Agenzia nazionale per le nuove tecnologie,
l'energia e lo sviluppo economico sostenibile

MINISTERO DELLO SVILUPPO ECONOMICO

Ricerca di Sistema elettrico

Ingegnerizzazione sistema sensoriale Sesto Senso

Alberto Nisti, Francesca Dini

Report RdS/PAR2017/101

INGEGNERIZZAZIONE SISTEMA SENSORIALE **SESTO SENSO**

Nisti Alberto, Dini Francesca

Settembre 2018

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Annuale di Realizzazione 2017
Area: Efficienza energetica e risparmio di energia negli usi finali elettrici e interazione con altri vettori energetici
Progetto: D.6 Sviluppo di un modello integrato di smart district urbano
Obiettivo: Sistemi e servizi smart per edifici
Responsabile del Progetto: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno del contratto "Ingegnerizzazione del Sistema Sesto Senso ideato e realizzato in forma prototipale da ENEA"
Responsabile scientifico ENEA: Francesco Romanello
Responsabile scientifico INNOSENSOR: Alberto Nisti

Indice

SOMMARIO 4
1 INTRODUZIONE 5
2 DESCRIZIONE DELLE ATTIVITÀ SVOLTE E RISULTATI 6
2.1 CENTRALINA 6
2.1.1 Elettronica e case *6*
2.1.2 Schema funzionale *7*
Z-WAVE MODULE REGISTER MAP (MODULE 8) 8
Z-WAVE MODULE REGISTER MAP (MODULE 10) 10
2.1.3 Restful http API *11*
2.1.4 Interfaccia web *12*
2.2 SENSORISTICA ACCESSORIA 14
2.2.1 Sensore Porta *16*
2.2.2 Sensore Finestra *17*
2.2.3 Sensore centro stanza *17*
2.2.4 Sensore conta persone *18*
2.2.4.1 Schema elettrico 21
2.2.4.2 Firmware 22
2.2.4.3 Meccanica 24
2.3 SENSORE VIRTUALE DI CO2 26
3 CONCLUSIONI 34
4 RIFERIMENTI BIBLIOGRAFICI 35
5 ABBREVIAZIONI ED ACRONIMI 36

Sommario

Innosensor ha realizzato per conto di ENEA una ingegnerizzazione del sistema prototipale sviluppato dall'ENEA denominato SESTO SENSO.
Tale sistema è composto da un'unità centrale, dei sensori accessori commerciali basati su tecnologia Zwave e un sensore conta persone Zwave il cui algoritmo di calcolo è stato sviluppato dall'ENEA stessa.
Il sistema è capace di ricavare informazioni dai sensori, elaborarne i segnali trasmessi e ricavare un valore virtuale di CO2 (diossido di carbonio) secondo un algoritmo di predizione basato sull'utilizzo di rete neurali sviluppato dall'Università Roma TRE.
Tutte le informazioni reali e virtuali sono messe a disposizioni degli utenti localmente tramite interfaccia web visibile su un touchscreen di cui è fornita l'unità centrale e da remoto attraverso un servizio di API basato su chiamate REST (Representational state transfer) con il quale è possibile interrogare tutte le centraline attaccate in rete.
Il lavoro è stato svolto partendo dal prototipo realizzato da ENEA cercando attraverso una ingegnerizzazione completa di rendere il sistema più attraente a livello commerciale e realizzando una preserie industriale composta da tre sistemi completi.
L'ingegnerizzazione è stata applicata sia al sistema conta persone che alla centralina, i sensori accessori sono stati invece scelti dopo una selezione dei sensori Zwave attualmente presenti sul mercato in base alle performance, al prezzo e alle funzionalità.
L'ingegnerizzazione del sensore conta persone è stata focalizzata principalmente sull'ottimizzazione delle performance della coppia di sensori utilizzati dall'algoritmo per ricavare il numero di persone. Dopo l'analisi e test di diverse tipologie di sensori è stato individuato ed integrato il sensore che meglio soddisfa le esigenze del conta persone (sensore Infrarosso attivo). Si è proceduto poi con lo sviluppo di un'elettronica dedicata (alimentazione, trasduzione ed elaborazione) cercando di ridurre gli ingombri e abbassare i consumi, infine è stata introdotta la tecnologia Zwave attraverso una scheda elettronica specifica in modo da rendere il sensore indipendente dalla centralina e quindi di più facile installazione. In ultima analisi è stato realizzato un case di ridotte dimensioni e gradevole aspetto che permette di ridurre le interferenze tra i sensori rendendo la lettura più robusta.
La centralina è stata sviluppata partendo da una scheda Raspberry Pi 3 alla quale è stato aggiunto un touchscreen da 7'', il tutto racchiuso in un case che rende la centralina un gradevole oggetto da tavolo. Alla centralina è stata aggiunta la tecnologia Zwave attraverso una pennina USB Zwave (open Zwave) che funge da controller della rete Zwave sesto senso. La centralina dispone di un attacco ethernet e interfaccia wifi grazie alle quali si connette con un cloud dedicato dove è possibile accedere al servizio di API attraverso chiamate REST. Tale servizio API è fondamentale per l'integrazione del sistema sesto senso in sistemi di domotica di terze parte, quali ad esempio il sistema cloud di APIO.
I sensori accessori commerciali selezionati oltre a fornire tutte le informazioni necessarie alla centralina per calcolare il valore virtuale di CO2 (Temperatura, umidità, stato apertura porta, stato apertura finestra), si compongono anche di un sensore PIR (passive infra red) che funge da feedback sul valore calcolato dal conta persona andando a completarne l'algoritmo.

1 Introduzione

L'obiettivo del lavoro svolto è stato l'ingegnerizzare in tutte le sue parti del sistema SESTO SENSO di proprietà dell'ENEA, al fine di produrre una pre-serie industriale di 3 apparati completi ready to market.
Lo scopo è stato quello di rendere il sistema user-friendly, sia in termini di utilizzo che in termini estetici e rendere il sistema, sia a livello hardware che computazionale, adeguato per integrare sistemi elaborativi più avanzati e aggiungere parti esterne utili per raggiungere gli scopi applicativi prefissati dal progetto e le ulteriori espansioni future.
Particolare attenzione è stata posta nel realizzare un sistema non chiuso ma espandibile; filosofia di progetto adottata in tutte le fasi e le scelte progettuali.
Il lavoro è stato svolto partendo del prototipo realizzato dall'ENEA, e si è proceduto andando a sostituire le parti con componenti più sofisticati nel caso in cui si ritenevano tali parti non sufficientemente performanti o limitanti per lo scopo ultimo del progetto.

Nel dettaglio, come da capitolato, le fasi di progetto hanno seguito la seguente linea guida:

Ottimizzazione del sistema di controllo accessi utilizzando l'algoritmo sviluppato dall'ENEA:
Preso atto dei test già effettuato dall'ENEA l'attività è stata mirata a realizzare un sistema di controllo accessi utilizzando sensori commerciali di diversa natura per minimizzare il più possibile gli errori causati da interferenze tra sensori, malfunzionamenti dei sensori IR utilizzati nel prototipo, robustezza della comunicazione con la centralina, disturbi, mal posizionamento dei sensori
Raggiunto un livello di errore del controllo accessi considerato accettabile i sensori Individuati sono stati integrati nell'unità centrale di controllo tramite sistema Wireless Zwave.

Ingegnerizzazione dell'unità centrale a microcontrollore:
Sostituzione dell'unità centrale attualmente utilizzata (Arduino Mega) con una scheda Raspberry Pi 3 al fine di permettere una più facile integrazione delle parti e sviluppi aggiuntivi per applicazioni future.
I sensori infrarossi attivi IR individuati sono stati collegati tramite Zwave, e quindi non più cablati, alla Raspberry Pi 3 e permettere così la libera installazione della centralina.
Lo schermo LCD e il tastierino di controllo sono stati sostituiti con uno schermo LCD TOUCH collegato alla Raspberry Pi 3 tramite scheda dedicata.
E' stata realizzata un'interfaccia grafica di controllo del sistema che permette la visualizzazione dei valori dei sensori, del numero di persone presenti e il valore virtuale di CO2 calcolato.
La scheda Raspberry Pi 3 è stata fornita di tecnologia trasmissiva Zwave tramite chiavetta USB e grazie al sistema di API (Application Programming Interface) http Restful rilasciato può essere connessa all'HUB APIO che verrà utilizzato come primo esempio di integrazione del sistema SESTO SENSO.

Il sistema è stato dotato di sensori wireless di luce, umidità e temperatura posti in punti diversi della stanza. Sono stati utilizzati inoltre un sensore PIR (Passive Infra Red) e un sensore di stato apertura/chiusura porte e finestre con tecnologie Zwave.

Infine è stato Integrato nel sistema dell'algoritmo per il calcolo del valore virtuale di CO2, secondo l'algoritmo sviluppato da ENEA-ROME TRE.

2 Descrizione delle attività svolte e risultati

2.1 Centralina

2.1.1 Elettronica e case

L'unità centrale del sistema sesto senso è basata sulla scheda elettronica Raspberry Pi 3 al quale è stato aggiunto tramite scheda dedicata un touch screen da 7'', con case:

Figura 1: Touchscreen + case + Raspberry Pi3

TOUCHSCREEN: Monitor a colori da 7" multi-touch capacitivo per Raspberry Pi
Il Touchscreen Dispone di una risoluzione di 800x480 pixel a 60 fotogrammi al secondo, colore RGB a 24 bit, angolo di visione 70° e scheda per il controllo del touch screen.
Per la configurazione sono necessari soltanto due collegamenti; l'alimentazione dalla porta GPIO del Raspberry Pi e un cavo ribbon DSI che si collega alla porta DSI presente sulla Raspberry Pi.
La scheda adattatore e Raspberry Pi è montata sul retro del monitor LCD utilizzando i fissaggi in dotazione.
Alimentazione: 5 Vdc - 500 mA tramite connettore micro USB.

Figura 2: Touch screen (installazione su raspberry)

CASE: Custodia in ABS (Acrilonitrile Butadiene Stirese) progettata per alloggiare la scheda Raspberry Pi 3 (2B / 3B e B+) e il touch screen LCD (Liquid Crystal Display) Raspberry Pi da 7" originale
Il case è predisposto con tutte le forature necessarie per i connettori e i collegamenti dei componenti. Sulla parte superiore è disponibile una fessura per inserire la camera originale per Raspberry Pi 3. Inoltre il coperchio posteriore può essere rimosso in modo semplice per proteggere/accedere a tutti i connettori interni.
Dimensioni esterne (mm): 197,12x46,76x115,64.

Figura 3: case della centralina

RASPBERRY: Raspberry Pi 3 - Model B

Tabella 1 - **Specifiche Tecniche Raspberry Pi 3:**

CPU	1.2GHz Quad-Core ARM Cortex-A53
Wireless LAN	802.11 bgn
Bluetooth	Bluetooth 4.1
RAM	1GB
USB Ports	4
GPIO	40 pins
HDMI	1
Camera interface (CSI)	1
Display interface (DSI)	1
Micro SD Slot	1
Ethernet Port	1
Power Source	Mirco USB 5V/2.5A

Power Charger Adapter:

- INPUT: AC 100V-240V 50/60 Hz

OUTPUT: DC 5V 2°

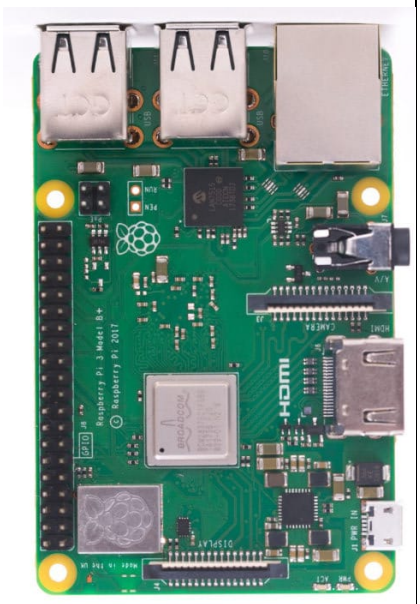


Figura 4: Raspberry

2.1.2 Schema funzionale

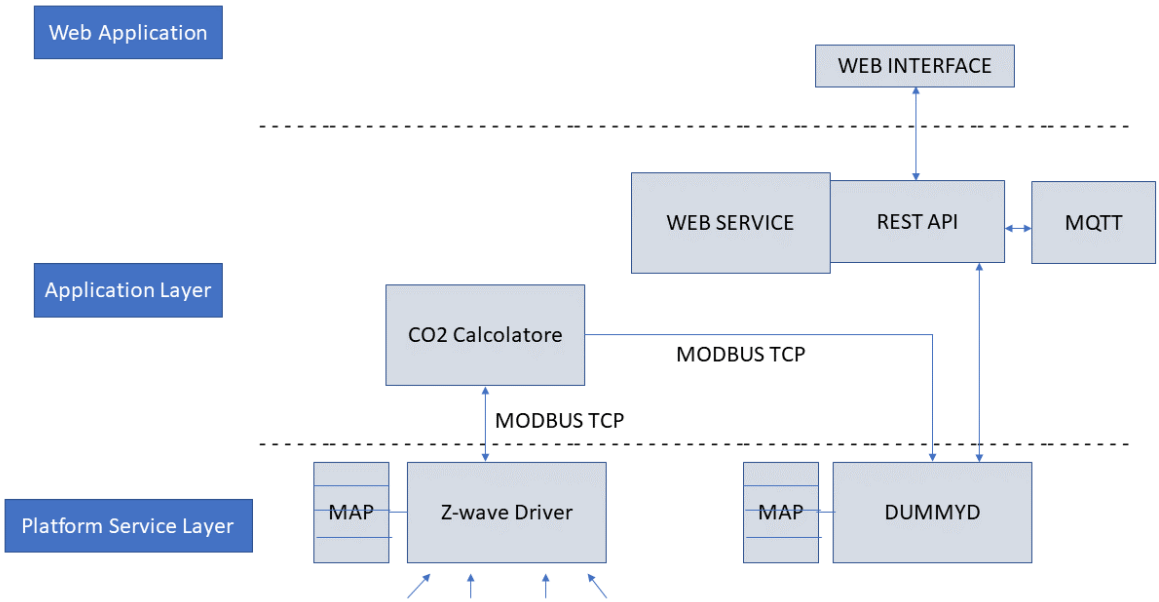


Figura 5: schema funzionale

Sulla centralina del sistema SESTO SENSO basata su scheda Raspberry Pi 3 è stato installato il sistema operativo Raspbian sul quale girano tutti gli applicativi; Raspbian è un sistema operativo basato sui rilasci ufficiali di Debian per l'Architettura ARM (Armhf), adattato proprio per l'utilizzo su Raspberry Pi 3.
Al fine di permettere un'interfaccia locale da visualizzare direttamente sulla centralina è stato installato il web Browser Chromium, configurato per partire come servizio in automatico (FULL SCREEN) all'accensione della Centralina.
L'interfaccia Web descritta nel dettaglio al capitolo 2.4 che gira a livello web application permette di visualizzare e modificare i dati grazie all'integrazione di un servizio di REST API messo a disposizione del Web Server locale installato sulla Raspberry Pi 3 che invece gira a livello Application Layer.
I dettagli di utilizzo delle REST API sesto senso sono descritti nel capitolo 2.3.
Lo stesso servizio di API è disponibile sia localmente che da remoto. Per l'interrogazione dei dati da remoto è stato creato un modulo MQTT (Message Queue Telemetry Transport) client sottoscritto a TOPIC specifici su un broker MQTT localizzato su un Server Cloud esterno.
I parametri di impostazione (indirizzo del Broker, id della centralina ecc) sono descritti su un file di configurazione il quale può essere modificato per ogni nuova centralina che si intende creare.
Sempre a livello Application Layer è stato sviluppato l'applicativo in Python che calcola la percentuale di CO2, "CO2 calcolatore" (Capitolo 5)
Il servizio REST API permette di leggere e modificare i dati immagazzinati in registri specifichi tramite protocollo MODBUS-TCP (Protocollo di comunicazione seriale a livello applicativo basato su registri dove nella versione TCP/IP ogni nodo (registro) può fungere da Master e da Slave).
In particolare il servizio REST API comunica con un modulo specifico DUMMYD a livello Platform Service Layer nel quale sono registrati tutti i valori dei sensori reali e virtuali.
I valori sono scritti nel modulo DUMMYD dall'applicativo "CO2 calcolatore" che provvede a riportare i valori letti dal driver Zwave nei registri secondo la mappatura (modulo e registro) di seguito riportata in Tabella 1:

Tabella 2: Mappatura dei valori dei sensori

Z-Wave module register map (module 8)		
Register Address	**Description**	**Source**
10	**People count**	
11	Battery level	Node 1
12	Temperature	Node 1
13	Relative Humidity	Node 1
14	Luminance	Node 1
15	Burglar/Tamper	Node 1
16	Door/Window	Node 1
17	Alarm type	Node 1
18	Alarm level	Node 1
19	Last seen timestamp HI	Node 1
20	Last seen timestamp LO	Node 1
21	Battery level	Node 2

22	Temperature	Node 2
23	Relative Humidity	Node 2
24	Luminance	Node 2
25	Burglar/Tamper	Node 2
26	**Door/Window (22: aperto, 23: chiuso)**	Node 2
27	Alarm type	Node 2
28	Alarm level	Node 2
29	Last seen timestamp HI	Node 2
30	Last seen timestamp LO	Node 2
31	Battery level	Node 3
32	**Temperature (*10)**	Node 3
33	**Relative Humidity (*10)**	Node 3
34	Luminance	Node 3
35	Burglar/Tamper	Node 3
36	Door/Window	Node 3
37	Alarm type	Node 3
38	Alarm level	Node 3
39	Last seen timestamp HI	Node 3
40	Last seen timestamp LO	Node 3
41	Battery level	Node 4
42	Temperature	Node 4
43	Relative Humidity	Node 4
44	Luminance	Node 4
45	Burglar/Tamper	Node 4
46	**Door/Window (22: aperto, 23: chiuso)**	Node 4
47	Alarm type	Node 4
48	Alarm level	Node 4
49	Last seen timestamp HI	Node 4
50	Last seen timestamp LO	Node 4

Z-Wave module register map (module 10)		
1	**CO2 ppm (*10)**	

I parametri di elaborazione sono prelevati dal Driver Zwave (libreria Open Zwave) installato nel sistema a livello Platform service layer. Tale Driver comunica con l'applicativo "CO2 calcolatore" tramite protocollo Modbus TCP.
Il driver che funziona da controller permette di gestire i sensori Zwave che devono essere inclusi in un ordine ben preciso nel sistema al fine di mantenere tutte le funzionalità (Tabella 2).

RICHIESTA NUMERO DI NODI
Il registro 1 del modulo 8 è riservato alla richiesta di informazioni alla centralina. In particolare leggendo il valore viene restituito il numero di nodi della rete Zwave presente.
INCLUSIONE / ESCLUSIONE SENSORE
Le modalità di inclusione/esclusione della rete Zwave seguono delle rigide regole di numerazione, è importante quindi includere/escludere i sensori mantenendone la numerazione al fine di preservare la concretezza delle informazione delle API.
Ad ogni nodo incluso viene assegnato un valore crescente; per ogni nodo escluso tutti i nodi con numerazione superiore subiscono un decremento di una posizione.
I registri 2 e 3 sono rispettivamente utilizzati per includere e escludere i nodi dalla rete.
Procedura di inclusione:
1 – Inviare il comando di inclusione (mod:8, reg:2, val:x). Il sistema entrerà in modalità inclusione fino al ricevimento di un segnale di inclusione da parte di un sensore.
2 – Cliccare sull'apposito pulsante di inclusione/esclusione sul sensore che si vuole includere.
3 – Il sistema tornerà in modalità normale e il sensore verrà incluso in ordine crescente.

Procedura di esclusione:
1 – Inviare il comando di esclusione (mod:8, reg:3, val:x). Il sistema entrerà in modalità esclusione fino al ricevimento di un segnale di esclusione da parte di un sensore.
2 – Cliccare sull'apposito pulsante di inclusione/esclusione sul sensore che si vuole escludere.
3 – Il sistema tornerà in modalità normale e il sensore verrà escluso. Automaticamente i sensori di numero maggiore subiranno un decremento di una posizione.

Tabella 3: Ordine di inclusione sensori Zwave

ORDINE di INCLUSIONE	SENSORE
1	Sensore Accessi
2	Sensore Apertura/chiusura Porte
3	Sensore centro stanza (PIR, TEMP, RH)
4	Sensore Apertura/chiusura Finestre

Il controller della rete Zwave sesto senso utilizza una periferica USB attaccata direttamente su una porta COM della Rasberry. In particolare è stata utilizzata la USB Stick Zwave.me (ZME_UZB1).

Tabella 4: Caratteristiche USB Stick Zwave.me

Marca	z-wave.me
Serie	ZME_UZB1
Colore	Black
Peso articolo	4,54 g
Dimensioni prodotto	3,5 x 1,7 x 0,8 cm
Numero modello articolo	ZME_UZB1

Figura 6: USB STICK ZWAVE

Zwave Frequenza europea: 868.4 MHz

2.1.3 Restful http API

Il sistema Sesto Senso utilizza il cloud Telegea Smart Hub che fornisce un'API HTTP che consente ai clienti di terze parti di ottenere dati di sensori in tempo reale o controllare le uscite dei dispositivi. L'autenticazione di base viene fornita utilizzando un token di sicurezza (chiave API). L'API consente di interrogare più elementi in una singola richiesta.
I comandi di lettura o scrittura su un modulo del servizio di piattaforma sul dispositivo possono essere inviati da un client nella stessa rete locale all'API tramite una richiesta HTTP GET che viene utilizzata per leggere i dati del sensore in tempo reale o per controllare direttamente un'uscita del dispositivo. Il modulo di servizio della piattaforma desiderato viene selezionato dal suo indirizzo di modulo specifico e dall'elemento all'interno del modulo dal suo indirizzo di registro.
API URL: https://my.telegea.org/input/command.php
Parameters (in URL encoded format):
apikey=<plant_apikey>
plant_id=<plant_id>
dev_cmd=<dev_cmd>

Return code: {result:[{"mod":<mod>,"reg":<reg>,"val":<val>}, ...]}
i valori che non possono essere letti o scritti sono NULL
NOT OK per operazione fallita
• apikey: token di sicurezza (parametro richiesto se è definito sullo Smart Hub)
• plant_id: ID della centralina a cui viene inviato il comando (parametro richiesto)
• dev_cmd: Elenco di comandi in formato JSON (parametro richiesto)
[{"mod": <mod>, "reg": <reg>, "val": <val>}, ...]
◦ mod: indirizzo del modulo (parametro richiesto)

Indirizzo del modulo:
8 - Modulo ZWave (sensori wireless)
10 - Sesto senso Data (parametri virtuali)
◦ reg: registra l'indirizzo all'interno del modulo specificato (parametro richiesto)
◦ val: registra il valore da scrivere (parametro opzionale), se presente verrà eseguito un comando di scrittura

Al completamento con successo dell'operazione il server risponde con un array JSON (javascript object notation) contenente tutti i valori che sono stati scritti o letti dai registri specificati.
La richiesta non riuscirà per i seguenti motivi:
• plant_id è mancante o non corrisponde
• Apikey è mancante o non corrisponde
• manca dev_cmd
• operazione di lettura / scrittura richiesta fallita
Esempio:

Richiesta remota del valore di temperatura (reg.12) dal modulo Zwave (mod.8): https://my.telegea.org/input/command.php?apikey=1234567890&plant_id=1&dev_cmd=[{"mod":8,"reg":12}]

Risposta:
{"result":[{"reg":12,"val":359,"mod":8}]}

Centraline enea attualmente utilizzate:
SESTO SENSO 001:
plant_id=168
apikey=d658305226ad26346f98f792c7c983755b45d6d0edb40

SESTO SENSO 002:
plant_id 169
api_key 359b3042f68db74884ae64eebb9a05515b9bc4b033adc

SESTO SENSO 003:
plant_id 170
api_key 0dd8eb06cae0ecb967aa18647756527b5b9bc4c852ce7

I nodi sono registrati come segue:
Node1: People counter
Node2: Fibaro Sensor2
Node3: Aeotec Multi6
Node4: Fibaro Sensor 2

NB: Per i parametri fare riferimento alla Tabella 1 dove i valori visualizzati attualmente nella centralina sono riportati in grassetto.

2.1.4 Interfaccia web

L'interfaccia web è stata realizzato in PHP (hypertext ppreprocessor) e Javascript. La pagina web utilizza le API precedentemente descritte per leggere o modificare i valori visualizzati.
Mentre i valori provenienti dai sensori vengono registrati real time dal sistema la pagina web e quindi la visualizzazione dell'evento si aggiorna ogni 5 secondi.

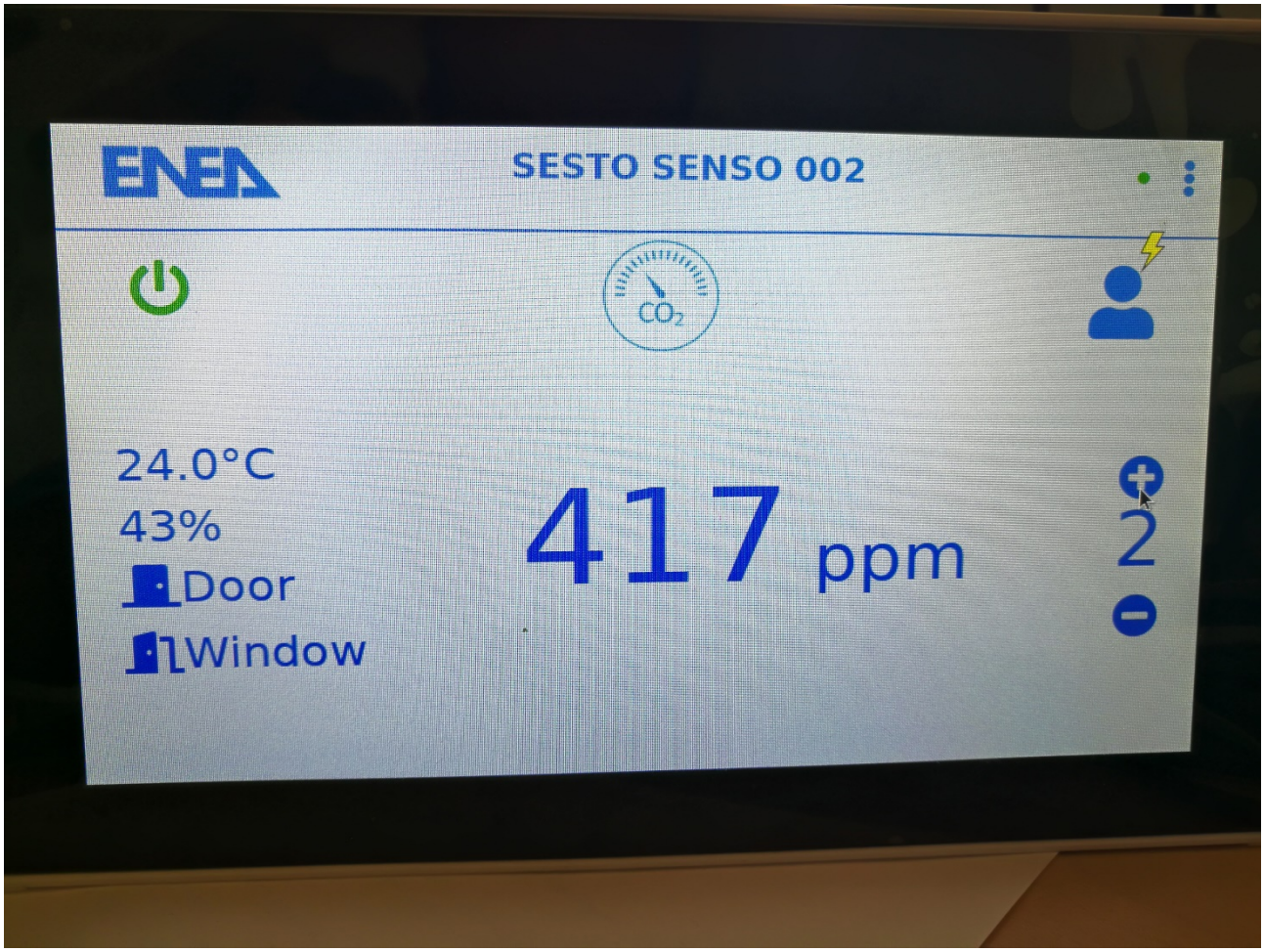


Figura 7: Interfaccia web sistema sesto senso

Sull'interfaccia web sono visualizzati tutti i dati fondamentali reali e virtuali tenuti sotto controllo dal sistema SESTO SENSO.
La temperatura espressa in gradi celsius (C°) è il risultato della media del valore di temperatura letto dal sensore di apertura e chiusura porta, apertura e chiusura finestra e del sensore centro stanza.

L'umidità relativa espressa in percentuale (%) viene rilevata dal sensore centro stanza.

L'apertura/chiusura porte e finestre (DOOR, WINDOW) viene indicata tramite un'immagine che raffigura la chiusura o l'apertura.

Il numero di persone (n) presenti nella stanza è l'unico parametro che può essere modificato dal pannello touch. Questa funzione serve per impostare il numero di persone presenti al momento dell'accensione della centralina, parametro che il sistema non è in grado di conoscere.

Il valore di CO2 virtuale calcolato viene visualizzato in ppm (parti per milione) a centro schermo. Tale valore sarà calcolato solo quando tutti i sensori necessari sono presenti, in caso contrario verrà visualizzato un valoro di default di 0ppm.

NB: Nel caso la centralina non veda i sensori, il valore corrispettivo sull'interfaccia sarà riportato in grigio invece che blu.

E' possibile cambiare il nome del sistema cliccando sulla scritta. Il nome di default è SESTO SENSO XXX (dove XXX indica un numero crescente).

Sulla schermata in alto a destra un pallino colorato indica in verde la presenza di una connessione e in rosso l'assenza di connessione.

Cliccando sul menu in alto a destra è possibile accedere all'interfaccia di configurazione.

Dall'interfaccia web di configurazione è possibile:

- Connettersi al WiFi (opzione da utilizzare nel caso non si disponga di una LAN)
- Cambiare le impostazioni dello schermo:
  - Luminosità
  - Tempo di spegnimento dello schermo per il risparmio energetico
- INCLUSIONE / ESCLUSIONE dei sensori (funzioni attualmente non abilitata)

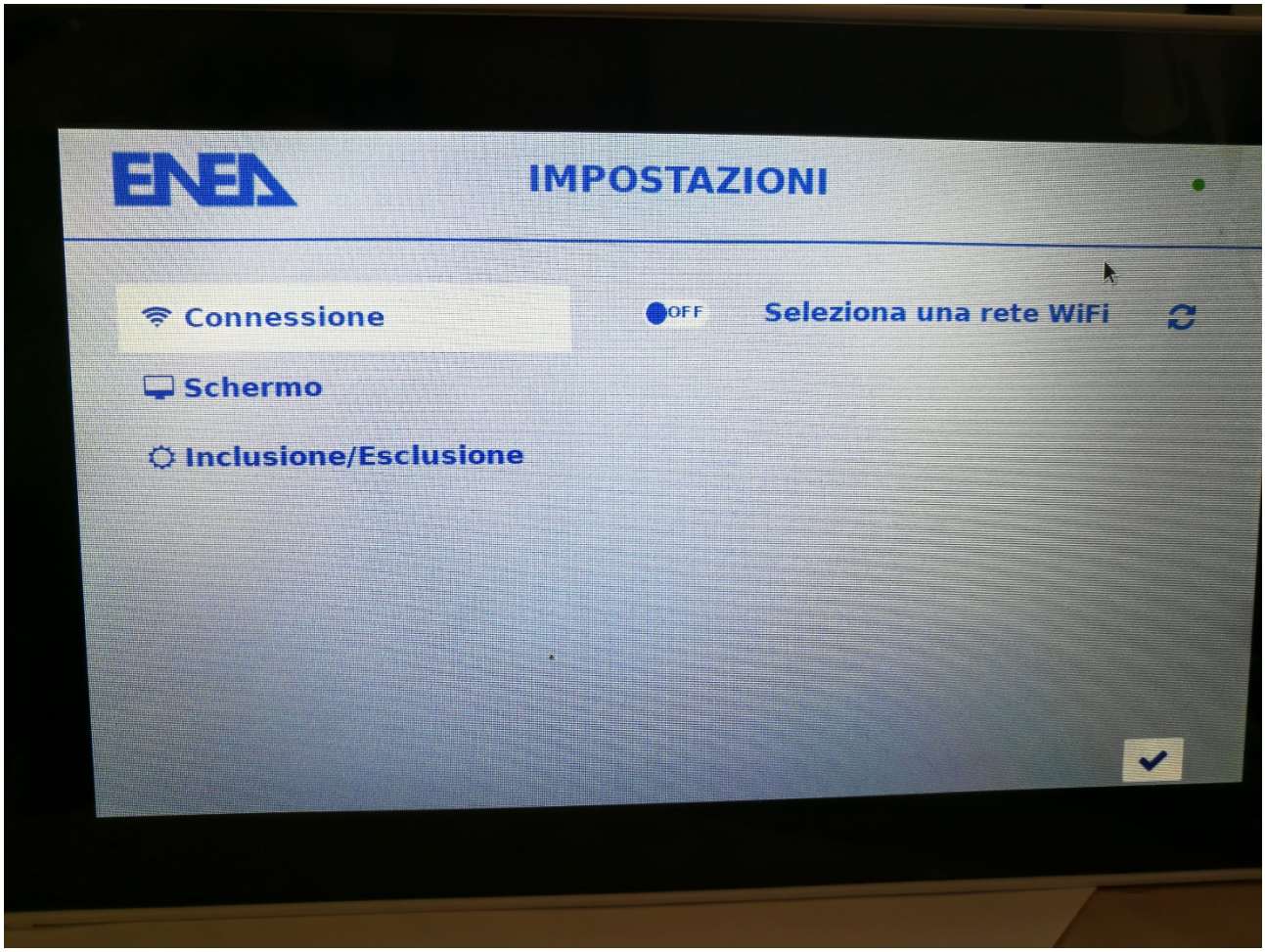


Figura 8: Interfaccia web sistema sesto senso (impostazioni)

2.2 Sensoristica accessoria

Il sistema sesto senso si basa sull'utilizzo di sensori commerciali Zwave per prelevale le informazioni dall'ambiente e compiere elaborazioni di livello superiore estrapolando parametri virtuali quali ad esempio la percentuale di anidrite carbonica.

L'algoritmo che calcola la percentuale di CO2 (descritto nel capitolo 4) ha come input 5 parametri ambientali della stanza dove è installato:

- 1 Stato di apertura porta
- 2 Stato di apertura della finestra
- 3 Umidità
- 4 Temperatura
- 5 Numero delle persone presenti nella stanza

Mentre le informazioni 1,2,3,4 sono reperibili da sensori commerciali, per il numero di persone non è presente attualmente sul mercato un sensore che soddisfi tale esigenza, si è quindi scelto di realizzarne uno ad hoc che diventa parte fondamentale del sistema sesto senso.
Per quel che riguarda la scelta dei sensori si è pensato di utilizzare dei sensori wireless Zwave essendo ormai uno standard di utilizzo molto diffuso in domotica, ambito nel quale può essere applicato il sistema sesto senso.
Scelta l'opzione Zwave anche il sensore conta persone è stato progettato per funzionare con la stessa tecnologia, facendo si che lo stesso sensore conta persone diventasse un vero sensore di domotica facente parte del sistema sesto senso ma con la possibilità di essere usato anche in altri sistemi.
La scelta di utilizzare lo standard wireless elimina inoltre la problematica di collegare i sensori direttamente alla centralina per l'invio del segnale, in questo modo è possibile posizionare la centralina in qualunque luogo all'interno della stanza; in aggiunta i sensori 1,2,3,4 hanno la possibilità di essere alimentati a batteria rendendo l'installazione ancora più semplice e meno invasiva.

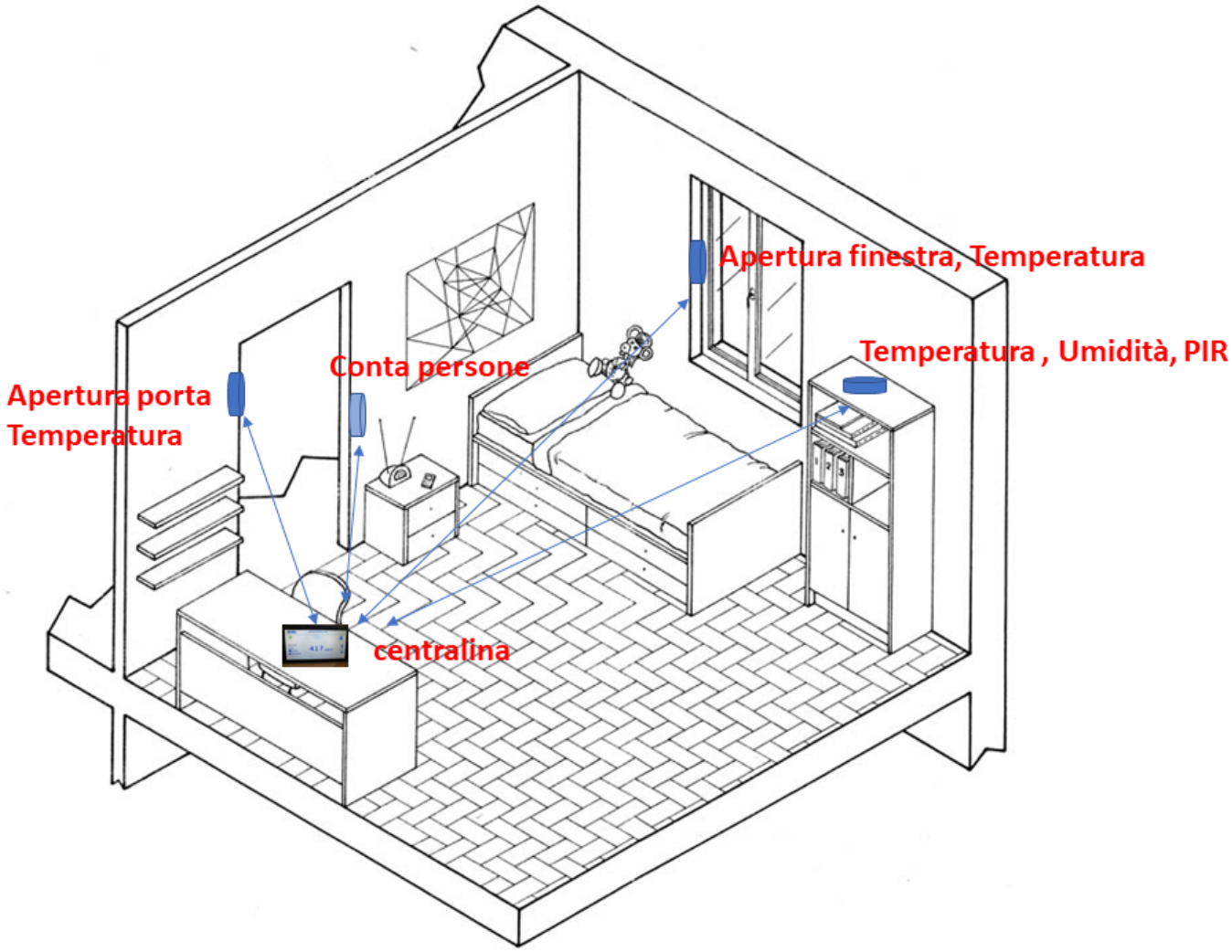


Figura 9: disposizione dei sensori in una stanza tipo

2.2.1 Sensore Porta

Il sistema SESTO SENSO accetta come sensore di apertura/chiusura porte qualsiasi sensore Zwave con uscite a stato secondo lo standard.

Tabella 5: stato-interfaccai del sensore porta

- **Segnale stato apertura: 22**	
- **Segnale stato chiusura: 23**	

Per il sistema base, dopo una rapida analisi dei sensori attualmente in commercio, è stato scelto il sensore **FIBARO Door/Window Sensor 2,** che oltre allo stato della porta fornisce anche la temperatura (informazione che viene utilizzata per mediare la temperatura nella stanza), e un sistema di rilevazione anti sabotaggio.
Il sensore che rileva lo stato di apertura e chiusura della porta deve essere incluso in seconda posizione come illustrato nel capitolo 2.2.
Il segnale di INCLUSIONE/ESCLUSIONE viene inviato premendo 3 volte velocemente sull'apposito pulsante dietro il sensore. Il sensore è alimentato tramite batteria quindi ogni volta che la centralina viene resettata il

sensore deve mandare un segnale (premere una volta sul sensore) così tornerà automaticamente visibile dalla centralina se già registrato precedentemente.

Figura 10: Sensore porta/finestra

L'installazione deve essere fatta in modo che alla chiusura della porta il magnete e il sensore siano vicini minimo 5 mm e che le stanghette riportate sul magnete e sul sensore siano allineate.

2.2.2 Sensore Finestra

Il sensore di stato delle finestre segue le stesse caratteristiche del sensore per la porta jn quanto per comodità d'uso il sensore di porta e finestra sono stati scelta uguali (**FIBARO Door/Window Sensor 2).**
Il sensore che rileva lo stato di apertura e chiusura della finestra deve essere incluso in quarta posizione come illustrato nel capitolo 2.2.1

2.2.3 Sensore centro stanza

Il sensore di centro stanza oltre a rilevare la temperatura e l'umidità deve rilevare anche la presenza di persone all'interno della stanza (solo presenza). La rilevazione di persone è un dato importate al fine di completare l'algoritmo del conta persone.
Si è deciso di mettere il sensore di presenza nel sensore di centro stanza per lasciare massima libertà di installazione al sensore porte e finestre e alla centralina.
La rilevazione di presenza viene fatta tramite un Motion Sensor PIR (Passive Infrared) il cui dato viene interpretato dalla centralina andando a correggere se necessario il valore del conta persone.
Tra i sensori commerciali è stato individuato il **Multisensor 6 della Aeotec** il quale oltre ad avere il sensore di temperatura, umidità e PIR è in grado di misurare l'intensità della luce, l'intensità dei raggi UV e le oscillazioni (allarme anti sabotaggio).

Figura 11: Sensore di centro stanza

Il sensore di centro stanza deve essere incluso in terza posizione come illustrato nel capitolo 2.2. Il segnale di inclusione/esclusione viene mandato premendo una volta il pulsante posto sul retro del sensore A differenza del sensore di porte/finestre in caso di reset della centralina il sensore non deve rimandare il segnale di inclusione/esclusione ma viene riconosciuto in automatico dalla centralina.
Il sensore di centro stanza può essere alimentato sia a batterie che via cavo USB in dotazione.

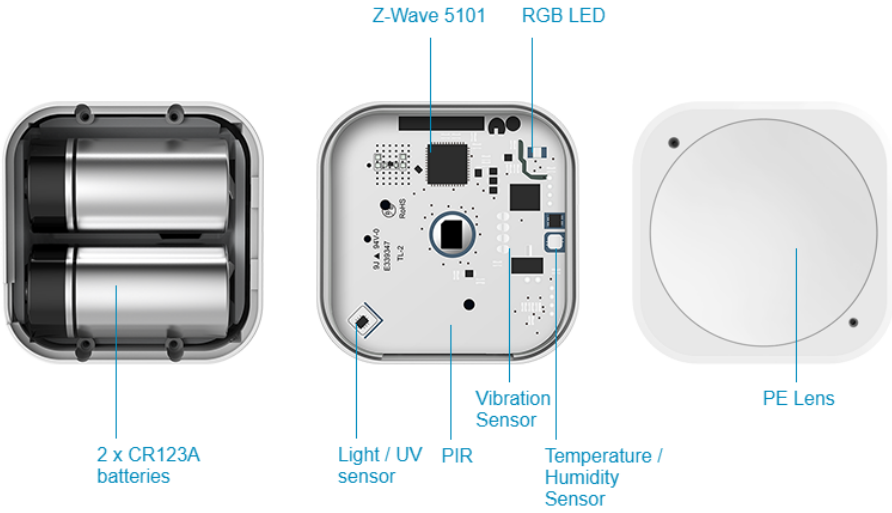


Figura 12: Sensore di centro stanza (dettaglio)

2.2.4 Sensore conta persone

Figura 13: Sensore conta persone

La progettazione del sensore conta persone è partita dal lavoro svolto precedentemente dall'ENEA il quale aveva già prodotto un prototipo che presentava alcune criticità:

- I 2 sensore utilizzati per la rilevazione presentavano delle interferenze che a volte compromettevano la misurazione
- I sensori erano stati pensati per essere installati direttamente sulle mostre delle porte, con conseguente impatto invasivo sulle stesse.
- Il sensore non essendo wireless richiedeva oltre all'alimentazione anche un collegamento diretto con la centralina per inviare il segnale

Ai fini di risolvere la criticità relativa alle interferenze è stato svolto uno studio per individuare il sensore che manifestasse di meno questo fenomeno.
Si è partito quindi dall'individuazione di tre classi di sensori:

- Sensori a infrarossi con uscita ON/OFF (utilizzati nel prototipo ENEA):
  E18-D80NK

Tabella 6 – Sensore E18-D80NK, caratteristiche

	Caratteristiche: Vin: 5 VDC I: 100 mA **Range misura: 3 - 80 cm** Diametro: 17 mm Sensore Lunghezza: 45 mm

Il sensore E18D80NK permette di regolare la distanza soglia di rilevazione tramite apposita vite posta sul retro. Per l'applicazione specifica è stata impostata una distanza soglia di 70 cm valore minimo approssimativo di passaggio in una porta.

- Sensori a infrarossi con misuratore di distanza ad uscita analogica: **GP2Y0A21YK0F**

Tabella 7 – Sensore GP2Y0A21YK0F, caratteristiche

	Caratteristiche: Vin: 4,5V - 5,5Vdc I: 30mA Range misura: 10 - 80cm DIMENSIONI 44,5X19X13,5mm

- Sensore ad Ultrasuoni: **HC-SR04**

Tabella 8 – Sensore HCSR04, caratteristiche

	Caratteristiche: Alimentatore: 5V DC; corrente di riposo: meno di 2mA; angolo di efficacia: meno di 15°; distanza: 2cm~300cm; risoluzione: 0.3 cm

Dai test effettuati è stato evidenziato che tutti i sensori hanno una buona capacità di rilevazione della presenza ma nell'utilizzo in coppia non è stato possibile evidenziare una efficienza superiore di uno dei sensori. Tutti infatti presentavano problemi di individuazione che a volte compromettono le misure.
Si è quindi deciso di cambiare approccio e concentrare l'ottimizzazione del sensore concentrandosi sugli angoli di efficienza del cono di detenzione del sensore.
Si quindi deciso di utilizzare il sensore **E18D80NK** perché già utilizzato nel prototipo e perché di più facile interpretazione a livello elettronico, inoltre la conformazione cilindrica ne permette una più facile integrazione meccanica.
Lo svantaggio principale nell'utilizzo del sensore E18D80NK sono gli ingombri e i consumi. Quest'ultimi però si possono abbassare utilizzando il sensore a impulsi invece che in continua.

Figura 14: angolo di rilevamento sensore E18D80NK

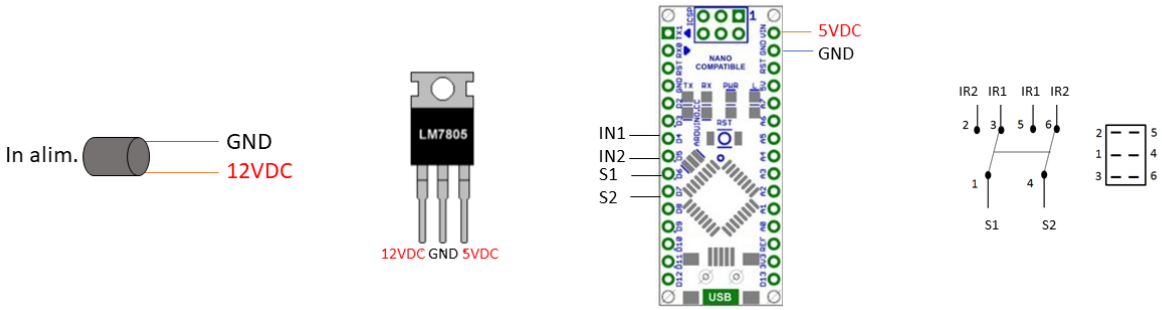
Analizzando l'angolo di rilevamento del sensore esso presenta un cono di circa 15°, ciò significa che nell'utilizzo in coppia, affinché l'algoritmo sia eseguito bene, gioca un ruolo molto importante il posizionamento dei sensori. D'altro lato posizionare i sensori troppo distanti risulta difficile dovendo essi essere posizionati su una porta e possibilmente in un unico device.
Dai test si è visto che posizionando i sensori a una distanza di circa 8 cm (massimo accettabile per un device di questo tipo) con una inclinazione di circa 10 gradi c'è un miglioramento nell'interpretazione dei segnali da parte dell'algoritmo.

Figura 15: sovrapposizione coni di rilevamento di due E18D80NK inclinati

Partendo dal risultato ottenuto è stato progettato un case che permetta da un lato di rispettare il posizionamento angolare dei sensori e dall'altro permettesse un installazione al di fuori della porta (criticità 2).
L'ultima criticità (il collegamento dei sensori con la centralina) è stato risolto rendendo il sensore wireless tramite l'utilizzo di una board Zwave. Inoltre al fine di rendere il sensore compatibile anche con altri sistemi Zwave è stato implementato l'algoritmo di rilevamento direttamente nell'elettronica di bordo del sensore.
Il risultato è un sensore conta persone wireless che invia in uscita due segnali impulsivi su due canali di cui uno rappresenta una persona in entrata mentre l'altro rappresenta una persona in uscita. Grazie a questi due segnali che vengono mandati istantaneamente alla centralina ogni volta che viene riconosciuto uno dei due eventi, la centralina, conoscendo il numero iniziale di persone nella stanza e avendo il feedback del PIR del sensore di centro stanza, può incrementare o decrementare il numero di persone.

2.2.4.1 Schema elettrico

Per realizzare il sensore sono stati utilizzati 2 sensori IR **E18D80NK** con uscita alto/basso (rilevazione/non rilevazione) collegati rispettivamente a due ingressi digitali della scheda **Arduino nano** (D5, D4) sulla quale è stato caricato il firmware che leggendo i valori di ingresso esegue l'algoritmo sviluppato dall'ENEA.
In base al risultato dell'algoritmo (ENTRATA/USCITA) vengono attivati corrispondentemente due OUTPUT digitali (D6/D7) che vanno a pilotare tramite impulsi la scheda Zwave **FIBARO UNIVERSAL SENSOR** la quale converte i segnali digitali in due segnali Zwave che la centralina interpreta come entrata o uscita.
L'alimentazione del sensore conta persone è 12VDC i quali alimentano direttamente la scheda Zwave; mentre sia i sensori IR che Arduino nano sono alimentati a una tensione di 5VDC proveniente dal convertitore DC/DC **LM7805.**
Le uscite dei sensori passano per uno switch a 6 vie interno che permette di invertire i segnali in ingresso ad Arduino. Tale Switch è stato pensato per permettere l'installazione del device sia fuori che dentro la stanza su tutte e due le pareti.



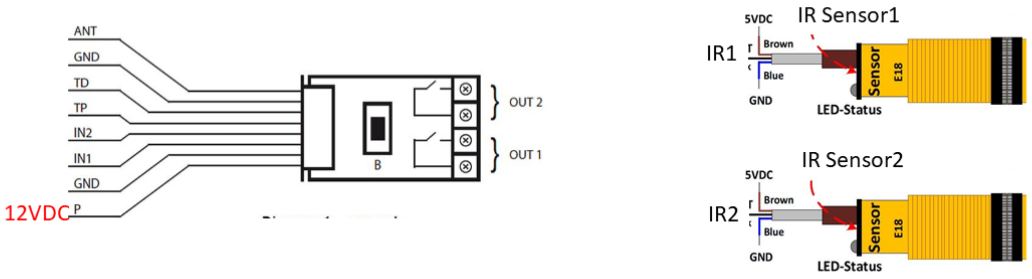


Figura 16: Componenti e schema elettrico del sensore conta persone

2.2.4.2 Firmware

Il firmware ricalca esattamente quanto sviluppato dall'ENEA con le uniche differenze che alla corrispondenza dell'entrata o dell'uscita vengono attivati corrispondentemente due uscite digitali mentre il feedback del PIR non viene eseguito dal firmware del sensore ma dalla centralina.

CODICE:

```
/* dichiaro le variabili */
int transito[8];
int persona = 1;
int statosensore1 = 0;
int statosensore2 = 0;

/* inizializzo la comunicazione seriale e definisco i pin */
void setup(){
Serial.begin(9600);
  pinMode(2,INPUT);
 pinMode(3,INPUT);
  pinMode(4,OUTPUT);
 pinMode(5,OUTPUT);
 digitalWrite(4,LOW);
 digitalWrite(5,LOW);
}

void lettura(){
statosensore1=digitalRead(2);
statosensore2=digitalRead(3);
}

void loop(){
/* conteggio persone */
start:
lettura();

//quando i sensori non rilevano passaggio lo stato è 1, quando rilevano un passaggio lo stato è 0
if (statosensore1 == 1 && statosensore2 == 1){
 //se non passa nessuno eseguo il resto del programma
 goto start;
}

if (statosensore1 == 0 || statosensore2 == 0) //inizia entrata o uscita: condizione1 10 oppure 01
{
transito[0] = statosensore1;
transito[1] = statosensore2;
}
sensorloop:
lettura();
if (statosensore1 == 1 && statosensore2 == 1){
 //passaggio vicino porta
 goto start;
```

```
}
if (statosensore1 == 0 && statosensore2 == 0) //mi trovo davanti ai due sensori: condizione2 0 0 transito in corso
{
transito[2] = statosensore1;
transito[3] = statosensore2;
}
else{ goto sensorloop;}

sensorloop2:
lettura();
if (statosensore1 == 1 || statosensore2 == 1){
// continuo il percorso in entrata o uscita, uno dei due va a 1: condizione3 1 0 oppure 0 1
transito[4] = statosensore1;
transito[5] = statosensore2;
}
else{ goto sensorloop2;}

sensorloop3:
lettura();
if (statosensore1 == 1 && statosensore2 == 1)
{
 //transito finito, si ritorna allo stato iniziale con 1 1 e faccio conteggio
transito[6] = statosensore1;
transito[7] = statosensore2;
}
else if (statosensore1 == 0 && statosensore2 == 0){
 //ripensamento del transito , tornato indietro quindi mi ritrovo con 0 0
 goto sensorloop;
 }else{
  goto sensorloop3;
  }

if ( transito[0] == 0 && transito[4] == 0){ Serial.println("********** ripensamento in uscita **********");}
if ( transito[1] == 0 && transito[5] == 0){ Serial.println("******** ripensamento in entrata **********");}
if ( transito[0] == 0 && transito[1] == 0){Serial.println("================= Attenzione: ANOMALIA analisi (doppio IN) ==========");}
if ( transito[6] == 0 || transito[7] == 0){Serial.println("================= Attenzione: ANOMALIA analisi ==========");}

// ************************** incremento persona
if ( transito[0] == 0 && transito[5] == 0) {
 Serial.println("<-<-****<-<- uscita <-<-****<-<-");
 persona=persona-1;
 digitalWrite(4,HIGH);
 }
if ( transito[1] == 0 && transito[4] == 0) {
 Serial.println("->->****->-> entrata ->->****->->");
 persona=persona+1;
```

```
 digitalWrite(5,HIGH);
 }

if ( persona < 0){Serial.println("================= Attenzione: ANOMALIA N°persone ==================");}
if ( persona < 0 ){ persona=0;}

 Serial.print("Numero persone presenti= ");
 Serial.println(persona);
 delay(100);
 digitalWrite(4,LOW);
 digitalWrite(5,LOW);
}
```

2.2.4.3 Meccanica

Il case della preserie realizzato in stampa 3d presenta due alette laterali per l'installazione a parete, due fori frontali laterali per i sensori IR e un foro posteriore per l'ingresso del jack da pannello per l'alimentazione.
Il piano frontale e removibile tramite una slitta. In questo modo è possibile accedere all'elettronico a allo switch interno per invertire i sensori.
La parete frontale laterale presenta una duplice inclinazione in modo da disporre i sensori il più distanziati possibile con un'angolatura di 10°.

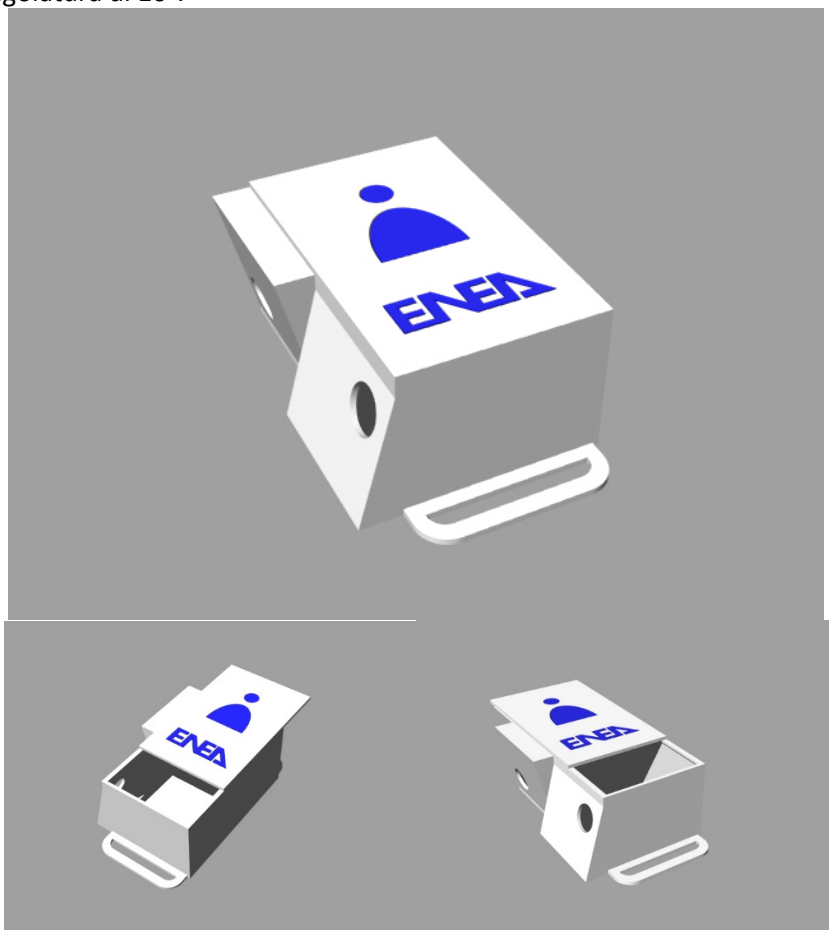


Figura 17: Render sensore conta persone

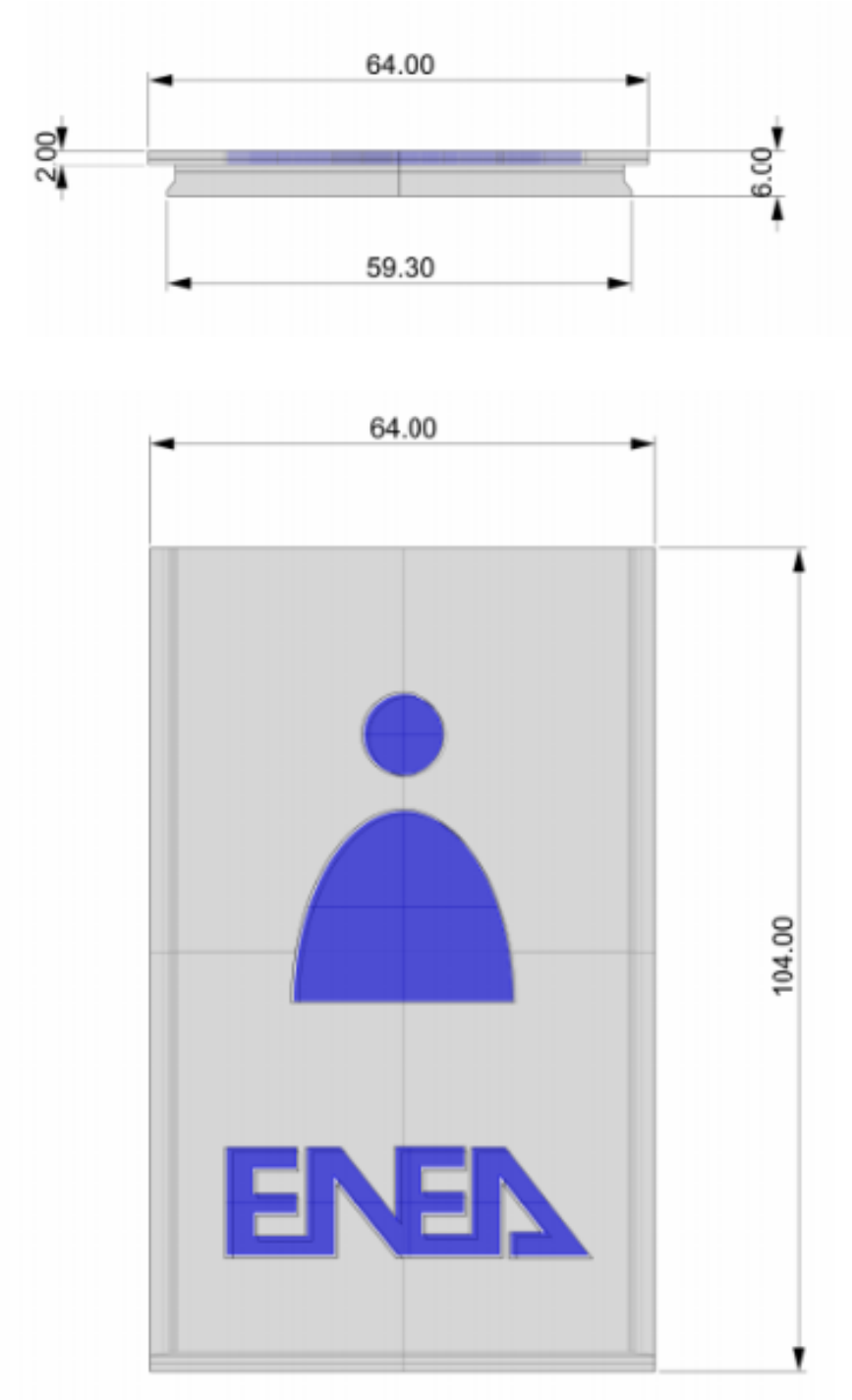


Figura 18 - COVER FRONATALE DEL SENSORE CONTA PERSONE (dimensioni)

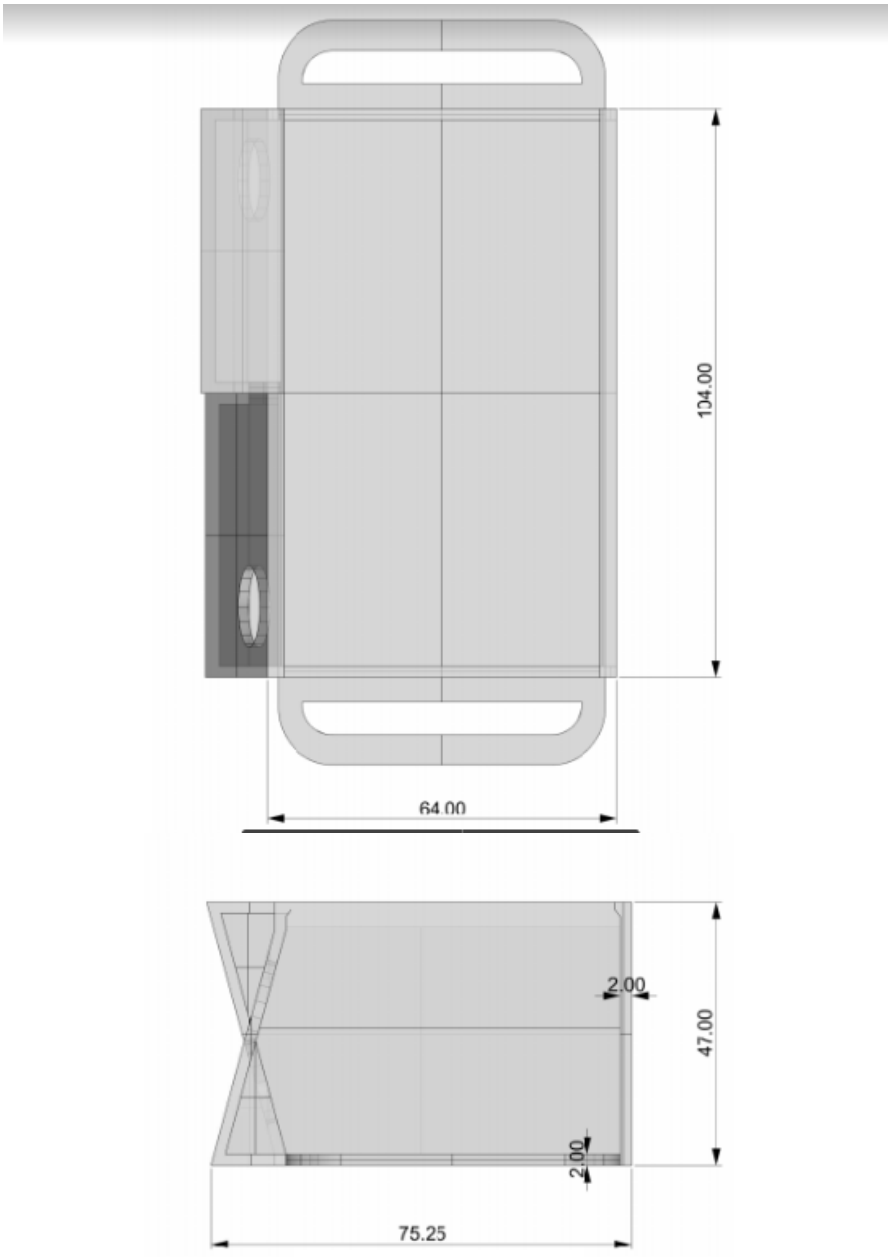


Figura 19 - CORPO DEL SENSORE CONTA PERSONE (dimensioni)

2.3 Sensore virtuale di CO2

```
#!/usr/bin/env python
##############################################################################
#
# CO2 calculator
#
# Author: Ondrej Wisniewski
#
# Description:
#
# Changelog:
#   05-09-2018: Initial version
```

```
#
# Copyright 2018, DEK Italia
#
# Uses the PyModbus Python library
# https://github.com/bashwork/pymodbus
# Install with "pip install -U pymodbus" or "apt-get install python-pymodbus"
#
#############################################################################

import math
import signal
import syslog
import sys
import time
import ConfigParser
from pymodbus.client.sync import ModbusTcpClient
from pymodbus.constants import Defaults


VERSION = "0.1"

# Override the ModbusTcpClient default settings
Defaults.Timeout = 5 # default is 3
Defaults.Retries = 1 # default is 3

# Constants
CYCLE_PERIOD = 5
CONF_FILE    = '/etc/telegea.conf'
IN_SCALE     = 10
OUT_SCALE    = 10

# Neural network constant definitions

NNHL   = 5  # numero di neuroni hidden layer
Ninput = 5  # numero di external input Neural Network

IB = [ -2.991742032554759,  0.046451616540915, -4.940594936087134,  1.691344713123214, -
4.646813915646349 ]

IW =[[ -0.561556306900530, -0.400280757271960,  0.629685338267886, -1.676464419190815,
1.066334195627757 ],
   [ -0.000745431545058, -0.001352750342335,  0.007499065322726,  0.007902790439092, -
0.009318901338443 ],
   [  0.462266606725910,  0.383982563758404,  0.016362402791320,  2.807746352032552, -
1.533096809677863 ],
   [ -0.108653674850386, -0.207865618141932,  1.778045142347343, -0.353038051167227,
0.043575216107726 ],
   [ -3.945766034394127, -4.906152135528951, -1.955735680847527,  5.153579837291587, -
1.933722244052638 ]]
```

```
OW = [ -0.591501054455276,  16.427201357411043, -1.363573156264782, -0.105715852077232, -0.089657256573455 ]
OB = -2.684000101471681

OWR =[[  6.139783231699427, -5.621830644857823 ],
    [  0.070048508462593, -0.008755724935387 ],
    [  3.076102207878104, -2.393179808067225 ],
    [ -5.902195764714204,  5.971526200012827 ],
    [  0.718856261578153,  3.559343915467826 ]]

Xmax = [ 55.7000, 29.6000, 3.0000,  1.0000,  1.0000 ]
Xmin = [ 42.3000, 26.7000, 0,      -1.0000, -1.0000 ]
ymax = 1845
ymin = 417

target_data_delayed = [-1, -1]


# Helper functions

def tansig(x):
  return 2.0/(1.0+math.exp(-2.0*x))-1.0

def NormalizedInput(MeasuredValue, Variable):
  return 2.0*(MeasuredValue-Xmin[Variable])/(Xmax[Variable]-Xmin[Variable])-1.0

def unNormalizedOutput(Value):
  return (Value+1.0)*(ymax-ymin)/2.0+ymin


############################################################
#
# Function:    Neuralpredictor()
# Description: Neural network predictor algorithm for
#          CO2 calculation from input measurments
#
#          Network input data:
#          1 Humidity
#          2 Temperature
#          3 People count
#          4 Door (-1 closed, +1 open)
#          5 Window (-1 closed, +1 open)
#
############################################################
def Neuralpredictor(Humidity, Temperature, NumOfPerson, OpenDoor, OpenWindow):

  if Humidity==None or Temperature==None or NumOfPerson==None or OpenDoor==None or OpenWindow==None:
     return None
  print (Humidity, Temperature, NumOfPerson, OpenDoor, OpenWindow)
```

```
    inp = [None]*Ninput
    L  = [None]*NNHL

    inp[0]= NormalizedInput(Humidity, 0)
    inp[1]= NormalizedInput(Temperature, 1)
    inp[2]= NormalizedInput(NumOfPerson*1.0, 2)
    inp[3]= NormalizedInput(OpenDoor, 3)
    inp[4]= NormalizedInput(OpenWindow, 4)
    #print(inp[0],inp[1],inp[2],inp[3],inp[4])

    for i in range(NNHL):
        L[i] = IB[i]
        for k in range(Ninput):
            L[i] = L[i]+IW[i][k]*inp[k]

        for k in range(2):
            L[i] = L[i]+OWR[i][k]*target_data_delayed[k]

    output = OB;
    for i in range(NNHL):
        output = output+tansig(L[i])*OW[i]

    target_data_delayed[1] = target_data_delayed[0]
    target_data_delayed[0] = output

    return unNormalizedOutput(output)


############################################################
#
# Function:    on_sigterm()
# Description: signal handler for the TERM and INT signal
#
############################################################
def on_sigterm(signum, frame):
    syslog.syslog(syslog.LOG_NOTICE, "Exiting CO2 Calculator daemon")
    sys.exit()


############################################################
#
# Function:    read_config()
# Description: reads parameters from the
#              configuration file
# Parameters:  None
# Return:      True if success, False otherwise
#
############################################################
def read_config():

   # Parameters from global config file
```

```
  config = ConfigParser.ConfigParser()
  config.read(CONF_FILE)

  try:
    read_config.register_list_file = config.get('generic', 'MODBUS_REGISTER_LIST').strip('"')
  except ConfigParser.NoSectionError:
    return False

  # Application parameters
  try:
    read_config.ip_addr    = config.get('co2calc', 'CO2_DEVICE_IP').strip('"')
    read_config.h_reg_name = config.get('co2calc', 'CO2_H_REG_NAME').strip('"').split(";")[0]
    read_config.t_reg_name = config.get('co2calc', 'CO2_T_REG_NAME').strip('"').split(";")[0]
    read_config.p_reg_name = config.get('co2calc', 'CO2_P_REG_NAME').strip('"').split(";")[0]
    read_config.d_reg_name = config.get('co2calc', 'CO2_D_REG_NAME').strip('"').split(";")[0]
    read_config.w_reg_name = config.get('co2calc', 'CO2_W_REG_NAME').strip('"').split(";")[0]
    read_config.c_reg_name = config.get('co2calc', 'CO2_C_REG_NAME').strip('"').split(";")[0]
  except ConfigParser.NoOptionError:
    syslog.syslog(syslog.LOG_ERR, "ERROR - Needed parameter not found in config file")
    return False

  # DEBUG
  print "register_list_file: " + read_config.register_list_file
  print "device_ip: " + read_config.ip_addr
  print "h_reg_name: " + read_config.h_reg_name
  print "t_reg_name: " + read_config.t_reg_name
  print "p_reg_name: " + read_config.p_reg_name
  print "d_reg_name: " + read_config.d_reg_name
  print "w_reg_name: " + read_config.w_reg_name
  print "c_reg_name: " + read_config.c_reg_name

  return True


############################################################
#
# Function:    get_config_param()
# Description: reads a parameter from the register
#              configuration file
# Parameters:  pname - parameter name
#              idx   - parameter index
#
############################################################
def get_config_param(reglist_file, pname, idx):

  try:
    with open(reglist_file) as f:
      for line in f:
        if pname in line:
          param = line.split()
          return int(param[idx])
```

```
  except IOError:
    syslog.syslog(syslog.LOG_ERR, "ERROR opening file " + reglist_file)

  return None


#############################################################
#
# Function:    get_reg_addr()
# Description:
# Parameters:
# Return:
#
#############################################################
def get_reg_addr(reglist_file, reg_name, ip_addr):

   if reg_name and (reg_name != ""):
      mod_addr = get_config_param(reglist_file, reg_name, 1)
      reg_addr = get_config_param(reglist_file, reg_name, 2)
      addr = {'mod':mod_addr, 'reg':reg_addr, 'ip':ip_addr}
   else:
      addr = None

   return addr


#############################################################
#
# Function:    read_input()
# Description: reads input data from Modbus slave
# Parameter:   addr - Module and register address
# Return:      input value
#
#############################################################
def read_input(addr):

  # Sanity check
  if addr == None or addr['ip'] == None or addr['mod'] == None or addr['reg'] == None:
    return None

  ip_addr  = addr['ip']
  tcp_port = 5000 + addr['mod']

  # Call modbustcp client to read current register value
  client = ModbusTcpClient(host=ip_addr, port=tcp_port)
  if client.connect() == False:
    syslog.syslog(syslog.LOG_ERR, "Connection to Modbus slave failed")
    return None

  reply = client.read_holding_registers(address=addr['reg'], unit=addr['mod'])
  client.close()
```

```
  if reply == None:
    syslog.syslog(syslog.LOG_ERR, "No reply while reading Modbus register ("+str(addr['mod'])+"/"+str(addr['reg'])+")")
    return None

  if reply.function_code != 3:
    syslog.syslog(syslog.LOG_ERR, "Reading Modbus register returned wrong function code ("+str(addr['mod'])+"/"+str(addr['reg'])+")")
    return None

  return reply.registers[0]


############################################################
#
# Function:   write_output()
# Description: writes output data to Modbus slave
# Parameter:   addr - Module and register address
#              val  - output value
# Return:     none
#
############################################################
def write_output(addr, val):

  # Sanity check
  if addr == None or addr['ip'] == None or addr['mod'] == None or addr['reg'] == None or val == None or val < 0:
    return

  ip_addr  = addr['ip']
  tcp_port = 5000 + addr['mod']

  # Call modbustcp client to write register value
  client = ModbusTcpClient(host=ip_addr, port=tcp_port)
  if client.connect() == False:
    syslog.syslog(syslog.LOG_ERR, "Connection to Modbus slave failed")
    return

  reply = client.write_register(address=addr['reg'], value=val, unit=addr['mod'])
  client.close()

  if reply == None:
    syslog.syslog(syslog.LOG_ERR, "No reply while writing Modbus register ("+str(addr['mod'])+"/"+str(addr['reg'])+")")
    return

  if reply.function_code != 6:
    syslog.syslog(syslog.LOG_ERR, "Writing Modbus register failed ("+str(addr['mod'])+"/"+str(addr['reg'])+")")
```

```
  return


############################################################
#
# Main program
#
############################################################

# Init syslog
syslog.openlog("co2calc", syslog.LOG_PID|syslog.LOG_CONS, syslog.LOG_USER)
syslog.syslog(syslog.LOG_NOTICE, "Starting CO2 Calculator daemon (version "+VERSION+")")

# Init signal handler
signal.signal(signal.SIGINT, on_sigterm)
signal.signal(signal.SIGTERM, on_sigterm)

# Read config
if read_config():

  # Get Modbus addresses for input data (H, T, P, D, W)
  h_addr = get_reg_addr(read_config.register_list_file, read_config.h_reg_name, read_config.ip_addr)
  t_addr = get_reg_addr(read_config.register_list_file, read_config.t_reg_name, read_config.ip_addr)
  p_addr = get_reg_addr(read_config.register_list_file, read_config.p_reg_name, read_config.ip_addr)
  d_addr = get_reg_addr(read_config.register_list_file, read_config.d_reg_name, read_config.ip_addr)
  w_addr = get_reg_addr(read_config.register_list_file, read_config.w_reg_name, read_config.ip_addr)

  # Get Modbus addresses for output data (CO2)
  c_addr = get_reg_addr(read_config.register_list_file, read_config.c_reg_name, read_config.ip_addr)

  syslog.syslog(syslog.LOG_NOTICE, "H sensor is at mod "+str(h_addr['mod'])+", reg "+str(h_addr['reg']))
  syslog.syslog(syslog.LOG_NOTICE, "T sensor is at mod "+str(t_addr['mod'])+", reg "+str(t_addr['reg']))
  syslog.syslog(syslog.LOG_NOTICE, "P sensor is at mod "+str(p_addr['mod'])+", reg "+str(p_addr['reg']))
  syslog.syslog(syslog.LOG_NOTICE, "D sensor is at mod "+str(d_addr['mod'])+", reg "+str(d_addr['reg']))
  syslog.syslog(syslog.LOG_NOTICE, "W sensor is at mod "+str(w_addr['mod'])+", reg "+str(w_addr['reg']))
  syslog.syslog(syslog.LOG_NOTICE, "C value is at mod "+str(c_addr['mod'])+", reg "+str(c_addr['reg']))

  # Process loop
  while True:

    # Read input data
    H = read_input(h_addr)
    if H != None:
      H = H/float(IN_SCALE)
    T = read_input(t_addr)
    if T != None:
      T = T/float(IN_SCALE)
    P = read_input(p_addr)
    if P != None:
      P = max(0,P)
    D = read_input(d_addr)
```

```
    if D != None:
      D = -1 if D==22 else 1
    W = read_input(w_addr)
    if W != None:
      W = -1 if W==22 else 1

    # Calculate output (CO2)
    C = Neuralpredictor(H, T, P, D, W)
    print(C)
    if C != None:
      C = max(0,C)
      write_output(c_addr, round(C*OUT_SCALE))

    # Wait for next polling cycle
    time.sleep(CYCLE_PERIOD)

else:
  syslog.syslog(syslog.LOG_ERR, "No valid configuration found, exiting")
```

3 Conclusioni

Il lavoro svolto ha prodotto una preserie industriale del sistema completo SESTO SENSO che rispecchia tutti gli obiettivi prefissati a inizio progetto.

Il risultato è una centralina con sensoristica accessoria wireless che permette al sistema SESTO SENSO di diventare un vero e proprio HUB di domotica basato su tecnologia Zwave connesso alla rete.

Il sistema integra i risultati, secondo lo stato dell'arte, delle ricerche svolte dall'ENEA in collaborazione con l'Università Roma Tre per il calcolo virtuale della CO2 e presenta tra la proprio sensoristica accessoria un sensore conta persone ottimizzato e reso wireless che integra gli algoritmi di calcolo sviluppati dall'ENEA.

Il sensore conta persone dotato di connessione Zwave può essere considerato da solo un sensore di domotica pre-commerciale che può quindi essere integrato anche in altri sistemi, diversi da SESTO SENSO.

Lo stesso sistema SESTO SENSO grazie al sistema di API http Restful può essere integrato in altri sistemi di domotica più grandi o connesso a sistema Cloud di terze parti.

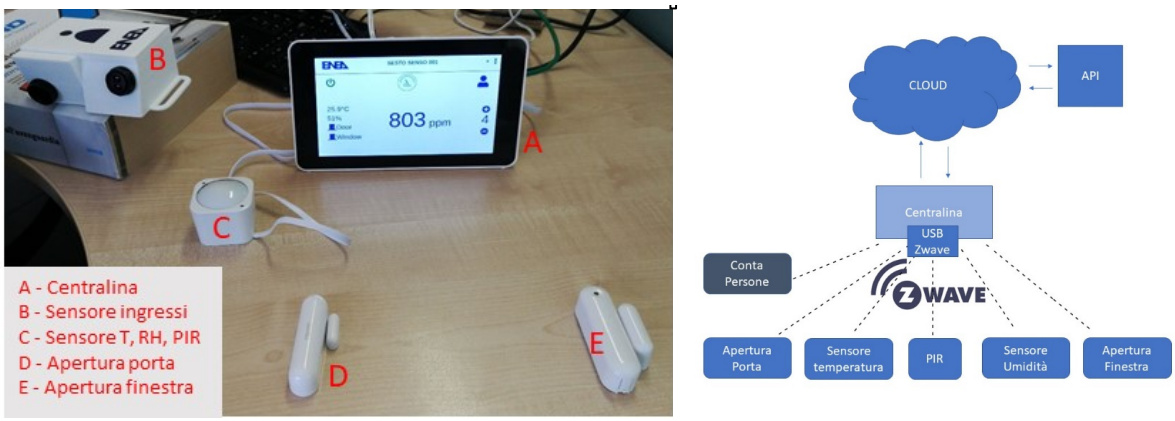


Figura 20 – Sistema sesto senso, device e architettura software

In previsione dei prossimi sviluppi, la centralina presenta la possibilità di aggiungere ulteriori sensori Zwave di diversa natura o che usino tecnologie di comunicazione anche differenti come per esempio I2C, BLE, MODBUS, WIFI, input ANALOGICI o DIGITALI ecc in modo da permettere massima libertà di sviluppo per le espansioni previste dal progetto.
Ad ogni canale comunicativo è stato riservato un registro nel quale vengono registrare i parametri e dal quale i programmi derivanti dagli algoritmi sviluppati dal team di ricerca prenderanno le informazioni per elaborare i dati al fine di calcolare ulteriori valori virtuali come fatto per la CO2 o per il contapersone.
In automatico i valori dei sensori attuali e futuri e i valori calcolati sono resi disponibili nelle API locali per la visualizzazione sul touchscreen della centralina e da remoto per gli applicativi lato web.
Grazie quindi alle predisposizioni fatte e il sistema di API il sistema SESTO SENSO diventa un vero e proprio strumento di sviluppo nel quale è possibile aggiungere e testare facilmente le tecnologie che si andranno a sviluppare.

4 Riferimenti bibliografici

- Autori: Mauro Olivieri ; Titolo: Elementi di progettazione dei sistemi VLSI -Introduzione all'elettronica digitale , Edizione EdiSES anno 2014
- Autori: Jacob Millman, Christos C. Halkias ; Titolo: Dispositivi e circuiti elettronici, Edizione Boringhieri anno1975
- Autori: Sedra Smith ; Titolo: Circuiti per la micro elettronica , Edizione EdiSES anno 2013
- Autori: Roberto Cusani; Titolo: Teoria dei Segnali, Edizione Ingegneria 2000, anno 1999
- Autori: P. Clerici Maestosi, L. Luccarini, S. Pizzuti, F. Romanello, S. Romano, A. Zanela; Titolo: Gestione energetica Smart Home e assisted living: sviluppo di sistemi e soluzioni integrate Ricerca di Sistema Elettrico 2017
- Autori: Tiziana Marsella, Romano Lombardi; Titolo: Elementi Base del linguaggio di programmazione di Arduino
- Autori: Thyagaraju Damarla, Asif Mehmood, James Sabatier; Titolo: Detection of people and animals using non-imaging sensors 14th Intenational Conference on Information Fusion Chicago, Illinois, USA, july 5-8 2011

5 Abbreviazioni ed acronimi

- CO2 (diossido di carbonio)
- Zwave: Protocollo di comunicazione wireless
- REST, RESTful (Representational state transfer)
- HTTP (Hyper text transfer protocol)
- PIR (passive infra red)
- API (application programming interface)
- Raspberry pi 3: scheda elettronica single board computer.
- Raspbian: sistema operativo
- ABS (Acrilonitrile Butadiene Stirese)
- LCD (Liquid Crystal display)
- ARM: Famiglia di microprocessori RISC a 32-bit
- MQTT (message queue telemetry transport)
- MODBUS-TCP: Protocollo di comunicazione seriale a livello applicativo basato su registri
- JSON (javascript object notation)
- PHP (hypertext ppreprocessor) linguaggio di scripting interpretativo
- Javascript: linguaggio di scripting orientato agli oggetti.
- ppm (parti per milione)
- Python: linguaggio di programmazione ad alto livello orientato agli oggetti.

APPENDICE:

Il team che ha svolto il lavoro è composto da 4 persone appartenenti a Innosensor srl e Dek italia srl (partner di innnosensor):

- **Francesca Dini** (CEO di Innosensor)
  Attività svolta: si è occupata di curare gli aspetti amministrativi, contabili e reportistici del lavoro.
  Breve curriculum: Laureata in Ingegneria Medica presso l'Università di Roma Tor Vergata inizia da subito la sua attività di ricerca presso l'Università di Linkoping in Svezia e il gruppo Sensori dell'Università di Roma Tor Vergata dove consegue il Dottorato di ricerca in "Ingegneria dei sistemi sensoriali e di apprendimento".
  Dopo il dottorato, prosegue l'attività di ricerca presso il gruppo Sensori dove affina le proprie conoscenze e competenze.
  I risultati delle ricerche portano a numerose pubblicazioni su riviste scientifiche e partecipazioni a conferenze e progetti nazionali e internazionali.
  Nel Maggio 2012 partecipa come socio fondatore alla costituzione di Innosensor, mettendo a disposizione della società la propria esperienza e professionalità nell'ambito della ricerca.
  Attualmente lavora inoltre presso la Alien Technology Transfer UK dove ricopre il ruolo di Quality Manager and Director Italy.

- **Alberto Nisti** (Ingegnere presso Innosensor)
  Attività svolta: si è occupato di progettare e coordinare tutte le fasi di sviluppo e ha realizzato e programmato il sensore conta persone e tutte le parti hardware.
  Breve curriculum: Laureato in Ingegneria Medica presso l'Università di Roma Tor Vergata, inizia da subito la propria esperienza nell'ambito della ricerca applicata, collaborando con Università e aziende private per lo sviluppo di sensori innovativi e sistemi multi-sensoriali. Promotore della fondazione della Innosensor srl raggiunge il proprio obiettivo nel Maggio 2012 mettendo a disposizione della società le proprie conoscenze nell'ambito della ricerca applicata e progettazione hardware.
  Attualmente lavora inoltre presso la Greenspider gbmh dove ricopre il ruolo di CTO (Chef Technical Officer).

- **Ondrej Wisniewski** (ingegnere presso Dek italia)
  Attività svolta: si è occupato della programmazione della centralina.
  Breve curriculum: Ondrej ha una vasta esperienza nella progettazione di sistemi embedded nel settore delle telecomunicazioni e dell'automazione.
  Attualmente è impegnato nello sviluppo di soluzioni di virtualizzazione e tecnologie cloud. In particolare sta contribuendo al progetto di community di software cloud OpenStack e alla sua implementazione in diverse aree industriali.
  Ha guidato lo sviluppo della piattaforma "Telegea" per il controllo e il monitoraggio dell'energia termica mirata al mercato degli edifici sostenibili ad alta efficienza energetica.
  In passato, ha anche fatto un'esperienza importante come ingegnere di sistemi per applicazioni di telecomunicazione nel reparto R & D di Ericsson. In questa posizione ha avuto la responsabilità di tutte le fasi del ciclo di vita tipico di un progetto di sistemi embedded, implementato nelle reti di telecomunicazione.

- **Marco Romano** (sviluppatore web presso DEk)
  Attività svolta: ha realizzato l'interfaccia web locale della centralina.
  Breve curriculum: Professionista nel settore informatica e servizi. Lavora presso Dek italia dove si è occupato di Sviluppi Android, Web e gestione server.