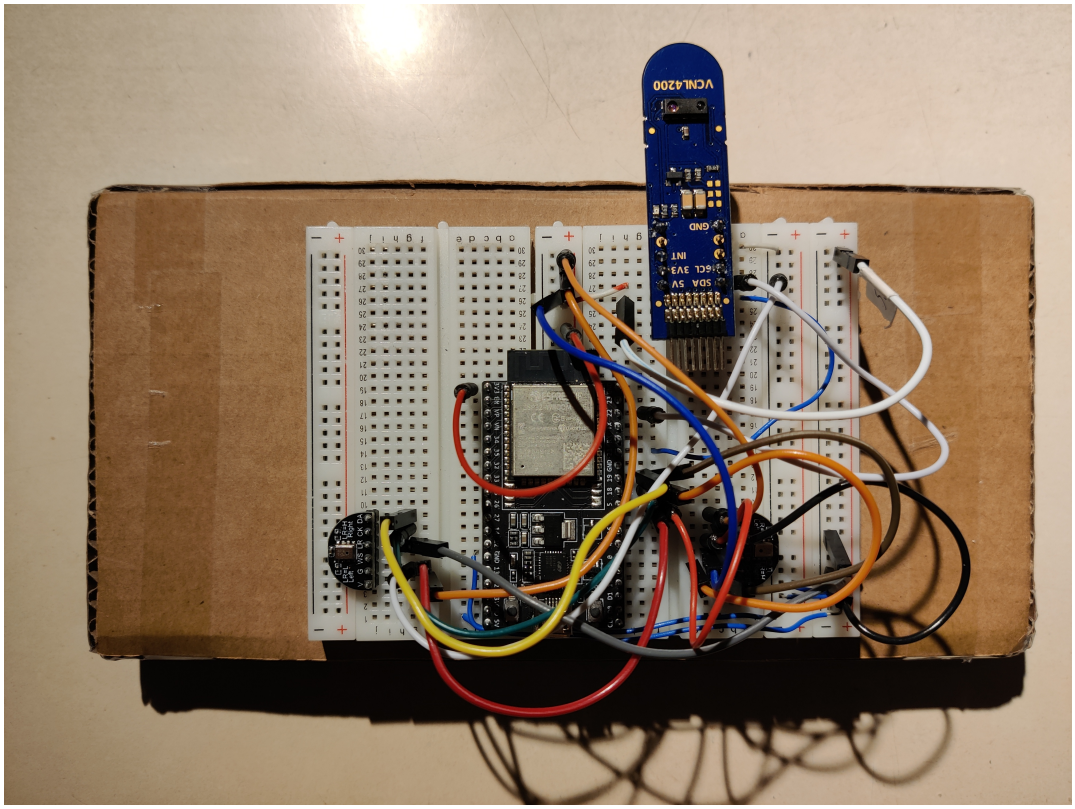


Figura 1.8: Prototipo per la rilevazione della presenza. Possiamo notare i due microfoni (a destra e sinistra) e il sensore VCNL4200 (in alto).

Capitolo 2

Progettazione

2.1 Hardware

Valutando le specifiche del progetto e la sua finalità, l'hardware necessario per il prototipo è il seguente:

- 1x Microcontrollore ESP32-WROOM-32E
- 1x Sensore di distanza e luminosità ambientale Vishay VCNL4200
- 2x Microfono digitale MEMS Sipeed MSM261S4030H0

2.1.1 ESP32-WROOM-32E

Figura 2.1: Microcontrollore ESP32-WROOM-32E

L'**ESP32-WROOM-32E** è un microcontrollore prodotto da Espressif System, divenuto popolare per la versatilità, le prestazioni, la connettività bluetooh e WiFi

integrati, oltre ad un prezzo contenuto. Viene utilizzato molto in ambiti come automazione, domotica e Internet of Things (IoT).
Questo microcontrollore presenta le seguenti principali caratteristiche:

- Oscillatore a quarzo per la gestione del timing del microcontrollore a 40 MHz
- integrato ESP32-D0WD-V3
- microprocessore Xtensa dual-core 32 bit con architettura LX6
- connettore micro-usb per la comunicazione via seriale
- pin di gestione delle comunicazioni
- pin analigici (con i rispettivi ADC)
- antenna Wi-Fi integrata
- bluetooth low energy (BLE)

e più nello specifico abbiamo

- 448KB di ROM per booting e funzioni del processore
- 520 KB di SRAM per dati ed istruzioni
- interfaccia UART
- Watchdog Timer
- interfaccia I2C
- interfaccia I2S
- 16x pin ADC a 12 bit
- 32x pin GPIO, di cui 4 solo input.
- 28x pin per PWM

L'ambiente scelto per lo sviluppo del firmware è Arduino IDE, sfruttando le librerie messe a disposizione da Espressif. Tra le varie funzionalità, include al proprio interno un editor di testo per la scrittura del codice, una sezione per il debug, un monitor seriale per vedere i dati scambiati dalla scheda, una gestione semplificata delle librerie e apposite opzioni per compilazione e caricamento del firmware.

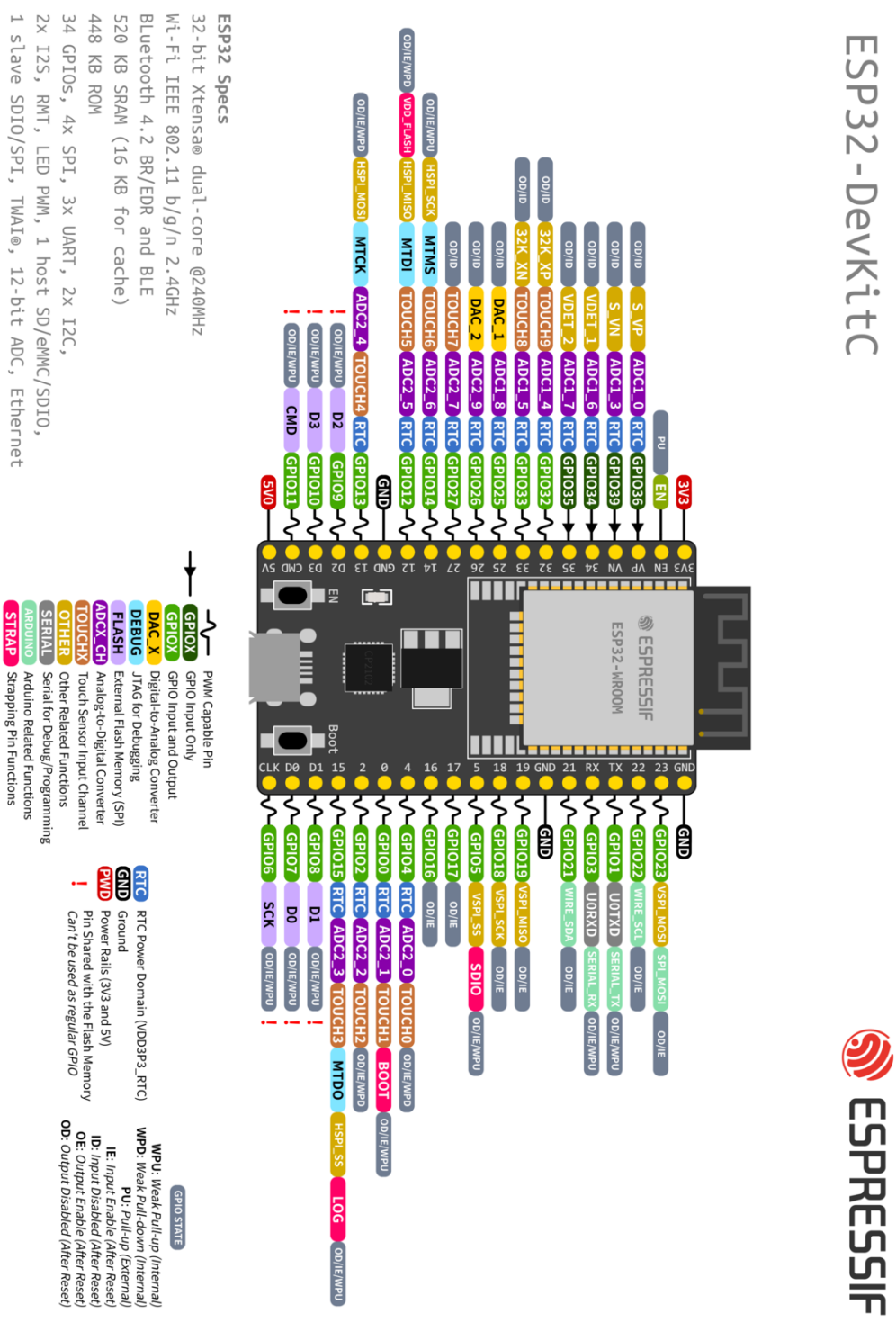


Figura 2.2: Mappa pin ESP32

2.1.2 VCNL4200

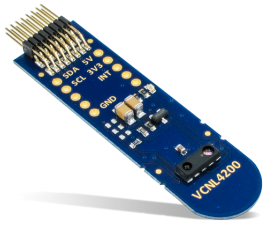
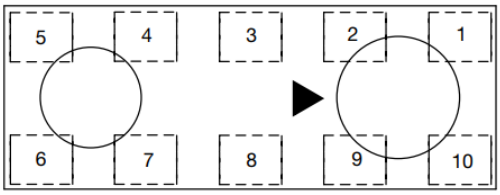


Figura 2.3: VCNL4200

Il **VCNL4200** è un sensore con interfaccia I2C (vedi sezione 2.2.1.1) prodotto da Vishay. Integra in un unico componente un sensore di prossimità per lunghe distanze ed un sensore di luce ambientale capaci di lavorare in parallelo. Grazie alla sua risoluzione impostabile da 12 o 16 bit per la misura di prossimità, offre misure molto accurate ed affidabili. È in grado di operare con grandi variazioni di temperatura ambientale senza risentirne sulle misure (da -40°C a +85°C).
Il sensore presenta 10 pin, definiti come in fig. 2.4, dove i pin 9 e 10 sono i pin dedicati alla comunicazione con l'interfaccia I2C, gestendone i registri e le richieste del master. I suoi registri sono 15, costituiti da 2 byte (un byte meno significativo ed un byte più significativo), 8 sono di lettura/scrittura, i restanti 7 sono disponibili per la sola lettura. Il range di operatività del sensore è compreso tra 2.5V e 3.6V. Tuttavia, per



Top View

1	GND	6	LED+
2	LED_CATHODE	7	NC
3	V_{DD}	8	INT
4	NC	9	SDAT
5	LED-	10	SCLK

Figura 2.4: Definizione pin VCNL4200

la misura della distanza viene utilizzato un led ad infrarossi, che necessita 5V per il funzionamento, quindi il sensore ha due alimentazioni separate.
Le caratteristiche di questo sensore per la misura di prosimità sono:

- immunità alla luce visibile

- sistema intelligente di cancellazione della luce ambientale
- circuito adattato per ridurre il tempo della misura
- Prossimità misurabile fino a 1 metro e mezzo.

Mentre per la misura della luminosità ambientale:

- immunità alla luce fluorescente
- spettro luminoso vicino a quello dell'occhio umano
- intensità luminosa massima misurabile selezionabile (197 / 393 / 786 / 1573)lux con una sensibilità di 0.003 lux/step

Alla necessità può operare misurando solo con la chiamata di interrupt programmati. Data la sua efficenza e l'integrazione di due sensori in un unico integrato, il suo utilizzo lo troviamo in dispositivi della gestione delle collisioni, in sistemi della gestione di parcheggi per identificare i posti liberi e nella gestione di luci negli uffici, corridoi ed edifici pubblici.

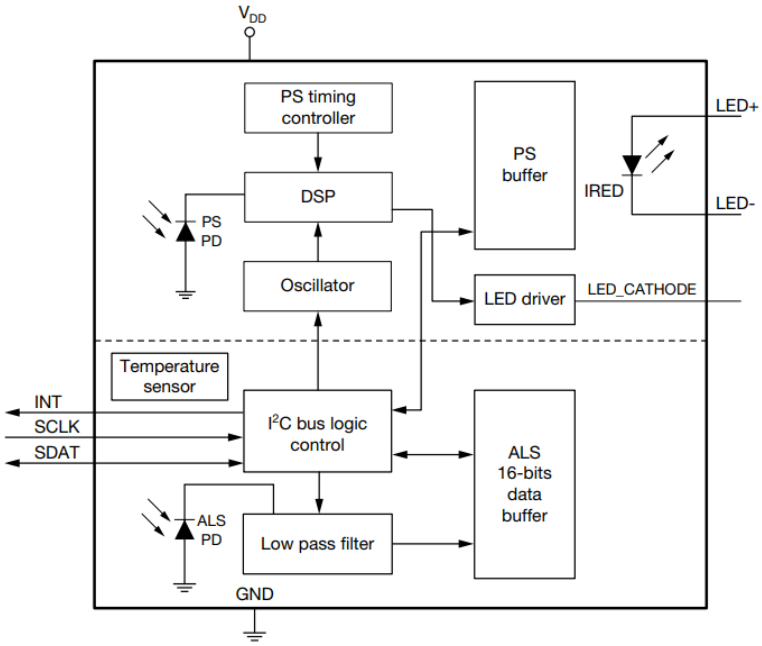


Figura 2.5: Diagramma a blocchi del VCNL4200

2.1.3 Microfono digitale MEMS I2S Sipeed MSM261S4030H0

Figura 2.6: Sipeed MSM261S4030H0

Il microfono **Sipeed MSM261S4030H0** è un microfono digitale MEMS utilizzabile con l'interfaccia I2S (vedi sezione 2.2.1.2). L'acronimo **MEMS** sta per *"Micro Electro Mechanical System"*, cioè sistemi costruiti da elementi meccanici ed elettromeccanici miniaturizzati, e quando si parla di microfoni MEMS si fa riferimento a microfoni che utilizzano le tecnologie MEMS per la loro costruzione. Questi microfoni sono incisi su un wafer di semiconduttore (silicio), ed una membrana mobile sensibile alla pressione chiamata *diaframma* è incisa dietro una piastra perforata fissa, che insieme formano un condensatore. Questi microfoni sono dotati di un componente chiamato **ASIC** (*Application Specific Integrated Circuit*), un microchip progettato per posizionare una carica fissa tra membrana e piastra fissa. Le onde sonore entrano nel microfono attraverso una copertura meccanica e passano attraverso la piastra perforata prima di raggiungere il diaframma. Le onde sonore provocano una pressione sonora variabile su di esso ed una differenza di pressione tra la sua parte posteriore ed anteriore, facendo muovere il diaframma in concomitanza con le onde sonore, variandone la capacità del condensatore. Grazie alla carica fornita dall'ASIC, qualsiasi movimento produce una variazione inversamente proporzionale della tensione, che necessiterà di amplificazione per essere utile come segnale audio. Questa amplificazione è svolta da un circuito integrato separato, usando infine un ADC interno al sensore, che comporta il non utilizzo degli ADC interni al microcontrollore perchè i dati gli arrivano già convertiti.

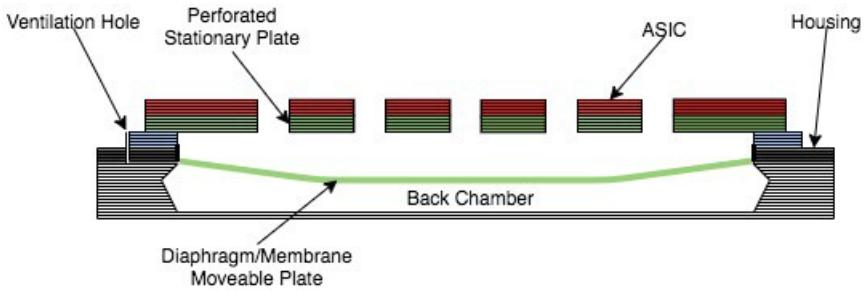


Figura 2.7: Struttura fisica microfono MEMS

Questo specifico componente è stato scelto per la sua efficienza in ambito di dispositivi domestici intelligenti, quindi adatto agli ambienti chiusi.
Le principali caratteristiche di questo microfono sono:

- alimentazione a 3.3 V
- range di temperatura di lavoro tra -30°C e +85°C
- sensibilità di -26 dB
- SNR di 57 dB
- frequenza di clock impostabile (1.0-4.0 MHz modalità standard, 150-800 kHz modalità low power)
- basso consumo

Presenta 6 pin di collegamento:

- G: GND
- V: alimentazione
- WS: *Word Select*: per la sincronizzazione dei dati in uscita.
- LR: *Left/right channel*, per la selezione del canale d'utilizzo.
- CK: *clock*, per il clock di funzionamento. È dato dal microcontrollore.
- DA: per la trasmissione dei dati dal microfono.

Molto importante per questo componente è il pin *LR*, che serve per settare l'utilizzo del microfono come canale di destra o sinistra, distinguendo la provenienza dei rumori. Per l'utilizzo come canale di destra, è necessario collegare il pin a 3.3V (cioè alla tensione di alimentazione), mentre per l'utilizzo come canale di sinistra è necessario collegare il pin a GND. La gestione dei dati inviati dai due microfoni al microcontrollore è affidata al firmware.

2.2 Firmware

2.2.1 Interfacce utilizzate

Le interfacce principali utilizzate per la comunicazione con il VCNL4200 e con i due microfoni MEMS sono **I2C** e **I2S**.

2.2.1.1 Protocollo I2C

I2C (abbreviazione di Inter Integrated Circuit) è un protocollo di comunicazione seriale con l'utilizzo di due linee utilizzato tra circuiti integrati, creato da Philips nel 1982. Il sistema di comunicazione è composto da uno o più *controller* e da uno o più *target*, e richiede due linee seriali di comunicazione: **SDA** (Serial Data) per lo scambio di dati tra controller e targer, **SCL** (Serial Clock) per il segnale di clock, controllato dal controller. Queste linee seriali hanno bisogno di *resistenze di pull-up*, utili per mantenere il potenziale alto sulla linea in assenza di trasmissione.
Con I2C, i dati sono trasmessi come *messaggio*, diviso in diversi *frame*. Il messaggio è strutturato come in fig. 2.8:

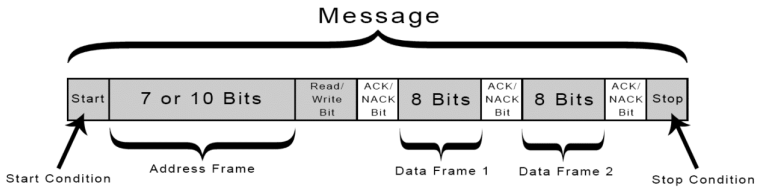


Figura 2.8: Divisione frame del messaggio utilizzato con I2C.

- **Start Condition**: la linea SDA tramuta dal livello alto al livello basso di tensione prima che la linea SCL tramuta dal livello alto al livello basso di tensione.
- **Stop Condition**: la linea SDA tramuta dal livello basso al livello alto di tensione dopo che la linea SCL tramuta dal livello basso al livello alto di tensione.
- **Address Frame**: frame dedicato a 7 o 10 bit che identifica il target col quale il controller vuole parlare.
- **Read/Write Bit**: bit che specifica se il controller sta inviando dati al target (bit è 0) o se sta chiedendo dati al target (bit è 1). È il primo frame che viene inviato dopo il bit di start.

- **ACK/NACK bit**: ogni frame del messaggio è seguito da un bit di acknowledge/no-acknowledge. Se un frame è stato ricevuto correttamente, un bit di acknowledge è ritornato al mittente dal destinatario del messaggio.

I principali vantaggi del suo utilizzo sono:

- utilizzo di solo due linee
- supporto di multipli controller e multipli target
- bit di acknowledge che da conferma della corretta ricezione

mentre gli svantaggi:

- trasferimento dati più lento rispetto ad altri protocolli
- i frame dei dati sono limitati alla dimensione di 8 bit

2.2.1.2 Protocollo I2S

I2S (abbreviazione di Inter-IC Sound, o Integrated Interchip Sound) è uno standard di bus seriale usato per connettere insieme dispositivi audio digitali introdotto da Philips per la prima volta nel 1986. È generalmente usato per comunicare audio di tipo PCM (Pulse-Code modulation, metodo usato per rappresentare digitalmente campioni di segnali analogici) tra circuiti integrati, ed è anch'esso come I2C un protocollo a due canali. ESP32 a livello hardware è dotato di due processori interamente dedicati a I2S, settabili tramite firmware.
Le caratteristiche principali di I2S sono descritte dal funzionamento dei suoi canali: **Serial Data (SD)**, **Word Select (WS)** e **Clock (CK)**.

- **Serial Data**: il bit più significativo è il primo dato dei valori digitali che viene trasmesso. Il trasmettitore ed il ricevitore non hanno bisogno di bit dedicati per capire quello che verrà inviato e ricevuto; il trasmettitore invia quello che ha, ed il ricevitore prende quello che può utilizzare. I nuovi bit in arrivo possono interrompersi sia nel fronte di salita che nel fronte di discesa del clock, ma devono essere obbligatoriamente inviati nel fronte di salita. Una particolarità di questo protocollo è che il clock viene sempre utilizzato, non ci sono periodi di clock inutilizzati tra i byte in trasmissione e ricezione, quindi il bit meno significativo alla fine di un byte inviato è immediatamente seguito dal bit più significativo del nuovo byte in arrivo.
- **Word Select**: È il canale di clock di I2S che gestisce il canale di provenienza delle informazioni. Funziona con i livelli logici HIGH e LOW. Se il livello è HIGH, allora il byte in uscita dal microfono fa parte delle informazioni del canale di sinistra, se il livello è LOW allora fa parte delle informazioni del canale di destra. Per facilitarne la gestione sia da parte del trasmettitore che del ricevente, il segnale di WS cambia un periodo di clock prima del completamento del trasferimento del byte.

- **Clock**: funziona continuamente, senza mai fermarsi. Un vantaggio di questo protocollo è quello che non specifica una velocità massima di trasmissione dati, ma bigosna regolarla in base all'hardware utilizzato per avere la trasmissione più chiara possibile.

L'illustrazione del funzionamento è in figura 2.9.

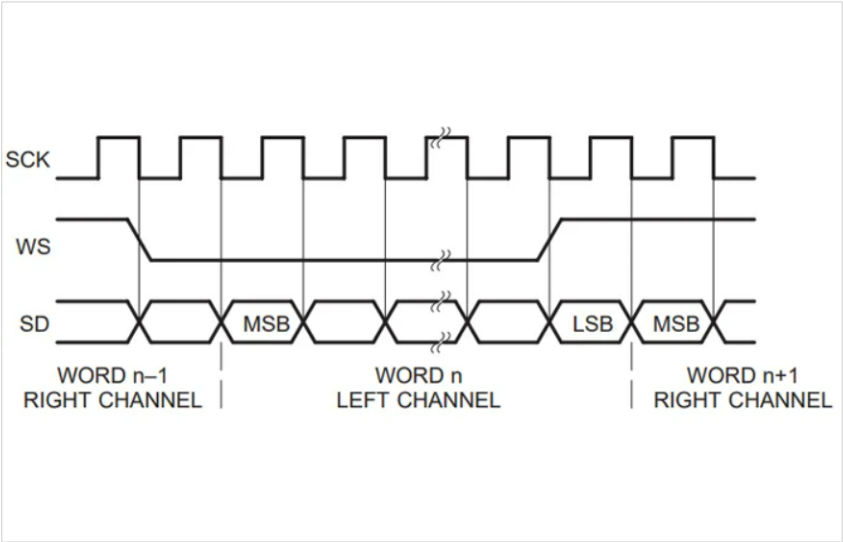


Figura 2.9: Schema di trasmissione I2S

Per questo protocollo è molto importante la velocità di trasmissione, e il trasferimento di audio richiede molta più banda rispetto alle trasmissioni con I2C. Un'altra differenza rispetto ad I2C è che I2S non richede le resistenze di pull-up, perchè la sincronizzazione della trasmissione è gestita dalla linea WS.

2.2.2 Codice

Il codice creato per il prototipo è stato scritto col linguaggio C utilizzando Arduino IDE. Di seguito le sezioni più rilevanti del codice.

2.2.2.1 Librerie importate

Le *librerie* sono un insieme di funzioni, procedure e/o classi che possono essere utilizzate da un programma per svolgere determinate operazioni senza dover scrivere il codice da zero. In C, le librerie usate sono comprese tra '< >' (oppure " ") e hanno estensione .h (*header*), precedute dalla parola chiave #**include**:

```
1 #include <Arduino.h>
2 #include <DHT.h>
3 #include <time.h>
4 #include <esp_task_wdt.h>
5 #include <driver/i2s.h>
6 #include <driver/dac.h>
7 #include <soc/i2s_reg.h>
8 #include <Wire.h>
9 #include <Vishay_VCNL4200.h>
```

- **Arduino.h**: libreria base di Arduino usata anche per ESP32, serve per l'utilizzo di macro e metodi base.
- **DHT.h**: nel progetto è stato implementato il firmware anche per il sensore DHT11, che è un sensore di temperatura ed umidità ambientale, munito di libreria propria. Il suo utilizzo non è necessario al nostro scopo, quindi la sua funzionalità è stata esclusa dal progetto, ma non eliminata, facilitandone l'eventuale reintegrazione.
- **time.h**: utilizzata per conteggio dei tempi, utile in fase di debug.
- **esp_task_wdt.h**: libreria dedicata al *watchdog timer*, che è un timer (elettronico o software) usato per l'identificazione e il recupero da malfuznionamenti del sistema.
- **driver/i2s.h**: libreria utilizzata per la gestione dell'interfaccia I2S.
- **driver/dac.h**: libreria utilizzata per l'eventuale gestione approfindita dei DAC del microcontrollore.
- **soc/i2s_reg.h**: libreria per la gestione più accurata dei registri di I2S.
- **Wire.h**: libreria importata per l'utilizzo di I2C.
- **Vishay_VCNL4200.h**: libreria importata fondalmentalmente per una prima inizializzazione generica del VCNL4200.

Per la gestione dei protocolli, sono stati creati dei **metodi** (o funzioni), che sono un costrutto sintattico di un determinato linguaggio di programmazione che permette di raggruppare, all'interno di un programma, una sequenza di istruzioni in un unico blocco, isolando una specifica operazione, azione o elaborazione sui dati del programma stesso in modo tale che, a partire da determinati input, restituisca (se necessari) determinati output. I metodi principali creati per semplificarne la gestione sono **write_register** per gestire la comunicazione con il VCNL4200, **i2s_install** e **i2s_setpin** per configurare l'interfaccia I2S.

2.2.2.2 Metodo write_register

```
1 void write_register(int device_addr, int register_num, int
    low_byte, int high_byte){
2
3   Wire.beginTransmission(device_addr);
4
5   Wire.write(register_num);
6   Wire.write((uint8_t)(high_byte & 0xFF));
7   Wire.write((uint8_t)((low_byte >> 8) & 0xFF));
8
9   Wire.endTransmission(false);
10 }
```

Questo metodo è dedicato al VCNL4200, dove mediante l'utilizzo della libreria Wire, lo gestiamo con la comunicazione I2C. Il metodo necessita di quattro parametri di tipo **int**:

- *device_addr*: è l'indirizzo del target con cui andiamo a comunicare (nel nostro caso è 0x51).
- *register_num*: il numero del registro che andiamo a scrivere o ad interrogare.
- *low_byte*: byte meno significativo da inviare al sensore.
- *high_byte*: byte più significativo da inviare al sensore.

Wire.beginTransmission(device_addr) apre la comunicazione con il sensore all'indirizzo device_addr. **Wire.write(register_num)** punta all'indirizzo passato come argomento, che sarà il destinatario delle operazioni di scrittura.
Wire.write((uint8_t)(high_byte & 0xFF)) e **Wire.write((uint8_t)((low_byte » 8) & 0xFF))** sono rispettivamente in ordine l'invio del byte più significativo e meno significativo al registro register_num. I byte low_byte e high_byte arrivano come parametri in formato 16 bit, ma visto che le la parte alta e la parte bassa dei registri del VCNL4200 sono a 8 bit, hanno bisogno di un *casting* a 8 bit, dopo essere stati controllati con l'operazione di AND esclusivo **&** (XOR) con 0xFF. Il byte meno

significativo, ha bisogno anche di uno *shift logico* di 8 bit verso destra, così il valore andrà a posizionarsi precisamente nella parte bassa del registro del VCNL4200, come richiesto dall'architettura del sensore.
Dopo la scrittura dei registri del sensore si chiude la comunicazione, e per farlo si utilizza **Wire.endTransmission(false)**, con argomento *false* che serve per inviare un messaggio di restart dopo il termine della trasmissione, non liberando il bus e permettendo multiple scritture del master, a discapito di azioni di eventuali altri master collegati.
Questo metodo non restituisce valori, quindi è di tipo void.

2.2.2.3 Metodo I2S_install

```
1  void i2s_install(){
2    //Set up I2S processor configuration
3    const i2s_config_t i2s_config = {
4      .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
5      .sample_rate = 44100, //Sample rate in kHz
6      .bits_per_sample = i2s_bits_per_sample_t(16),
7      //.channel_format = I2S_CHANNEL_FMT_ONLY_RIGHT,
8      //.channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
9      .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT, //LEFT -
         GND, RIGHT - 3v3
10     .communication_format = I2S_COMM_FORMAT_I2S,
11     .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
12     .dma_buf_count = 4,
13     .dma_buf_len = bufferLen,
14     .use_apll = false,
15     .tx_desc_auto_clear = false,
16     .fixed_mclk = 0
17   };
18   i2s_driver_install(I2S_PORT,&i2s_config,0,NULL);
19 }
```

Questo metodo serve per installare e configurare l'interfaccia I2S. Consiste nella definizione del valore delle costanti di i2s_config_t appartenenti alla rispettiva libreria:

- **.mode**: stabilisce la modalità di funzionamento dell'I2S, che può essere controller (I2S_MODE_MASTER) o target (I2S_MODE_SLAVE), e se deve fare da ricevitore (I2S_MODE_RX), trasmettitore (I2S_MODE_TX), o entrambi.
- **.sample_rate**: rappresenta la frequenza di campionamento dell'audio in Hz.
- **.bits_per_sample**: indica il numero di bit utilizzati per la rappresentazione di ogni campione audio.

- **.channel_format**: specifica il formato dei dati per i canali audio. Può essere solo sinistro (I2S_CHANNEL_FMT_ONLY_LEFT), solo destro (I2S_CHANNEL_FMT_ONLY_RIGHT), o sinistro e destro (I2S_CHANNEL_FMT_RIGHT_LEFT).
- **.communication_format**: specifica il formato di comunicazione dei dati. Può essere I2S_COMM_FORMAT_I2S (formato I2S) o I2S_COMM_FORMAT_PCM (formato PCM, Pulse Code Mdulation, dove l'ampiezza del segnale analogico viene campionata ad intervalli regolari per poi rappresentarlo in forma digitale).
- **.intr_alloc_flags**: indica il livello di priorità delle interruzioni (ESP_INTR_FLAG_LEVEL1 o ESP_INTR_FLAG_LEVEL2), dove la priorità più alta appartiene ad ESP_INTR_FLAG_LEVEL1.
- **.dma_buf_count**: numero di buffer DMA (Direct memory Access) utilizzati per gestire il flusso audio. Il DMA è una tecnica che permette alle periferiche di accedere direttamente alla memoria del sistema senza coinvolgere la CPU, utile quando serve un flusso continuo e rapido di dati.
- **.dma_buf_len**: indica la lunghezza dei buffer (in byte) dove vengono raccolti i campioni. Nel progetto è stata inserita una lunghezza di 64.
- **.use_apll**: riferimento alla configurazione di I2S per l'utilizzo o meno dell'A-PLL, che è un componente hardware dedicato per la generazione di un clock ad alta precisione.
- **.tx_desc_auto_clear**: riferimento alla configurazione del controller I2S per abilitare o disabilitare la pulizia automatica dei descrittori di trasmissione, che sono strutture dati utilizzate da I2S per la gestione della trasmissione di ogni blocco di dati. Se il valore è *true*, allora la pulizia è automatica, se *false* non è automatica.
- **.fixed_mclk**: riferimento alla configurazione di I2S per l'uso di un clock controller fisso (MCLK). Questa funzione non viene utilizzata quindi mettiamo il valore 0.

La configurazione viene completata dalla funzione standard della libreria I2S **i2s_driver_install(I2S_PORT,&i2s_config,0,NULL)**, che applica i parametri stabiliti. *I2S_PORT* indica il processore scelto che utilizzerà I2S.

2.2.2.4 Metodo I2S_setpin

```
1 void i2s_setpin(){
2   //Set I2S pin configuration
3   const i2s_pin_config_t pin_config = {
4     .bck_io_num = I2S_SCK,
5     .ws_io_num = I2S_WS,
6     .data_out_num = -1,
7     .data_in_num = I2S_SD
8   };
9   i2s_set_pin(I2S_PORT, &pin_config);
10 }
```

Questo metodo serve per settare i pin che userà I2S per lavorare. Il numero del pin non è unico, si possono scegliere arbitrariamente. I pin da settare sono:

- **.bck_io_num**: pin di clock.
- **.ws_io_num**: pin per identificare se il dato arriva dal canale di destra o dal canale di sinistra.
- **.data_out_num**: output per mandare dati audio con I2S. Non necessitiamo di questa operazione, quindi il valore assegnato è -1.
- **.data_in_num**: pin per i dati audio in ingresso.

Per settare i pin scelti alla fine del metodo viene richiamata la funzione **I2S_set_pin(I2S_PORT, &pin_config)**, della libreria I2S.

Per scelta progettuale, la gestione del sensore VCNL4200 e dei microfoni sono separate, sfruttando la caratteristica multicore di ESP32. La gestione della prossimità e luminosità ambientale data dal VCNL4200 è gestita sulla routine principale preimpostata di ESP32 chiamata **loop**, mentre il rumore è gestito da un processo parallelo detto *thread* (o *task*), creato dal metodo **xTaskCreatePinnedToCore** chiamato **plot_i2s_microphone**.

2.2.2.5 Routine loop

```
1 void loop() {
2
3   esp_task_wdt_reset(); //reset watchdog
4   //t_h_DHT11();
5
6   int reg = 0x8; //Proximity Sensor (PS) output
7
```

```
8   //Write to regsiter
9   Wire.beginTransmission(SENSOR_ADDR);
10  Wire.write(reg);
11  Wire.endTransmission(false);
12
13  //Read data
14  uint16_t data[2] = {0, 0};  //The sensor provides the
       output over 2 bytes.
15  Wire.requestFrom(SENSOR_ADDR, 2);
16  data[0] = Wire.read();
17  data[1] = Wire.read();
18  int value1 = (int)(data[1] << 8) + (int)(data[0]); //
       Combines bytes
19
20  reg = 0x09; //Ambient Light Sensor (ALS) output
21
22  //Write to regsiter
23  Wire.beginTransmission(SENSOR_ADDR);
24  Wire.write(reg);
25  Wire.endTransmission(false);
26
27  Wire.requestFrom(SENSOR_ADDR, 2);
28  data[0] = Wire.read();
29  data[1] = Wire.read();
30  int value2 = (int)(data[1] << 8) + (int)(data[0]); //
       Combines bytes
31
32  Serial.print("Proximity:␣");
33  Serial.println(value1);
34  Serial.print("Lux:␣");
35  Serial.println(value2*sensitivity);
36  Serial.println("␣");
37
38  delay(500);
39
40 }
```

Il primo blocco di operazioni (da riga 6 a riga 18) tratta della misura di prossimità. Per ordinare al sensore di catturare la misura di prossimità, è necessario inviargli il comando adatto al compito, che da datasheet è 0x08, valore che prenderà la variabile *reg* da inviare al sensore come parametro all'interno del metodo Wire.write(). Per fare ciò, si apre la comunicazione con Wire.beginTransmission(device_addr), si invia il comando tramite Wire.write(reg) e si chiude la trasmissione per l'invio di comandi

con Wire.endTransmission(false).
Se la trasmissione è andata a buon fine, la misura sarà pronta all'interno del sensore dopo il tempo necessario di calcolo. Il VCNL4200 fornisce le misure su due byte da elaborare alla ricezione, e per avere correttamente questa misura utilizziamo un array di due celle con formato 16 bit chiamato *data*, inizializzato a valori 0. Per richiedere i byte necessari, si utilizza **Wire.requestFrom(SENSOR_ADDR, 2)** (dove il secondo parametro sta per il numero di byte richiesti) che restituirà i byte presenti all'interno del sensore nella sua memoria dopo la misura. I due byte vengono salvati all'interno dell'array *data* nell'ordine di arrivo dalla trasmissione. Per leggere i byte si utilizza il metodo **Wire.read()** che andrà a leggere il primo byte (il meno significativo) arrivato, assegnandolo alla prima posizione (data[0]) dell'array. Per leggere il secondo byte (il più significativo) si compie la stessa operazione, assegnandolo alla seconda posizione (data[1]) dell'array. Questa operazione non da errori di lettura con eventuali altre misure, perchè tra le due letture dei byte non ci sono altre operazioni di misura del sensore.
I byte letti devono essere elaborati per fornire una misura corretta. Questi subiscono un casting in int ed il byte più significativo uno shift logico verso sinistra di 8 bit, per posizionare il byte nella posizione corretta. I byte vengono sommati ed assegnati alla variabile *value1*, che conterrà il valore corretto di prossimità.
Il secondo blocco di operazioni (da riga 23 a riga 34) gestisce la misura di luminosità ambientale. La logica seguita e le operazioni sono identiche, differenziano solo nel comando da assegnare alla variabile reg che ora è 0x09 e nel calcolo della misura, perchè dopo la somma dei byte il risultato va moltiplicato per la misura di sensitività contenuta nella variabile *sensitivity*.
La riga numero 3 di codice contenente **esp_task_wdt_reset()** è necessaria in quanto serve a resettare il timer del watchdog, per evitare un blocco involontario dell'esecuzione della routine. Essendo un timer, se non viene resettato durante l'esecuzione ciclica di loop, il conteggio del tempo avanza fino a raggiungere la soglia di trigger. Subito sotto a commento c'è **t_h_DHT11()**, routine per il sensore DHT11 implementata nel firmware ma non inclusa nell'esecuzione.
Tutte le stampe seriali sono utilizzate solamente in fase di debug.

2.2.2.6 Metodo xTaskCreatePinnedToCore

```
1 xTaskCreatePinnedToCore(
2   plot_i2s_microphone, /* Function to implement the task */
3   "plot_i2s_microphoneName", /* Name of the task */
4   10000,  /* Stack size in words */
5   NULL,  /* Task input parameter */
6   0,  /* Priority of the task */
7   &plot_i2s_microphoneName,  /* Task handle. */
8   1   /* Core where the task should run */
9 );
```

Questo metodo appartiene alla libreria **FreeRTOS**, una libreria di sistema operativo in tempo reale utilizzata su microcontrolori e microprocessori embedded per gestire il *multiprocessing*. Viene richiamato nella routine di **setup** che viene eseguita prima della routine di loop. I suoi parametri sono:

- *plot_i2s_microphone*: nome della funzione dove implementare il task.
- *"plot_i2s_microphoneName"*: nome del task.
- *1000*: numero di byte di memoria dedicati al task.
- *NULL*: parametri da dare in input al task.
- *0*: priorità del task.
- *&plot_i2s_microphoneName*: puntatore al task.
- *1*: indica il processore dove il task viene eseguito.

2.2.2.7 Thread plot_i2s_microphone

```
1  void plot_i2s_microphone(void * parameters){
2
3    for(;;){
4
5      esp_task_wdt_reset(); //reset watchdog
6
7      //Get I2S data and place in data buffer
8      size_t bytesIn = 0;
9      esp_err_t result = i2s_read(I2S_PORT, &sBuffer,
         bufferLen, &bytesIn, portMAX_DELAY);
10     float mean = 0;
11     int16_t samples_read;
12
13     if(result == ESP_OK){
14       samples_read = bytesIn / 8;
15       if(samples_read > 0){
16         for(int16_t i = 0; i < samples_read; i++){
17           if(i%2 == 0){ //EVEN LEFT, ODD RIGHT
18             left +=  abs(sBuffer[i]);
19             mean += abs(sBuffer[i]);
20           }else{
21             right += abs(sBuffer[i]);
22             mean += abs(sBuffer[i]);
23           }
```

```
24          }
25          //Average of the data captured
26          mean /= samples_read;
27        }
28      }
29
30      if(millis() - last >= PRINT_TIME){
31        if(left > right){
32          Serial.print("Il rumore viene da sinistra: ");
33          Serial.print(left);
34          Serial.print(", a destra invece ");
35          Serial.println(right);
36          Serial.print("Media: ");
37          Serial.println(mean);
38        }else{
39          Serial.print("Il rumore viene da destra: ");
40          Serial.print(right);
41          Serial.print(", a sinistra invece ");
42          Serial.println(left);
43          Serial.print("Media: ");
44          Serial.println(mean);
45        }
46        if(mean > BG_LEVEL){
47          Serial.println("Qualcuno parla");
48        }else{
49          Serial.println("Rumore di sottofondo");
50        }
51
52        left = 0;
53        right = 0;
54        mean = 0;
55        last = millis();
56
57      }
58
59    } //for graph
60
61    vTaskDelete(NULL);  //Deletes the thread when finished, it
          will never happen
62
63  }
```

L'unico parametro passato al thread è *void * parameters*, che punta ai parametri assegnati alla funzione xTaskCreatePinnedToCore. Non riceve alcun parametro avendo assegnato *NULL* al rispettivo campo, ma per una corretta sintassi la sua scrittura è necessaria per evitare errori in fase di compliazione.
Per catturare i campioni audio dai microfoni viene usato il metodo della libreria I2S **I2S_read()** che restituisce un valore di tipo *esp_err_t* assegnato alla variabile *result*. I parametri di questo metodo sono:

- *I2S_port*: processore utilizzato da I2S.
- &*sBuffer*: puntatore al buffer dove vengono memorizzati i dati.
- *bufferLen*: lunghezza del buffer, che corrisponde al numero di campioni da leggere.
- &*bytesIn*: variabile per indicare il numero di byte acquisiti. Inizializzata a 0 per permettere l'incremento in fase di esecuzione del metodo.
- *portMAX_DELAY*: timer che pone un limite al tempo di lettura dei dati. Se viene superato può dare un messaggio d'errore con il numero di byte effettivamente ricevuti. É un parametro opzionale.

Se non vengono riscontrati problemi in acquisizione, la variabile result assume valore *ESP_OK*, dove controllandola con un if, si andranno ad eseguire o meno le operazioni di gestione dell'audio. Se *result* ha un valore differente da ESP_OK, non verrà eseguita alcuna operazione. Per verificare che ci siano effettivamente campioni audio disponibili:

$$samples\_read = bytesIn/8$$

dove se il risultato dell'operazione è maggiore di 0, si suddividono i campioni per canale scorrendo l'array di buffer. La libreria I2S in presenza di due canali audio, suddivide i dati nel buffer in una maniera precisa: nelle posizioni pari i campioni provenienti dal microfono del canale di sinistra, nelle posizioni dispari i campioni provenienti dal canale di destra. In questa maniera scorrendo l'array possiamo suddividere il rumore secondo la seguente logica: se l'indice i dell'array di buffer è pari, allora incremento il contatore *left* con il dato contenuto nella cella con indice i, stessa cosa con il contatore *right* solo che l'indice i deve essere dispari. In entrambi i casi è incrementata la variabile *mean* per calcolare la media dei valori catturati alla fine del ciclo for. Importante che tutte le somme siano fatte con valori numerici positivi, perchè in input dai microfoni possiamo anche ricevere valori negativi, che andrebbero a falsare i nostri calcoli sul rumore. Questa operazione si compie utilizzando il metodo della libreria Arduino.h **abs()**, con argomento il numero da rendere positivo. Vengono passati nel metodo tutti i valori dell'array di buffer, ma i valori positivi rimangono invariati, e cambiano di segno solo i valori negativi.

La provenienza del rumore si verifica confrontando la grandezza dei due contatori *left* e *right* con la condizione **if(left $>$ right)**, dove se questa è vera allora il rumore proviene da sinistra, se è falsa allora il rumore proviene da destra. Infine verificando che la media sia sopra o sotto una certa soglia di rumore (BG_LEVEL) possiamo distinguere se qualcuno è presente nelle vicinanze del sistema e sta per esempio parlando o se è presente solo rumore di sottofondo.
Tutte le stampe seriali sono utilizzate solamente in fase di debug.

Capitolo 3

Testing

3.1 Sensore di prossimità

Per valutare l'affidabilità della misura di prossimità del VCNL4200, sono stati raccolti i dati necessari in fase di test, tramite l'utilizzo di un firmware creato per lo scopo utilizzante una piattaforma esterna per IoT. I gruppi di misure sono stati presi da tre posizioni differenti con distanza tra sensore e persona simile:

- Primo gruppo: posizione sensore laterale -45° da persona, distanza circa 80cm, inclinazione verticale approssimativa -20°.
- Secondo gruppo: posizione sensore centrale 0° da persona, distanza circa 80cm, inclinazione verticale approssimativa -20°.
- Terzo gruppo: posizione sensore laterale +45° da persona, distanza circa 80 cm, inclinazione verticale approssimativa -20°.

Le informazioni sono state raccolte e raggruppate con tre grafici, uno per ogni gruppo di misura. Le misure considerate sono 350 per ogni posizione, con la cattura di un dato ogni 20 secondi per un totale di 6 ore di test.
Il test è stato eseguito simulando un normale utilizzo del sistema, quindi stando seduti alla scrivania ed alzandosi al bisogno, annotando l'azione di alzarsi quando viene fatta e quando viene conclusa. Eventuali misure false (come in fase di sistemazione del sensore nella posizione corretta) non sono presenti, perché sono state sostituite da misure catturate durante l'utilizzo corretto del sistema
Nei grafici ricavati, nell'asse delle ascisse si trova il numero delle misure, nell'asse delle ordinate si trova il valore della misura di prossimità ricavata. I valori sono poi collegati tramite una linea blu per facilitarne la comprensione.

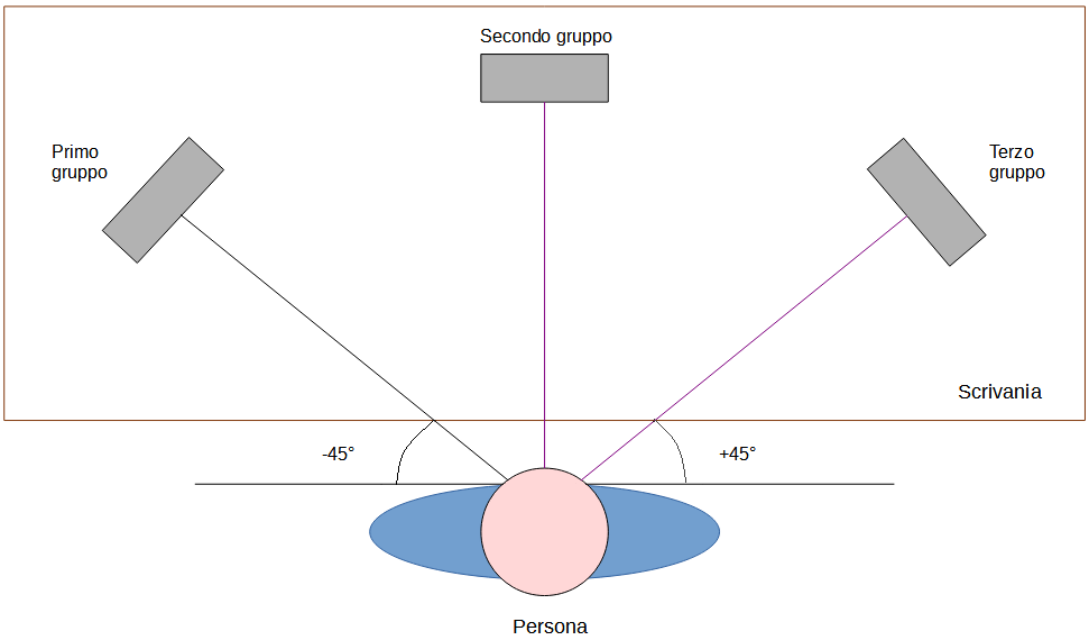


Figura 3.1: Vista schematica dall'alto del posizionamento. In figura presenti le tre posizioni del sensore.

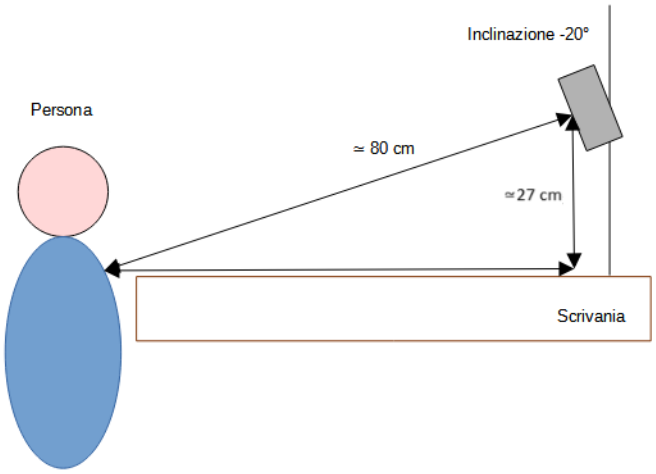


Figura 3.2: Vista schematica dal lato del posizionamento. Si noti l'inclinazione negativa approssimativa di 20° ad una distanza di circa 80 cm dalla persona.

Di seguito i grafici delle misure di prossimità ricavate col VCNL4200 nelle tre posizioni elencate.

Primo gruppo

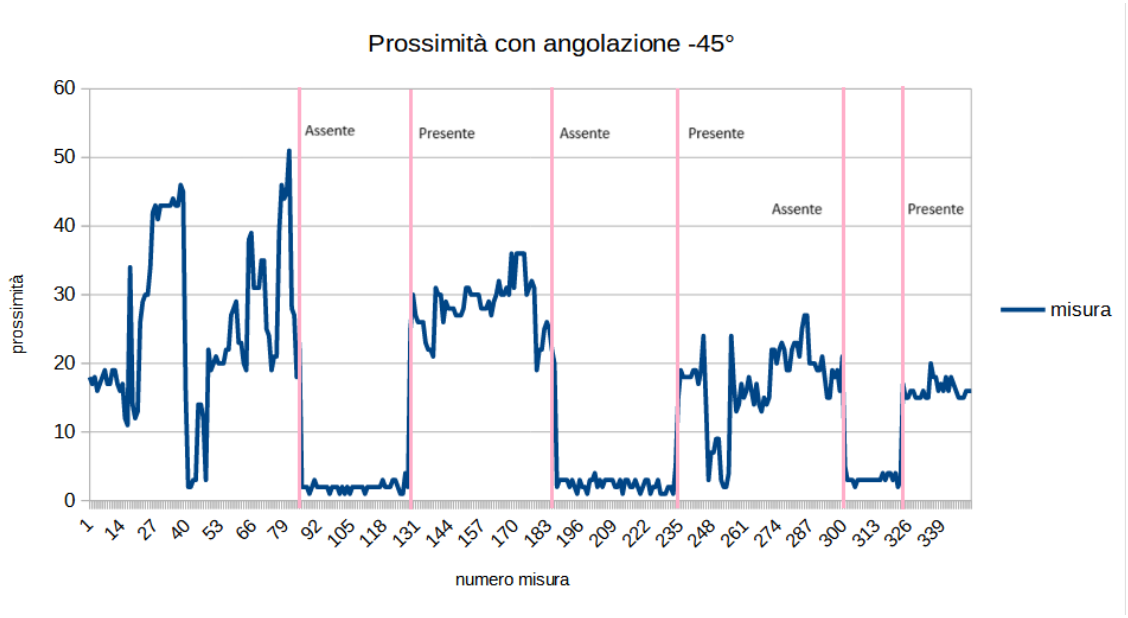


Figura 3.3: Primo gruppo di misure

In figura 3.3 possiamo vedere il cambiamento della possimità dal sensore nell'arco di due ore con il sensore posizionato alla nostra sinistra di -45° rispetto a noi ad una distanza di circa 80 cm. I picchi iniziali presenti nel grafico (dalla misura 16 alla misura 81) son dovuti a spostamenti del corpo naturali sulla scrivania, avvicinandosi anche verso il sensore in maniera rilevante. Gli intervalli di misura con valori bassi (187 - 235, e 301 - 324) sono gli intervalli dove non c'è nessuno seduto, calcolando una prossimità dal valore vicino allo zero che sono tipicamente un valore compreso tra 2 e 4, da cui possiamo dedurre che la misura oscilla poco tra questi valori. I restanti intervalli dal valore più alto (tipicamente compreso tra 15 e 35) segnano correttamente la presenza della persona, con i relativi spostamenti naturali del corpo. Gli intervalli di presenza e assenza raccolti dal sensore corrispondono ai movimenti appuntati durante la fase di test. Non è stato rilevato alcun cambiamento dell'efficienza del sensore durante il periodo di utilizzo, considerando che durante il test, le misure sono state effettuate ad intervalli di 100 ms, dimostrando un'ottima capacità di operatività in condizioni di intenso utilizzo.

Secondo gruppo

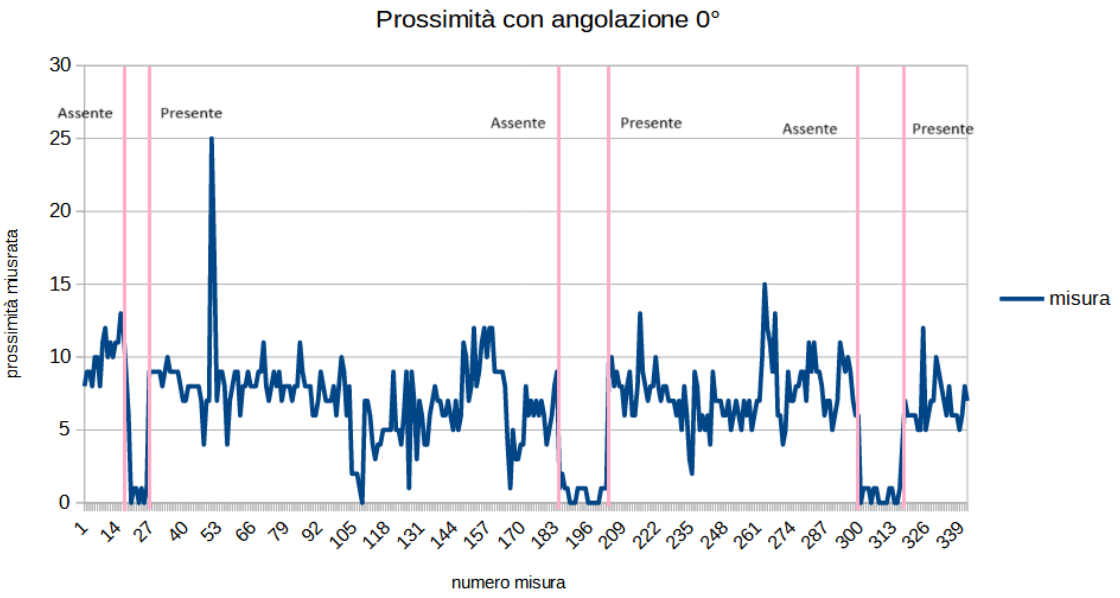


Figura 3.4: Secondo gruppo di misure

In figura 3.4 possiamo vedere il cambiamento della possimità dal sensore nell'arco di due ore con il sensore posizionato di fronte a noi a 0° di angolazione ad una distanza di circa 80 cm. Da questa angolazione, possiamo notare dei cambiamenti sulle misure rispetto al gruppo precedente. La misura fornita dal sensore quando siamo fermi sulla scrivania è formata anche da valori nulli, tali valori sono visibili tra gli intervalli di misure 13 - 21, 194 - 205 e 303 - 320, dove la misura arriva a toccare lo zero più volte. I picchi sul grafico sono dati da movimenti naturali, e stando seduti fermi si può notare come le misure oscillino per lo più intorno ad un valore costante. Rispetto al gruppo precedente, quando rileva la presenza da seduti ci da un valore di prossimità più basso, tendenzialmente tra 5 e 12. Questo dipende dalla posizione non identica del sistema per le tre posizioni, ed è ragionevole pensare che sia stato posizionato in partenza più vicino rispetto alle misure del primo gruppo. Quando il sensore non vede nessuno alla scrivania, la misura di prossimità assume un valore minore rispetto al gruppo precedente (compreso tra 0 e 2). Questo dipende, oltre dalla posizione non identica del sistema nelle tre posizioni, per esempio da come lasciamo la sedia una volta alzati. L'infrarosso del sensore, essendo puntato verso di noi quando siamo seduti, può andare incontro allo schienale della sedia che a sua volta può essere più vicino o più lontano ogni volta che ci alziamo. Come nel primo gruppo di misure, gli intervalli di presenza e assenza misurati dal sensore corrispondono ai movimenti

appuntati durante la fase di test. Anche qui non è stato rilevato alcun cambiamento dell'efficienza del sensore durante il periodo di utilizzo durante il test dimostrando un'ottima capacità di operatività anche con un'angolazione nulla rispetto al soggetto.

Terzo gruppo

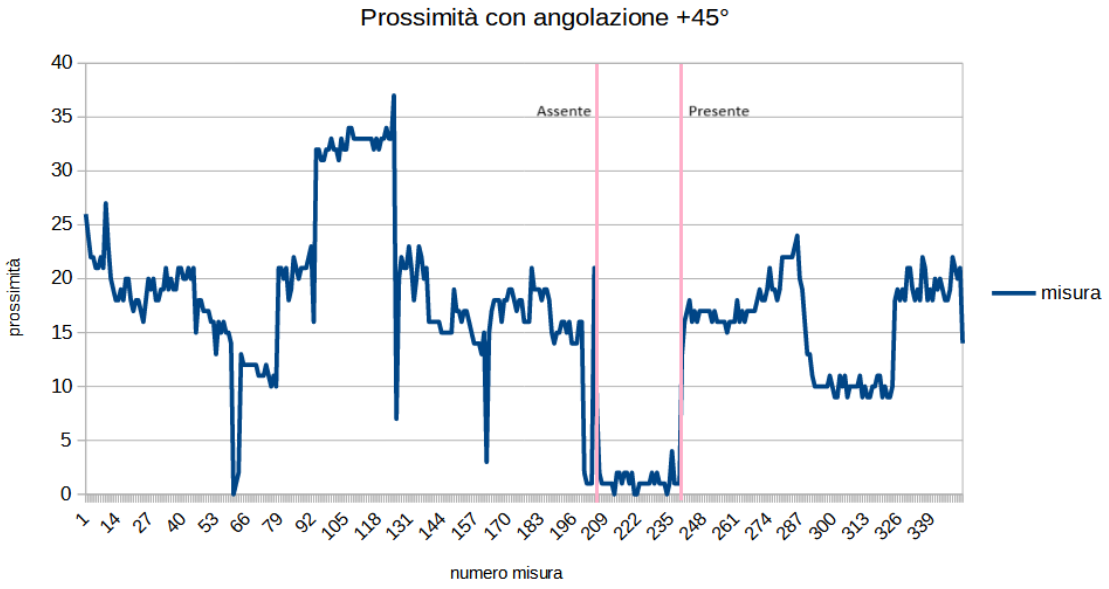


Figura 3.5: Terzo gruppo di misure

In figura 3.5 possiamo vedere il cambiamento della possimità dal sensore nell'arco di due ore con il sensore posizionato alla nostra destra di +45° rispetto a noi ad una distanza di circa 80 cm. Durante le misure iniziali sono stati effettuati dei movimenti con il corpo, che si possono notare dalla partenza del grafico da un valore più alto rispetto ai valori misurati da questa angolazione quando si è fermi. I valori nulli misurati in assenza dalla scrivania questa volta sono minori rispetto al secondo gruppo (intervallo di misura 210 - 243) sempre per il fatto che la posizione del sistema non è identica per le tre angolazioni. I picchi alti e bassi son dovuti a brevi spostamenti naturali momentanei, e si può notare che in condizioni di presenza e di stazionarietà sulla sedia nell'intervallo di misure 292 - 327 i valori misurati oscillano poco (generalmente tra 15 e 20), così come i valori in assenza (tra 0 e 3). Gli intervalli di presenza e assenza misurati dal sensore corrispondono anche in quest'ultimo gruppo ai movimenti annotati durante la fase di test, e non è stato rilevato nessun cambiamento dell'efficenza del sensore durante l'utilizzo da questa angolazione.

3.2 Sensori microfonici

Per quanto riguarda i microfoni, dobbiamo testare i suoi due compiti principali:

- Identificare la provenienza del rumore
- Identificare la tipologia di rumore (ambientale o parlato).

Per fare ciò, è stato utilizzato il terminale di Arduino IDE, sfruttando la comunicazione seriale.
Sono stati fatti due test differenti per queste funzionalità: il primo è riguardo il rumore che sappiamo, se presente, può arrivare da destra o da sinistra. Il secondo invece per essere definito rumore ambientale deve stare sotto una soglia di rumore stabilita, purchè sia sufficientemente bassa per definirlo tale. Per testare questa funzionalità, sono state fatte due prove differenti: la prima prova è fatta simulando una parlata dal lato sinistro e dal lato destro, poi stando in silenzio e facendo captare ai microfoni solo il rumore presente nella stanza.
La distinzione è interamente gestita con il firmware, e facendo i test abbiamo ottenuto i seguenti risultati:

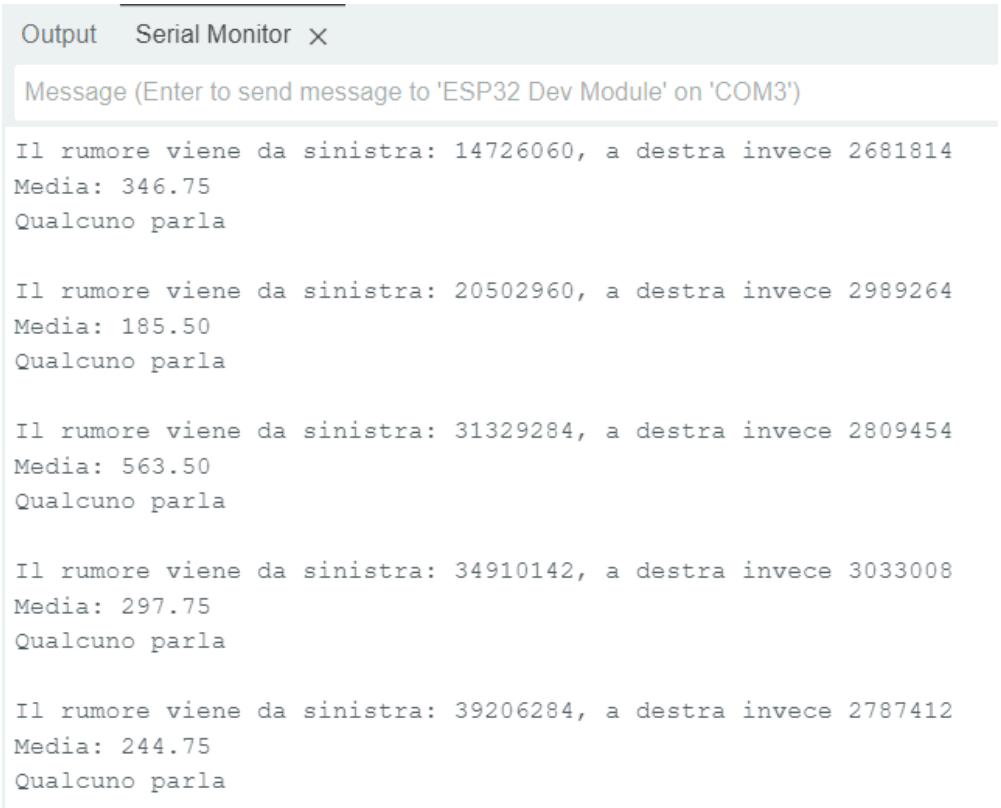


Figura 3.6: Parlata a sinistra.

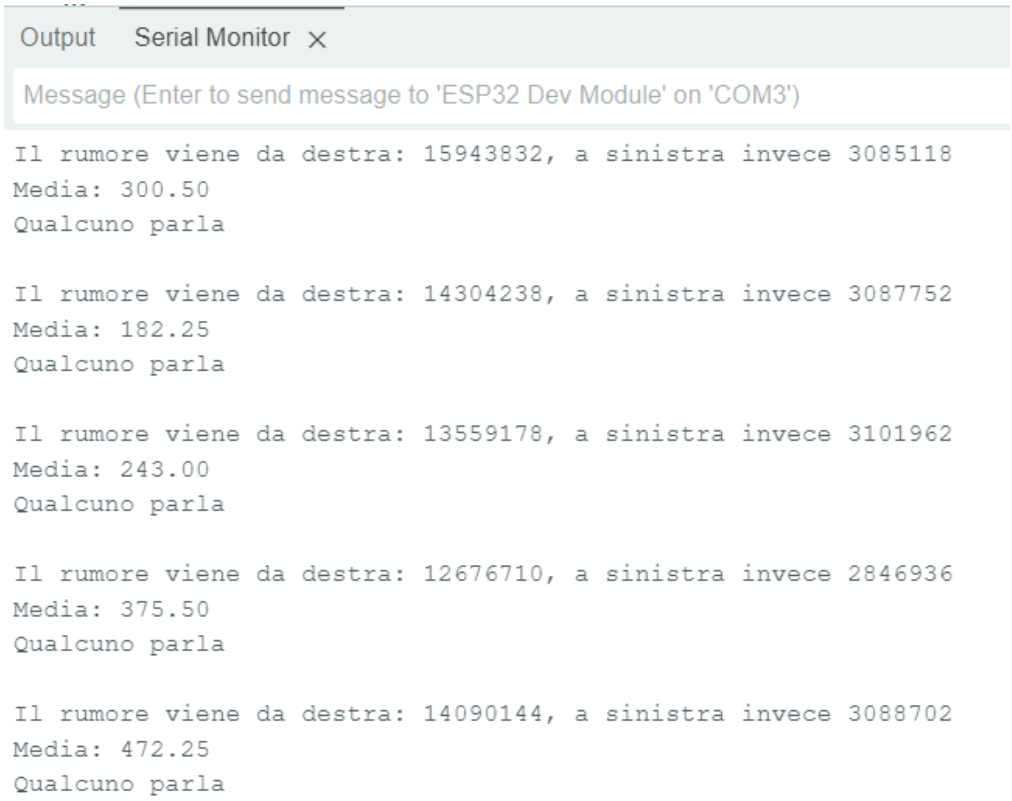


Figura 3.7: Parlata a destra.

Come possiamo vedere dal monitor seriale, parlando da sinistra o da destra, riusciamo a distinguere da dove arriva il rumore, osservando il valore delle somme del rumore captato in ingresso ai microfoni in un intervallo di tempo. La prima parlata simulata è stata compiuta a sinistra del prototipo. Infatti, in figura 3.6 possiamo vedere che il rumore della parlata arriva effettivamente da sinistra, notando un valore di rumore maggiore nel canale di sinistra rispetto al canale di destra. Stessa cosa per quanto rigaurda la seconda parlata simulata a destra, dove possiamo effettivamente vedere dalla figura 3.7 che il rumore della parlata arriva da destra, notando anche qui un valore di rumore maggiore nel canale di destra rispetto al canale di sinistra.
In entrambi i casi la media del rumore è sopra la soglia per definirlo rumore ambientale, quindi il sistema ha identificato correttamente la presenza di voce, comunicandocelo con la dicitura "Qualcuno parla".

L'ultimo test da eseguire è per identificare se il rumore è ambientale o meno, quindi ora vediamo la risposta in condizioni di silenzio ed in assenza di rumori forti. Senza

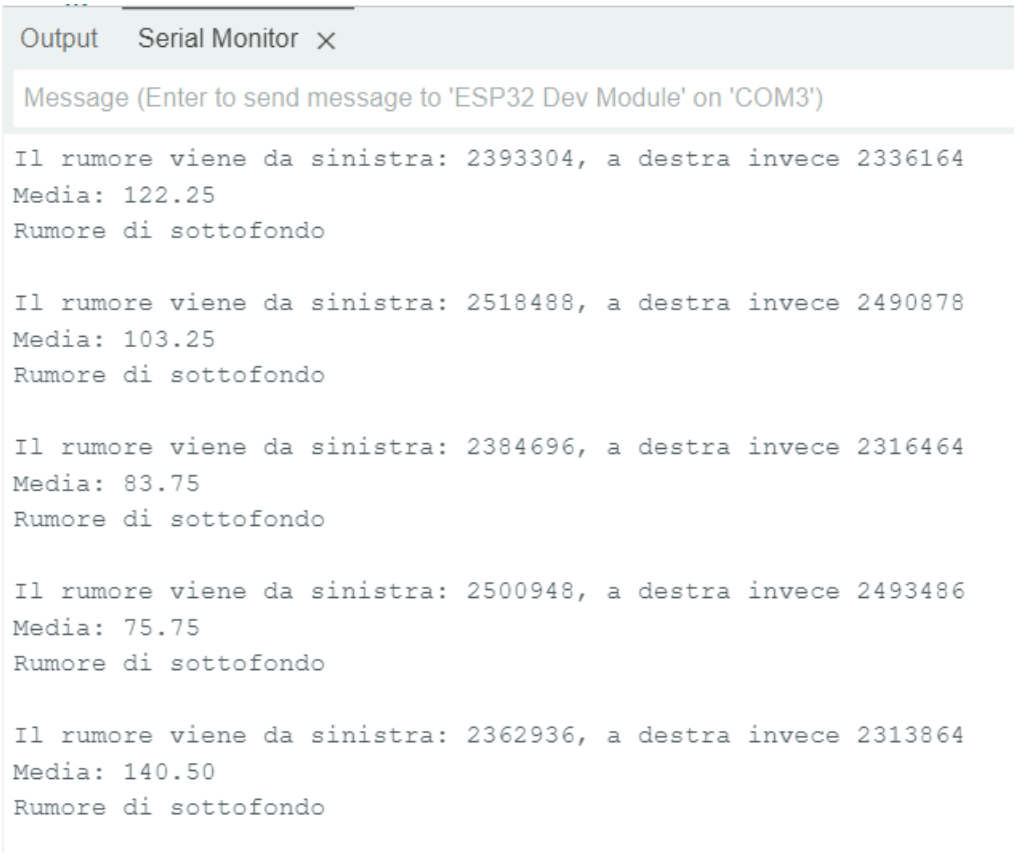


Figura 3.8: Identificazione rumore ambientale

parlare è presente solo il rumore ambientale, e la media calcolata è entro la soglia per definire il rumore come di sottofondo comunicandocelo con la dicitura "Rumore di sottofondo". I microfoni forniscono un valore simile tra i due, più basso rispetto ai precedenti.

Capitolo 4

Conclusioni

L'obiettivo di questo progetto era quello di ideare, progettare e realizzare un sistema di rilevamento della presenza per complementi d'arredo smart, in grado di rilevare la presenza sia fisica sia sonora di persone all'interno di una stanza ed alla scrivania. Si può dire che l'obiettivo è stato raggiunto in maniera più che soddisfacente. Rispetto all'idea iniziale, è stato aggiunto un sensore di temperatura ed umidità per eventuali funzioni aggiutive che possono tornare utili al progetto, come per esempio far attivare l'impianto di areazione.
Sono tuttavia emerse complicazioni durante lo sviluppo del progetto con il VCNL4200, che non ha una libreria standard creata per ESP32 ma solo per Arduino. Nonostante i due linguaggi siano identici, per un funzionamento corretto è stato necessario implementare nuove funzionalità e controlli per la scrittura dei registri con I2C. Inoltre, è stato notato che il sensore saltuariamente può non operare quando avviamo il sistema, ma partirà solo dopo aver effettuato un brevissimo cortocircuito tra 5V e GND. Questo può essere dovuto alla presenza di capacità parassite.
Per quanto riguarda il rumore, è stato visto nella fase iniziale di collaudo che inizialmente uno dei due microfoni captava un valore di rumore molto più alto rispetto all'altro. Questo valore si è assestato automaticamente con l'utilizzo nel tempo.

Uno sviluppo futuro del progetto sarà l'inclusione di altoparlanti utili ad eliminare il rumore ambientale, insonorizzando l'ambiente emettendo un apposito suono per quel tipo di rumore. Per fare ciò si utilizzerà sempre l'interfaccia I2S, però con l'aggiunta di un algoritmo necessariamente rapido ed efficiente per questo lavoro. Può essere utile integrare un sistema IoT per ricevere dati utili dal pannello mentre non siamo presenti, come per esempio qualità dell'aria, temperatura, presenza o meno di persone all'interno della stanza o nella scrivania. Questo può essere fatto senza l'aggiunta di ulteriore hardware per la comunicazione WiFi, sfruttando il modulo WiFi integrato in ESP32.
Dopo queste validazioni, in laboratorio sarà possibile procedere con il valutare la precisione nel contesto di applicazione reale, ma questo esula dal lavoro di tesi considerato.

Sitografia e datasheets consultati

1. https://it.wikipedia.org/wiki/Sensore_a_infrarossi_passivo
2. https://www.internet4things.it/iot-library/sensori-pir-cosa-sono-e-quali-sono-i-vantaggi-duso/
3. https://www.adrirobot.it/sensore_infrarosso_passivo_pir_sensor/#Rilevatore_di_movimento_basato_su_PIR
4. https://it.lambdageeks.com/how-does-ultrasonic-sensor-work/
5. https://it.lambdageeks.com/ultrasonic-sensor/
6. https://it.wikipedia.org/wiki/Ultrasuoni
7. https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino/
8. https://hbm.com/it/7646/what-is-a-pressure-sensor/
9. https://www.internet4things.it/iot-library/sensore-capacitivo-cose-e-quanti-tipi-ne-esistono/
10. https://it.wikipedia.org/wiki/Sensore_piezoresistivo
11. https://blog.atik.it/2011/04/10/principio-di-funzionamento-della-microonda-di-un-rivelatore-di-movimento-2/
12. https://reccom.org/sensori-di-movimento-pir-microonde-come-funzionano/
13. https://iccivitella.it/come-funziona-una-fotocellula/
14. https://it.wikipedia.org/wiki/Sensore_fotoelettrico#Sistemi_di_rilevamento
15. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf
16. https://it.wikipedia.org/wiki/Arduino_IDE
17. https://www.vishay.com/docs/84430/vcnl4200.pdf
18. https://www.vishay.com/docs/84327/designingvcnl4200.pdf

19. https://microfono.rocks/it/che-cose-un-microfono-mems-sistemi-micro-elettromeccanici/

20. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

21. https://it.wikipedia.org/wiki/I%C2%B2C

22. https://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2s-interface/

23. https://en.wikipedia.org/wiki/I

24. https://en.wikipedia.org/wiki/Watchdog_timer

25. https://it.wikipedia.org/wiki/Funzione_(informatica)

26. https://www.arduino.cc/reference/en/language/functions/communication/wire/

27. https://www.atomic14.com/2021/04/20/esp32-i2s-dma-buf-len-buf-count.html

28. https://www.arduino.cc/reference/en/language/functions/communication/wire/

29. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html

30. https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

Appendice 1 - Rullino foto

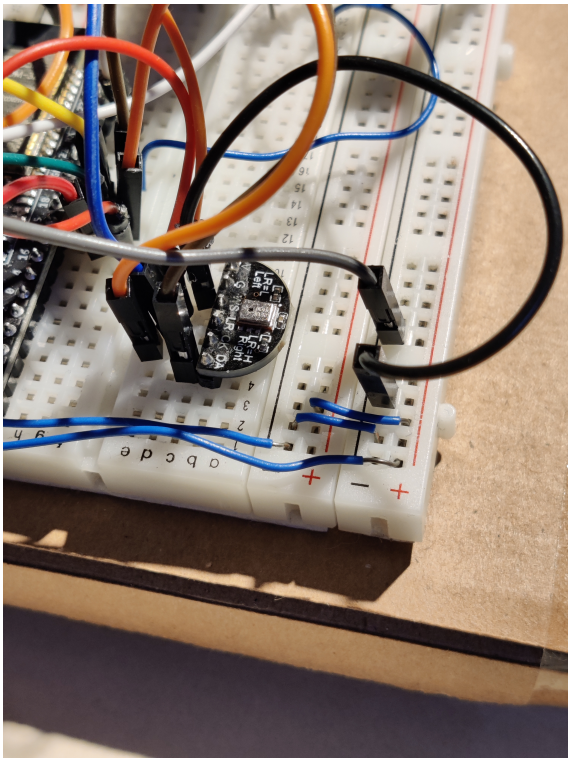


Figura 4.1: Microfono di destra.

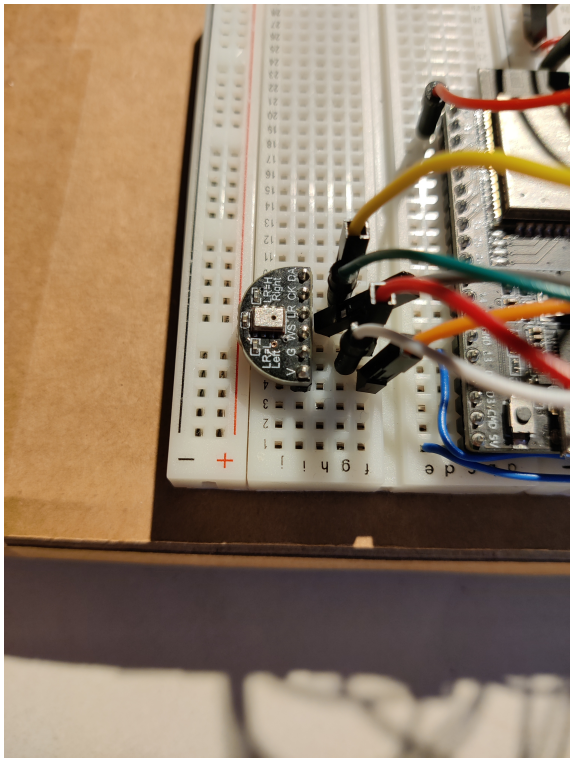


Figura 4.2: Microfono di sinistra.

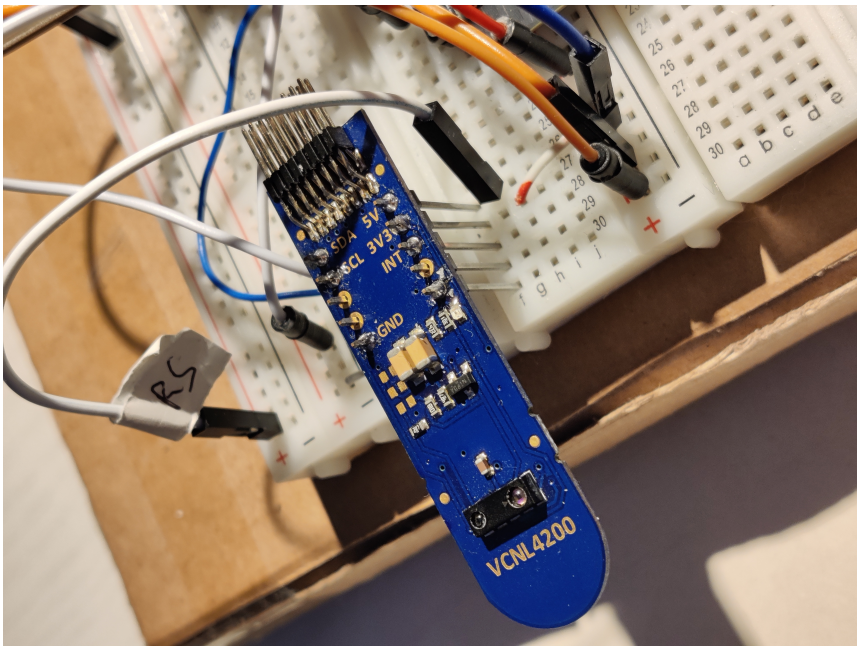


Figura 4.3: VCNL4200

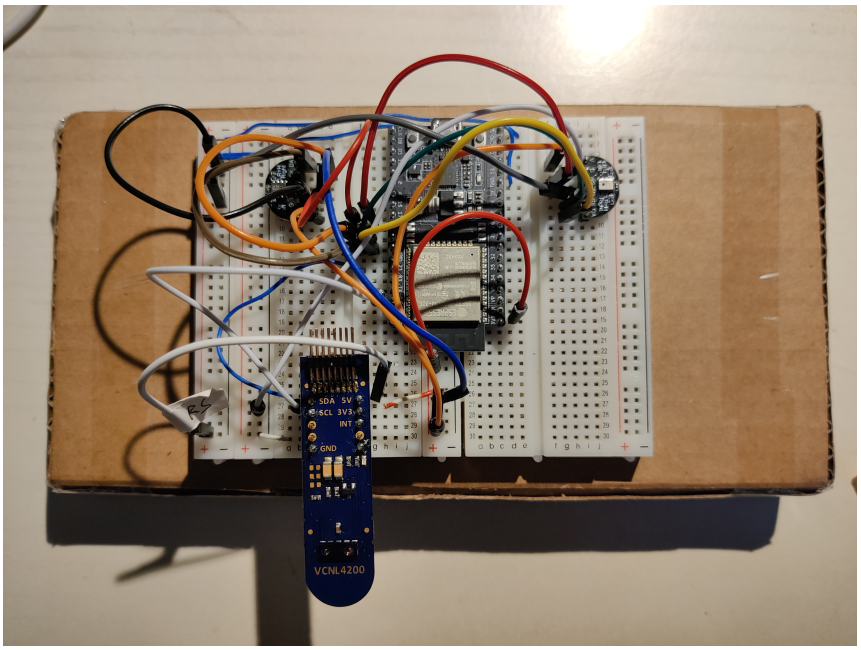


Figura 4.4: Fronte prototipo

Appendice 2 - Codice firmware

```
#include <Arduino.h>
#include <DHT.h>
#include <time.h>
#include <esp_task_wdt.h>
#include <driver/i2s.h>
#include <driver/dac.h>
#include <soc/i2s_reg.h>
#include <Wire.h>
#include <Vishay_VCNL4200.h>


/*COSTANTS*/
//Watchdog
#define WDT_TIMEOUT 3  /*Time before watchdog triggers (in s)*/
//DHT11
#define DHT11_PIN 23
#define DHTTYPE DHT11 /*Decomment to change DHT sensor type usage*/
//#define DHTTYPE DHT21
//#define DHTTYPE DHT22
//INMP1441 I2S microphone
#define I2S_WS 5 /*These pins (WS SD SCK) can be changed without any specification*/
#define I2S_SD 19
#define I2S_SCK 18
#define I2S_PORT I2S_NUM_1 /*I2S has two cores, we use core number 1*/
#define bufferLen 64 //Buffer size

#define PRINT_TIME 500 //In ms
#define BG_LEVEL 160 //Noise thereshold
```

```
29
30 #define PRINT_TIME_2 1000
31
32 //////////////////////////
33 #define SENSOR_ADDR 0x51
34
35 //Ambient Light Sensor range, defines changes integration
     time (ms)
36 //#define ALS_MAX_RANGE_1573  //Integration time: 50ms
37 #define ALS_MAX_RANGE_786   //"                " 100ms
38 //#define ALS_MAX_RANGE_393   //"                " 200ms
39 //#define ALS_MAX_RANGE_197   //"                " 400ms
40
41 #ifdef ALS_MAX_RANGE_1573
42   float sensitivity= 0.024f;
43   byte als_conf_L = B00000001;
44 #endif
45
46 #ifdef ALS_MAX_RANGE_786
47   float sensitivity= 0.012f;
48   byte als_conf_L = B01000001;
49 #endif
50
51 #ifdef ALS_MAX_RANGE_393
52   float sensitivity= 0.006f;
53   byte als_conf_L = B10000001;
54 #endif
55
56 #ifdef ALS_MAX_RANGE_197
57   float sensitivity= 0.003f;
58   byte als_conf_L = B11000001;
59 #endif
60
61 unsigned long last = 0;
62
63 float celsius;
64 //float fahrenheit;
65 float humidity;
66
67 int16_t sBuffer[bufferLen]; //Buffer I2S
68 uint32_t left = 0;
69 uint32_t right = 0;
70
```

```
71 DHT dht(DHT11_PIN, DHTTYPE);  //DHT11 initialization
72 TaskHandle_t plot_i2s_microphoneName;
73
74
75 void setup() {
76
77   Wire.begin();
78   vcnl4200.begin();
79   delay(100);
80
81   //Write to PS_CONF1 and PS_CONF2 to set Proximity Sensor (
       PS) settings.
82   write_register(SENSOR_ADDR, 0x3, B00011010, B00001000);
83   //Write to ALS_CONF to turn Ambient Light Sensor (ALS) off
       .
84   write_register(SENSOR_ADDR, 0x0, als_conf_L, B00000000);
85   //Write to PS_THDL_L and PS_THDL_H to set Proximity Sensor
        (PS) Threshhold.
86   write_register(SENSOR_ADDR, 0x6, B00010000, B00000000);
87   //Write to PS_CONF3 and PS_MS to set Proximity Sensor (PS)
        settings.
88   write_register(SENSOR_ADDR, 0x4, B01110000, B00000111);
89
90   //These two lines fixes false data on right channel when
       only left channel is interpellated
91   REG_SET_BIT(I2S_TIMING_REG(I2S_PORT), BIT(9));
92   REG_SET_BIT(I2S_CONF_REG(I2S_PORT), I2S_RX_MSB_SHIFT);
93
94   xTaskCreatePinnedToCore(
95     plot_i2s_microphone, /* Function to implement the task
          */
96     "plot_i2s_microphoneName", /* Name of the task */
97     10000,  /* Stack size in words */
98     NULL,  /* Task input parameter */
99     0,  /* Priority of the task */
100    &plot_i2s_microphoneName,  /* Task handle. */
101    1   /* Core where the task should run */
102  );
103
104  dht.begin();
105
106  //Serial
107  Serial.begin(115200);
```

```
108   while(!Serial)
109     delay(100);
110
111   //Set up I2S
112   i2s_install();
113   i2s_setpin();
114   i2s_start(I2S_PORT);
115
116   //Watchdog
117   esp_task_wdt_init(WDT_TIMEOUT, true); //Watchdog enabled
118   esp_task_wdt_add(NULL);
119
120 }
121
122
123 void loop() {
124
125   esp_task_wdt_reset(); //reset watchdog
126   //t_h_DHT11();
127
128   int reg = 0x8; //Proximity Sensor (PS) output
129
130   //Write to regsiter
131   Wire.beginTransmission(SENSOR_ADDR);
132   Wire.write(reg);
133   Wire.endTransmission(false);
134
135   //Read data
136   uint16_t data[2] = {0, 0};  //The sensor provides the
        output over 2 bytes.
137   Wire.requestFrom(SENSOR_ADDR, 2);
138   data[0] = Wire.read();
139   data[1] = Wire.read();
140   int value1 = (int)(data[1] << 8) + (int)(data[0]); //
        Combines bytes
141
142   reg = 0x09; //Ambient Light Sensor (ALS) output
143
144   //Write to regsiter
145   Wire.beginTransmission(SENSOR_ADDR);
146   Wire.write(reg);
147   Wire.endTransmission(false);
148
```

```
149   Wire.requestFrom(SENSOR_ADDR, 2);
150   data[0] = Wire.read();
151   data[1] = Wire.read();
152   int value2 = (int)(data[1] << 8) + (int)(data[0]); //
         Combines bytes
153
154   Serial.print("Proximity:␣");
155   Serial.println(value1);
156   Serial.print("Lux:␣");
157   Serial.println(value2*sensitivity);
158   Serial.println("␣");
159
160   delay(500);
161
162 }
163
164
165 /*
166 t_h_DHT11: it measure ambiental temperature and humidity.
167           Param: null
168           Type return: null
169 */
170 void t_h_DHT11(){
171
172   celsius = dht.readTemperature();
173   humidity = dht.readHumidity();
174
175   if(isnan(celsius) || isnan(humidity)) //If measue not
         valid return
176     return;
177
178   Serial.print("Temperatura␣in␣Celsius␣");
179   Serial.print(celsius);
180   Serial.print("␣e␣in␣Fahrenheit␣");
181   Serial.println(celsius * (9/5) + 32);
182
183   Serial.print("Umidita':␣");
184   Serial.println(humidity);
185
186 }
187
188
189
```

```
190 /*
191 write_register: sets VCNL4200's registers for the
      measurements.
192                 Param: int, int, int, int
193                 Type return: null
194 */
195 void write_register(int device_addr, int register_num, int
      low_byte, int high_byte){
196
197   Wire.beginTransmission(device_addr);
198
199   Wire.write(register_num);
200   Wire.write((uint8_t)(high_byte & 0xFF));
201   Wire.write((uint8_t)((low_byte >> 8) & 0xFF));
202
203   Wire.endTransmission(false);
204 }
205
206 /*
207 i2s_install: configures the processor for I2S usage.
208                 Param: null
209                 Type return: null
210 */
211 void i2s_install(){
212   //Set up I2S processor configuration
213   const i2s_config_t i2s_config = {
214     .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
215     .sample_rate = 464100, //Frequenza campionamento in kHz
216     .bits_per_sample = i2s_bits_per_sample_t(16),
217     //.channel_format = I2S_CHANNEL_FMT_ONLY_RIGHT,
218     //.channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
219     .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,//LEFT -
        GND, RIGHT - 3v3
220     .communication_format = I2S_COMM_FORMAT_I2S,
221     .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
222     .dma_buf_count = 4,
223     .dma_buf_len = bufferLen,
224     .use_apll = false,
225     .tx_desc_auto_clear = false,
226     .fixed_mclk = 0
227   };
228   i2s_driver_install(I2S_PORT,&i2s_config,0,NULL);
229 }
```

```
230
231 /*
232 i2s_setpin: set pins used by I2S.
233                 Param: null
234                 Type return: null
235 */
236 void i2s_setpin(){
237   //Set I2S pin configuration
238   const i2s_pin_config_t pin_config = {
239     .bck_io_num = I2S_SCK,
240     .ws_io_num = I2S_WS,
241     .data_out_num = -1,
242     .data_in_num = I2S_SD
243   };
244   i2s_set_pin(I2S_PORT, &pin_config);
245 }
246
247 /*
248 plot_i2s_microphone: thread who manage data transimtted by
     the microphones.
249                 Param: void *
250                 Type return: null
251 */
252 void plot_i2s_microphone(void * parameters){
253
254   for(;;){
255
256     esp_task_wdt_reset(); //reset watchdog
257
258     //Get I2S data and place in data buffer
259     size_t bytesIn = 0;
260     esp_err_t result = i2s_read(I2S_PORT, &sBuffer,
          bufferLen, &bytesIn, portMAX_DELAY);
261     float mean = 0;
262     int16_t samples_read;
263
264     if(result == ESP_OK){
265       samples_read = bytesIn / 8;
266       if(samples_read > 0){
267         for(int16_t i = 0; i < samples_read; i++){
268           if(i%2 == 0){ //EVEN LEFT, ODD RIGHT
269             left +=  abs(sBuffer[i]);
270             mean += abs(sBuffer[i]);
```

```
271           }else{
272             right += abs(sBuffer[i]);
273             mean += abs(sBuffer[i]);
274           }
275         }
276         //Average of the data captured
277         mean /= samples_read;
278       }
279     }
280
281     if(millis() - last >= PRINT_TIME){
282       if(left > right){
283         Serial.print("Il rumore viene da sinistra: ");
284         Serial.print(left);
285         Serial.print(", a destra invece ");
286         Serial.println(right);
287         Serial.print("Media: ");
288         Serial.println(mean);
289       }else{
290         Serial.print("Il rumore viene da destra: ");
291         Serial.print(right);
292         Serial.print(", a sinistra invece ");
293         Serial.println(left);
294         Serial.print("Media: ");
295         Serial.println(mean);
296       }
297       if(mean > BG_LEVEL){
298         Serial.println("Qualcuno parla");
299       }else{
300         Serial.println("Rumore di sottofondo");
301       }
302
303       Serial.println(" ");
304
305       left = 0;
306       right = 0;
307       mean = 0;
308       last = millis();
309
310     }
311
312   } //for graph
313
```

```
314   vTaskDelete(NULL);  //Deletes the thread when it's
        finished, it will never happen
315
316 }
```