UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

Corso di Laurea Magistrale in
Ingegneria Elettronica

Programmazione di Sistemi IoT

" Smart Indoor Environmental Monitoring and Safeting System "

Docente
Raffaele Gravina

Studente
Giuseppe Pirilli 237909

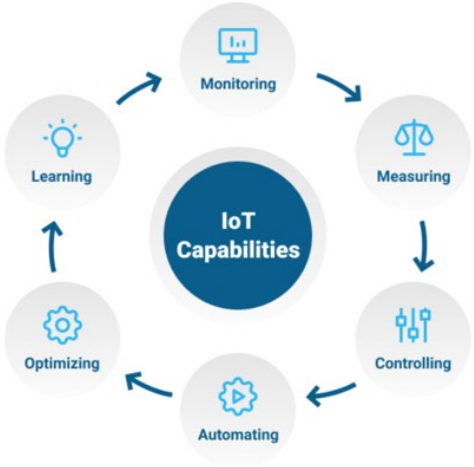
Anno Accademico 2023/2024

Indice

Introduzione

Cosa si intende per Internet of Things

- La definizione di IoT fa riferimento ad una rete globale di dispositivi cosiddetti '$smart$' che sfruttano le tecnologie wireless per comunicare tra di loro, ovunque essi si trovano, grazie alla rete internet, attuando eventualmente azioni automatiche in funzione dei dati raccolti.
- Caratteristiche di un dispositivo smart:
  - Dotato di un ID univoco
  - Connesso alla rete per trasmettere e ricevere informazioni
  - Capace di interagire con l'ambiente ed elaborare dati
- La capacità di comunicazione wireless consente la trasmissione ed archiviazione dei dati su cloud per elaborazioni avanzate, monitoraggio, analisi e controllo da remoto, utilizzando servizi di $cloud\ computing$.



Introduzione

SMART HOME AND BUILDING
- Intrusion detection system
- Monitoring and control of internal building environment
- Structural health of buildings and bridges

SMART HEALTHCARE
- Optimize patient care
- Wearable fitness devices
- Quality data reporting

SMART RETAIL
- Theft protection
- Inventory control
- Intelligent shopping applications (focused marketing)

IoT Uses By Industry

SMART ENVIRONMENT
- Forest fire detection
- Weather prediction
- Animal tracking
- Indicator for natural calamities
- Precision agriculture (microclimate monitoring, smart irrigation, ect.)

SMART GRID
- Monitoring and management of consumption

SMART CITY
- Traffic congestion control
- Connected cars
- Smart roads
- Lighting and parking management

SMART CAR
- Vehicle auto-diagnosis
- Optimized traffic flow

SMART INDUSTRY
- Machine-to-machine communication
- Machine health monitoring
- Quality control
- Indoor air quality
- indoor location of equipment

Idea progettuale

- Negli ultimi anni si è riscontrato un notevole incremento di incendi domestici che hanno causato gravi danni, tra cui intossicazione da monossido di carbonio e, in casi estremi, la perdita di vite umane.

- Alcune tra le cause di innesco di incendio negli ambienti di civile abitazione sono:
  - Disattenzioni;
  - Camino e/o canna fumaria;
  - Funzionamento difettoso di impianti e/o elettrodomestici.

- Perdite di gas combustibile, distribuito attraverso reti o in bombole possono potenzialmente innescare un incendio in seguito ad una sua deflagrazione.

Idea progettuale

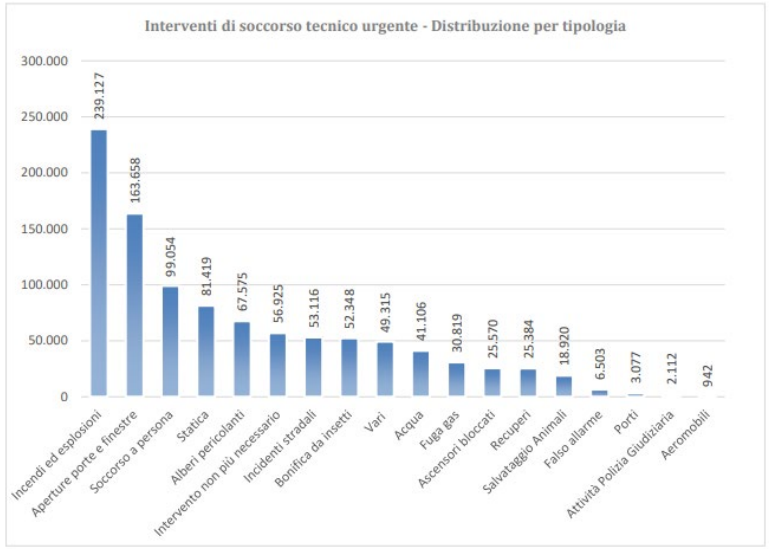


Figura 3

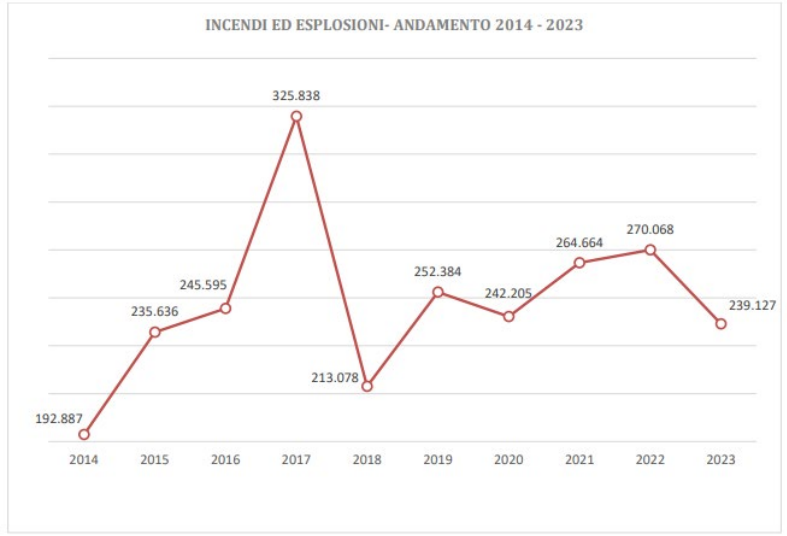


Figura 7

Idea progettuale

Interventi di soccorso tecnico urgente effettuati dal C.N.VV.F. inerenti ai luoghi con frequenza ≥ 0.2% rispetto al totale degli «incendi ed esplosioni»			
LUOGO	DETTAGLIO LUOGO	INCEDI ED ESPLOSIONI	
		N°	% su totale
Altri luoghi	Altre	2628	1,10%
	Non considerato	1504	0,63%
	Fiumi, corsi d'acqua, zone fluviali	825	0,35%
	Zone costiere (di mare)	649	0,27%
	Cantieri edili	494	0,21%
Ambienti ad uso particolare	Altri	1364	0,57%
	Scuole di ogni ordine e grado	773	0,32%
	Ospedali / case di cura / poliambulatori	605	0,25%
	Strutture ricettive turistico alberghiere	529	0,22%
Ambienti e luoghi di civile abitazione	Appartamenti e locali di abitazione	38016	15,90%
	Edifici in genere	10282	4,30%
	Altri	5074	2,12%
	Autorimesse private	2192	0,92%
	Campi nomadi	848	0,35%
	Locali quadri elettrici	801	0,33%
	Costruzioni provvisorie (dormitori di operai, baracche, ecc.)	730	0,31%
Depositi di combustibili solidi	Depositi di foraggi, paglia e simili	903	0,38%
	Depositi di rifiuti	819	0,34%
Esercizi commerciali	Ristoranti, mense e simili	1185	0,50%
	Altri	663	0,28%
Località agricole o per allevamento	Campi	30495	12,75%
	Zone rurali	20025	8,37%
	Boschi	6408	2,68%
	Altre	2870	1,20%
	Zona alberata	2442	1,02%
	Fabbricati agricoli	1502	0,63%
	Capannoni	1226	0,51%
Luogo non definito	Non definito	13994	5,85%
Zona di sosta e traffico	Strade e/o piazze cittadine	39861	16,67%
	Strade extraurbane	19743	8,26%
	Autostrade e tangenziali	4756	1,99%
	Cortili	2895	1,21%
	Parcheggi all'aperto	2550	1,07%
	Giardini	1907	0,80%
	Altre	646	0,27%
	Sedi ferroviarie	520	0,22%
Zone di montagna in genere	Altre	487	0,20%

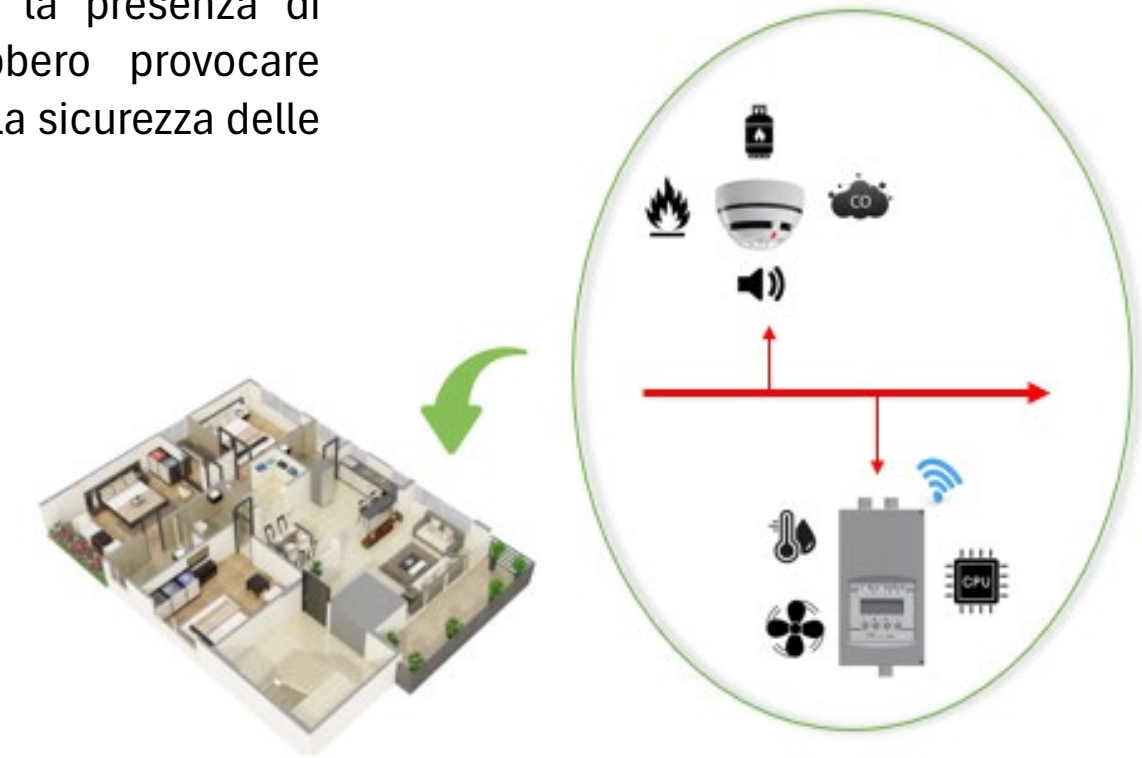
Tabella 1

Interventi di soccorso tecnico urgente effettuati dal C.N.VV.F. inerenti alle cause con frequenza ≥ 0.2% rispetto al totale degli «incendi ed esplosioni»			
CAUSA	DETTAGLIO CAUSA	INCENDI ED ESPLOSIONI	
		N°	%
Cause che determinano altri tipi di interventi	Altre	4727	1,98%
	Cause impreviste	2634	1,10%
	Disattenzione generale	2439	1,02%
Cause che determinano soccorso a persone	Non potute accertare nell'immediatezza dell'evento	820	0,34%
Cause di innesco di incendio	Altre	14546	6,08%
	Cause elettriche	11895	4,97%
	Camino e/o canna fumaria	10225	4,28%
	Mozzicone di sigaretta e fiammiferi	2823	1,18%
	Non corretta o mancata adozione di misure precauzionali, di esercizio e di sicurezza	1707	0,71%
	Autocombustione	1567	0,66%
	Surriscaldamento di motori e macchine varie	1546	0,65%
	Elettrodomestici (TV, Lavatrice, lavastoviglie, Computer, ecc.)	938	0,39%
Dolose	Probabile colpa	2784	1,16%
	Probabile dolo	10241	4,28%
Non considerato	Non considerato	5733	2,40%
Non potute accertare nell'immediatezza dell'evento	Non potute accertare nell'immediatezza dell'evento	144717	60,52%
Non definita	Non definita	14075	5,89%

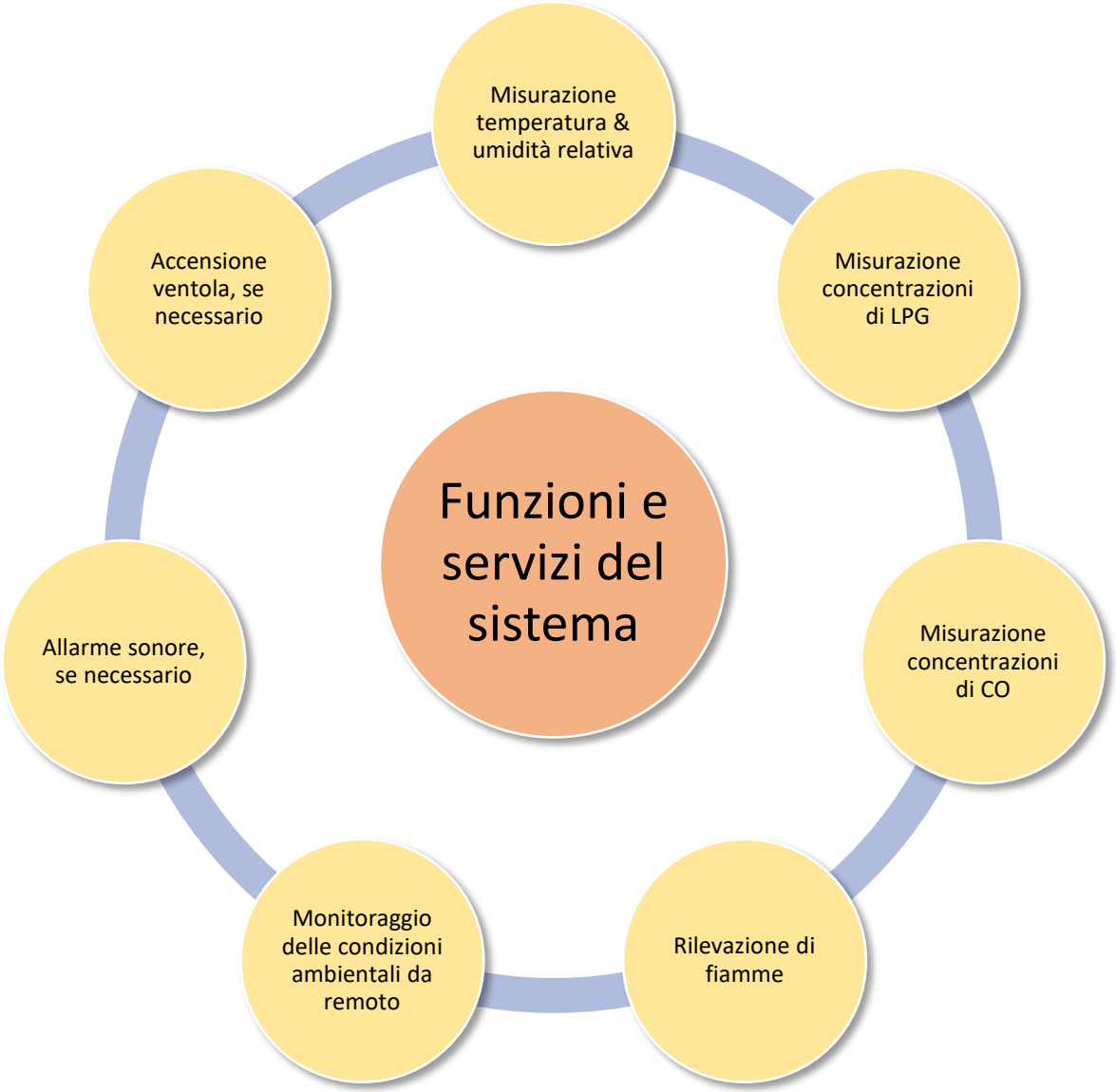
Tabella 2

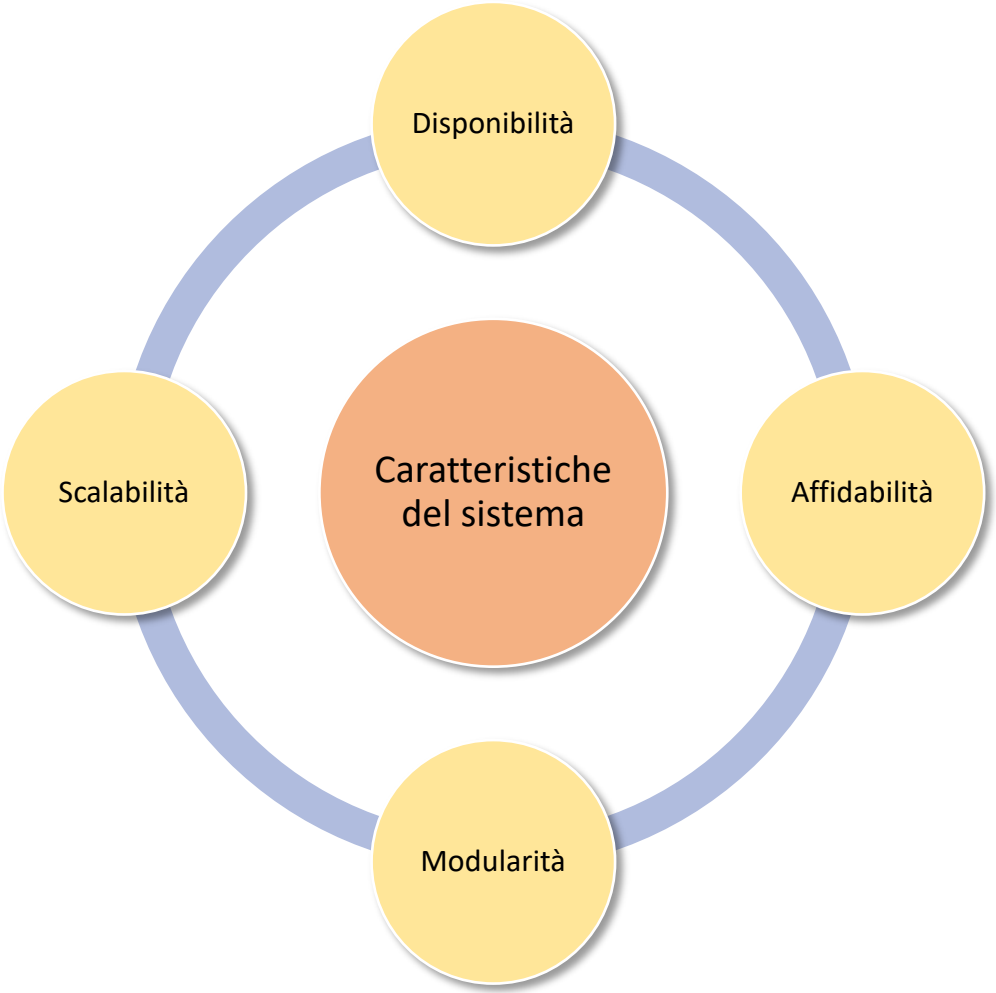
Idea progettuale

- **<u>Obiettivo</u>:** sviluppo di un sistema intelligente, connesso alla rete, per il monitoraggio da remoto e la sicurezza di ambienti domestici, misurando parametri quali: temperatura, umidità relativa, presenza di fiamme, gas pericolosi e/o sostanze tossiche (LPG e CO), in modo da rilevare tempestivamente, nella loro fase iniziale, la presenza di incendi, fughe di gas infiammabili che potrebbero provocare esplosioni e sostanze tossiche, in modo da garantire la sicurezza delle persone.



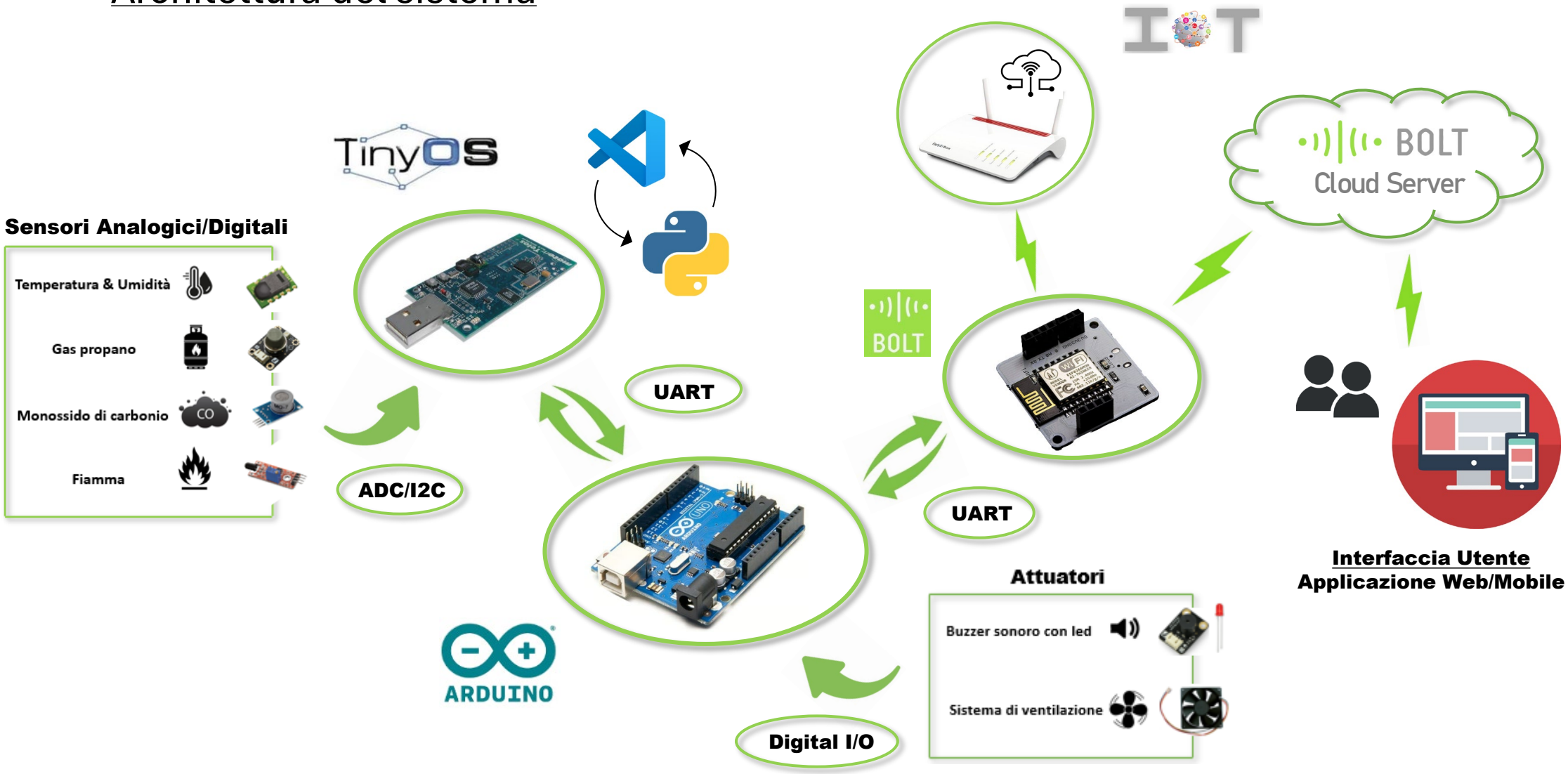
Idea progettuale





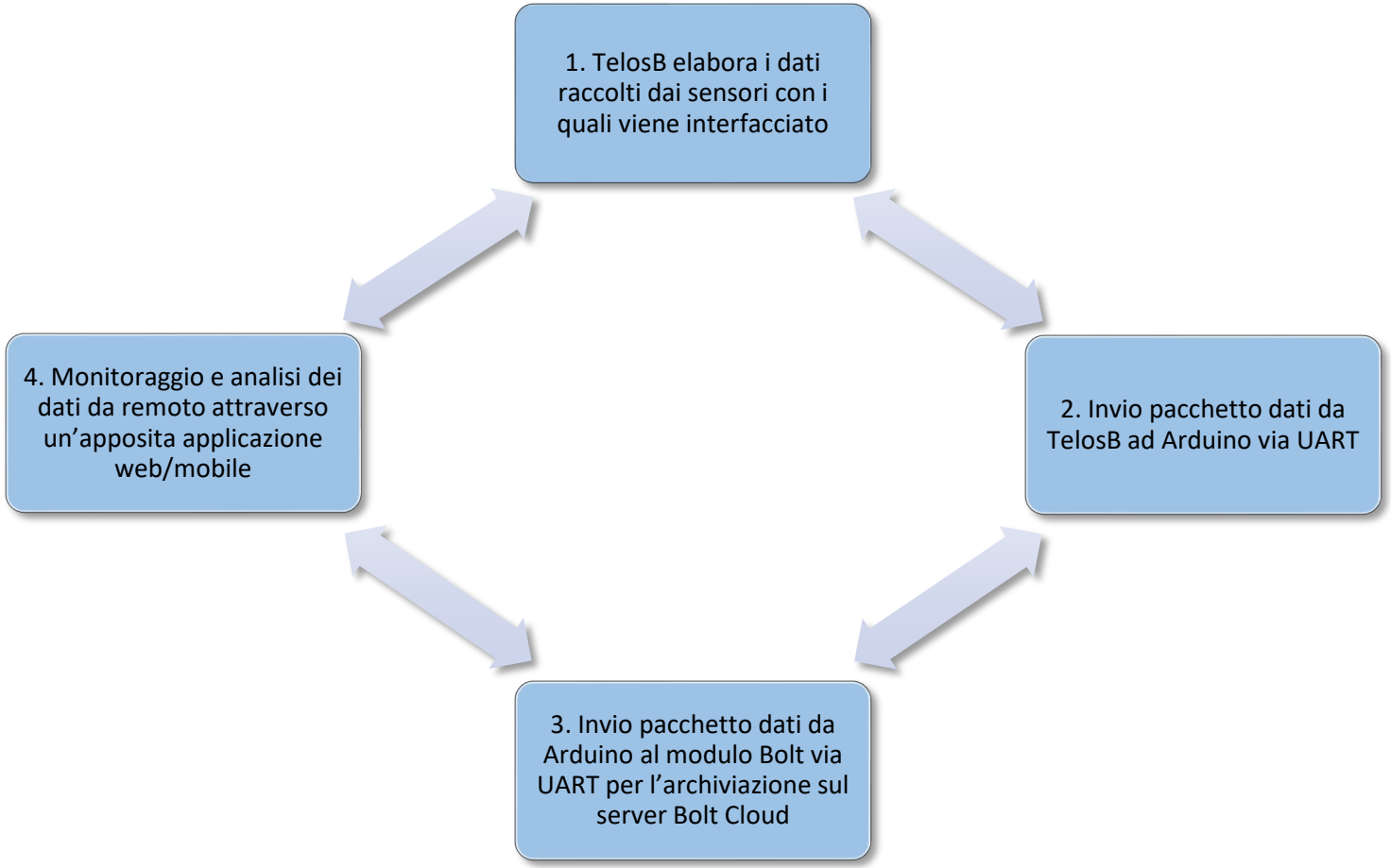
Panoramica del sistema

- Architettura del sistema



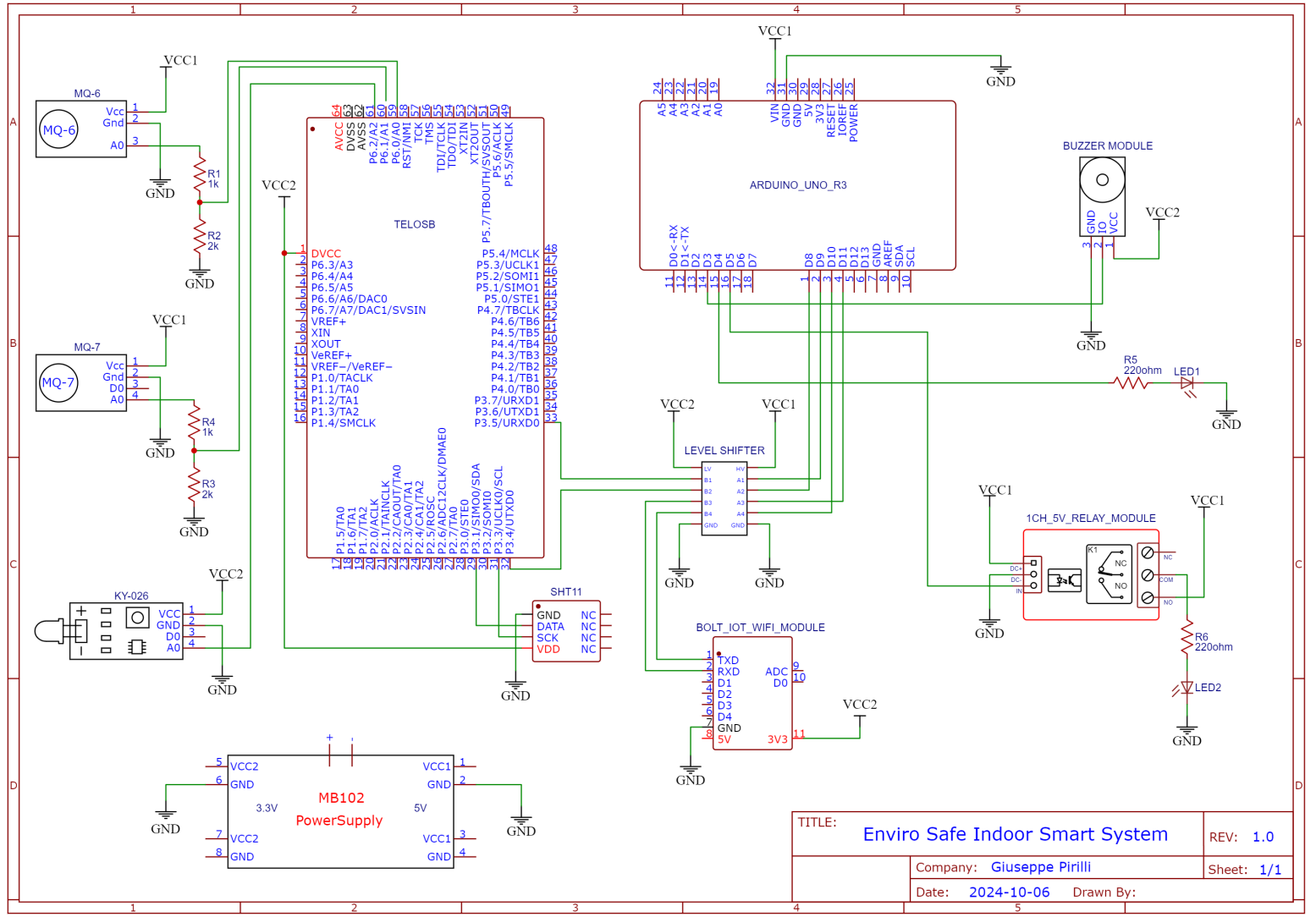
Panoramica del sistema

- Diagramma di flusso



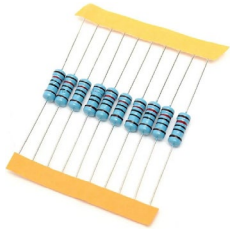
Panoramica del sistema

- Schema elettrico del sistema

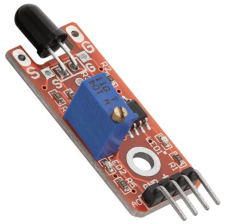


Progettazione hardware

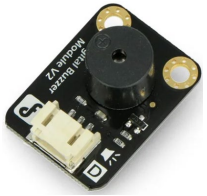
- Componenti hardware impiegati:
  - Power Supply
  - TelosB
  - Sensirion SHT11
  - MQ-6
  - MQ-7
  - KY-026
  - Resistori
  - Arduino Uno R3
  - Bolt IoT ESP8266-12S
  - BSS138 Level Shifter
  - KY-019 Relay 5V 1CH
  - Buzzer sonoro
  - Ventola
  - Leds





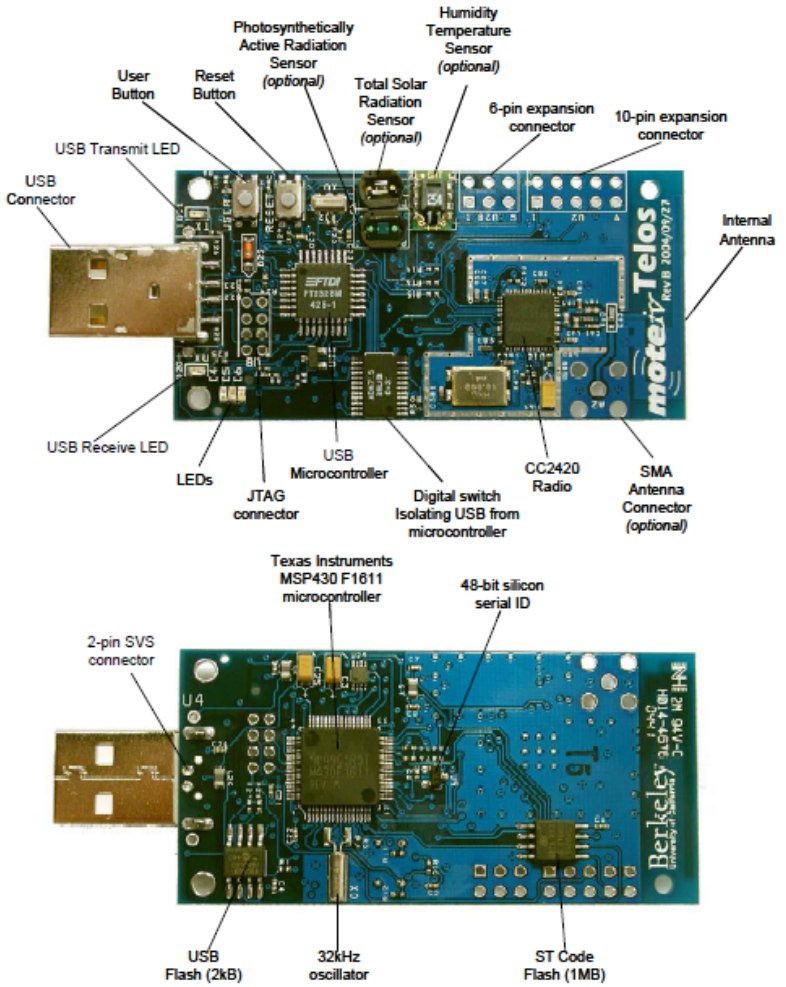



Progettazione hardware

- TelosB
  - Micro-controllore MSP430 Texas Instruments 8 MHz
  - 48 kB ROM Flash e 10 kB RAM
  - Convertitore ADC 12-bit e controller DMA
  - Ricetrasmettitore wireless 2.4 GHz Chipcon CC2420

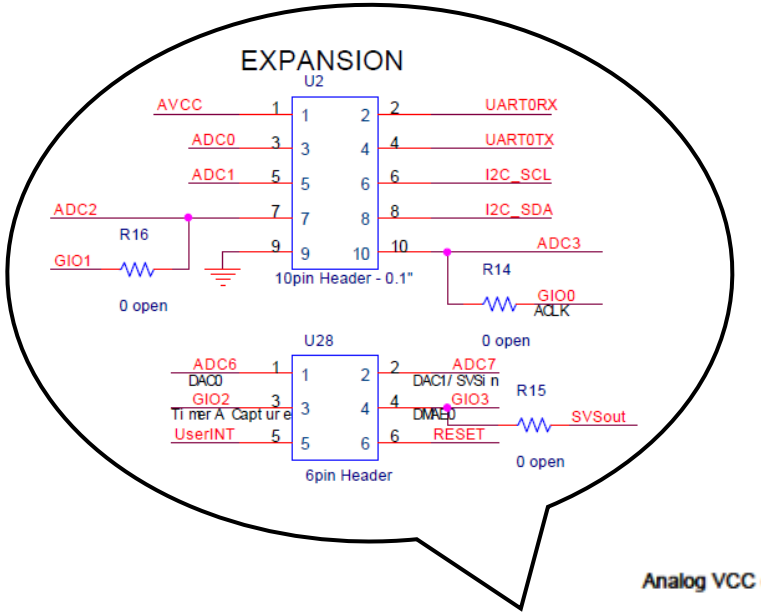
    IEEE 802.15.4 a 250kb/s
  - Limitate capacità di elaborazione
  - Sensori integrati di temperatura, umidità e luce
  - Alimentazione 3V tramite USB o batterie AA
  - Programmazione via USB nesC
  - Supporto sistema operativo TinyOS

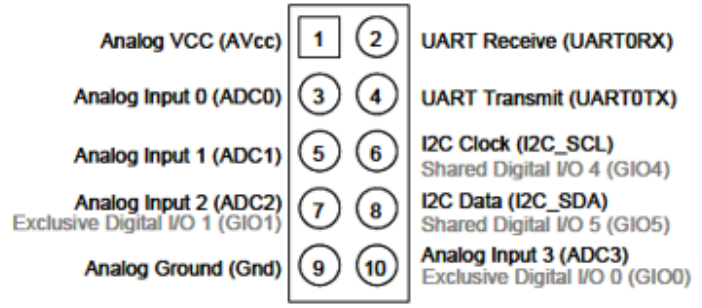


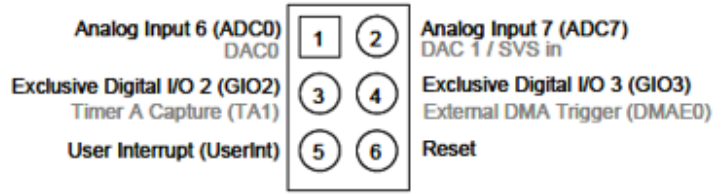
Progettazione hardware

Typical Operating Conditions

	MIN	NOM	MAX	UNIT
Supply voltage	2.1		3.6	V
Supply voltage during flash memory programming	2.7		3.6	V
Operating free air temperature	-40		85	°C
Current Consumption: MCU on, Radio RX		21.8	23	mA
Current Consumption: MCU on, Radio TX		19.5	21	mA
Current Consumption: MCU on, Radio off		1800	2400	µA
Current Consumption: MCU idle, Radio off		54.5	1200	µA
Current Consumption: MCU standby		5.1	21.0	µA







- Basso consumo di potenza
- Connettore di espansione a 16-pin
- 7 (U2) e 10 (U2) configurati come digital I/O popolando con una resistenza da 0 Ω, rispettivamente, R14 ed R15

Progettazione hardware

- Sensirion SHT11

Sensore	Parametri ambientali	Caratteristiche	Applicazioni	Specifiche elettriche
Sensirion SHT11 	Temperatura & Umidità 	— Basso consumo — Non richiede calibrazione — Stabilità e robustezza — Alta sensibilità — Alta precisione ($\pm 3\%\ RH, \pm 0.4$ °C)	Eccellente per progetti commerciali o domestici che richiedono tali letture	— Tensione operativa: 2.4 – 5.5 V — Corrente operativa: 28 μA — Consumo di potenza: 30 μW — Range operativo °C: -40 – 125 °C — Range operativo $\%RH$: 0 – 100 $\%RH$

  - Relazioni empiriche per convertire i dati grezzi di temperatura e umidità relativa in unità ingegneristiche $(°C, \%RH)$:

$$°C = -43.9 + 0.01 \cdot Raw\_value$$

$$\%RH = -2.0468 + 0.0367 \cdot Raw\_value - (1.5955 \cdot 10^{-6} \cdot Raw\_value^2)$$

Progettazione hardware

- MQ-6

Sensore	Parametri ambientali	Caratteristiche	Applicazioni	Specifiche elettriche
MQ-6 	Gas Propano (LPG) 	— Alta sensibilità a LPG — Bassa sensibilità al fumo — Stabilità e robustezza — Risposta veloce	Utilizzato nei dispositivi di rilevamento delle perdite di gas, quali LPG e metano, in ambito domestico e industriale	— Tensione operativa: $5\ V$ — Corrente operativa: $150\ mA$ — Range operativo: $200\ -10000\ ppm$ — Resistenza di carico R_L: Regolabile — Resistenza sensore R_S: $10-60\ k\Omega$

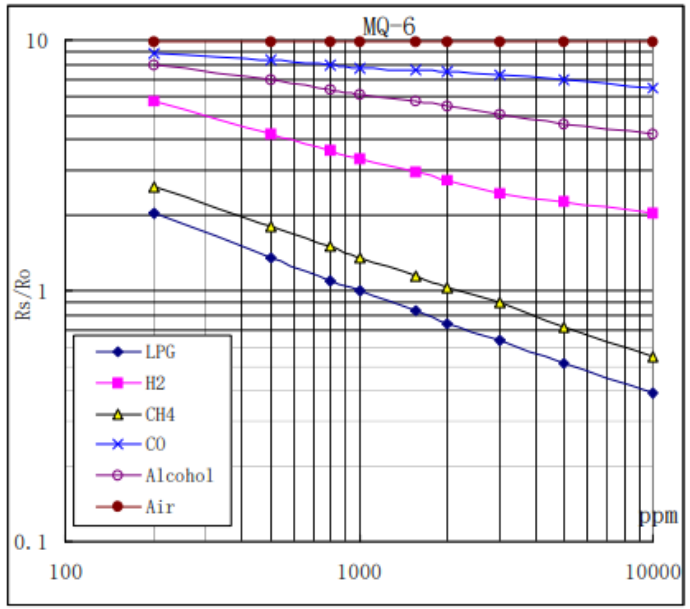
- Calibrazione sensore e calcolo concentrazioni LPG in ppm (part per million):

$$V_{out} = \frac{ADC\_data}{4095} \cdot V_{cc}$$

$$R_S = R_L \cdot \left(\frac{V_{cc}}{V_{out}} - 1\right)$$

$$R_0 = \frac{R_{S,clean\_air}^{avg}}{ratio_{clean\_air}}$$

$$ppm = ppm_1 + \frac{(ppm_2 - ppm_1) \cdot (ratio - ratio_1)}{ratio_2 - ratio_1}$$



Progettazione hardware

- MQ-7

Sensore	Parametri ambientali	Caratteristiche	Applicazioni	Specifiche elettriche
MQ-7 	Monossido di Carbonio (CO) 	— Alta sensibilità a CO — Stabilità e robustezza — Risposta veloce	Utilizzato nei dispositivi di rilevamento di monossido di carbonio in ambito domestico, industriale e automobilistico	— Tensione operativa: 5 V — Corrente operativa: 150 mA — Range operativo: 50 $-$ 5000 ppm — Resistenza di carico R_L: Regolabile — Resistenza sensore R_S: 2 $-$ 20 $k\Omega$

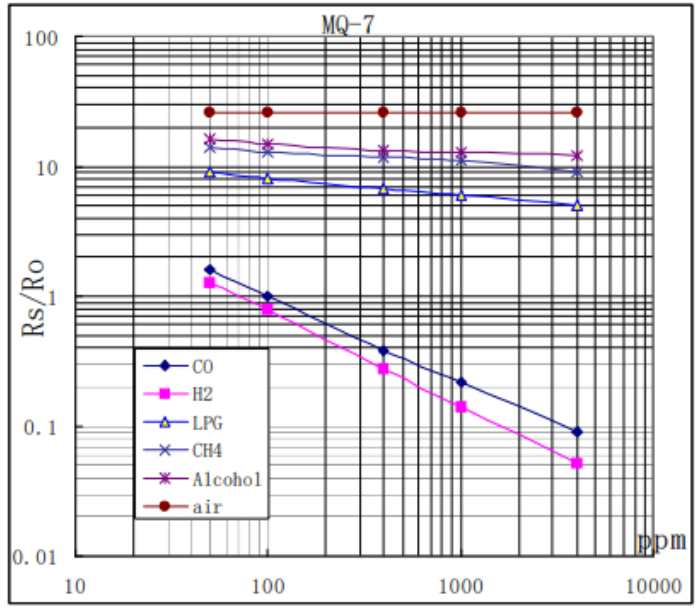
- Calibrazione sensore e calcolo concentrazioni CO in ppm (part per million):

$$V_{out} = \frac{ADC\_data}{4095} \cdot V_{cc}$$

$$R_S = R_L \cdot \left(\frac{V_{cc}}{V_{out}} - 1\right)$$

$$R_0 = \frac{R_{S,clean\_air}^{avg}}{ratio_{clean\_air}}$$

$$ppm = ppm_1 + \frac{(ppm_2 - ppm_1) \cdot (ratio - ratio_1)}{ratio_2 - ratio_1}$$



Progettazione hardware

CARBON MONOXIDE LEVELS CHART

Exposure Limits / Guidelines

Component	UK WEL	NIOSH REL	OSHA PEL
LPG (68476-85-7)	TWA: 1000 ppm (1750 mg/m3)ppm STEL: 1250 ppm(2180 mg/m3)	TWA: 1000 ppm(1800 mg/m3)	TWA: 1000 ppm(1800 mg/m3)

Toxicological Effects of Components

Toxicological Information		
Component	**Category**	**Data**
LPG (68476-85-7)	Exposure Routes	Inhalation, ingestion, skin, eye
	Symptoms	Irritation of eyes, nose, or throat; dyspnea (breathing difficulty); wheezing; chest pain; pulmonary edema; pink frothy sputum; skin burns; vesiculation; frostbite (liquid). Frostbite with redness, pain and blistering. Loss of sight to eyes.
	Target Organs	Eyes; skin; respiratory system, central nervous system.
	Short-Term Exposure	Vapors may cause narcosis with headache, dullness, difficulty breathing, lightheadedness, drowsiness, unconsciousness, and possibly death. Exposure may cause dizziness.
	Long-Term Exposure	Asphyxiation with rapid respiration, dyspnea, reduced mental alertness and muscle coordination, nausea, vomiting, prostration, unconsciousness, coma and death.

0 ppm	Permissible exposure level. No apparent toxic symptoms.
6 ppm	
9 ppm	
25 ppm	No poisoning symptoms for long time period. Allowable for several hours.
30 ppm	
35 ppm	Physical symptoms after 2-3 hours.
50 ppm	Physical symptoms in 1-2 hours. Life threatening 3 hours.
30-69 ppm	Physical symptoms in 45 minutes. Unconscious in 2 hours. Fatal in 2-3 hours.
87 ppm	Physical symptoms in 20 minutes. Fatal within 1 hour.
70-149 ppm	Physical symptoms in 5-10 minutes. Fatal within 25-30 minutes.
200 ppm	Physical symptoms in 1-2 minutes. Fatal within 10-15 minutes.
150-399 ppm	Fatal within 1-3 minutes.
400+ ppm	
800 ppm	
1,600 ppm	
3,200 ppm	
6.400 ppm	
12,800 ppm	

Physical Symptoms
physical symptoms may include headache, fatique, dizziness and/or nausia.

Progettazione hardware

- KY-026

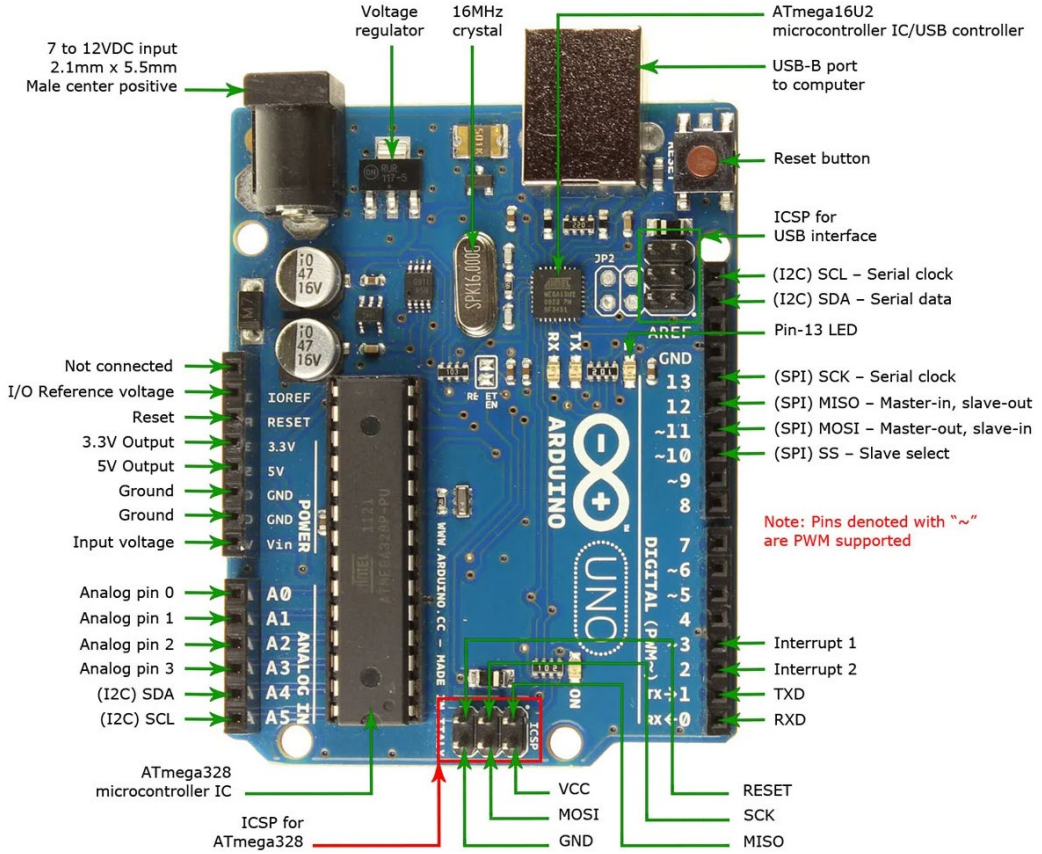
Sensore	Parametri ambientali	Caratteristiche	Applicazioni	Specifiche elettriche
KY-026 	Rilevatore di fiamma 	— Il fotodiodo collegato è sensibile allo spettro luminoso prodotto dalle fiamme — Risposta veloce — Alta sensibilità	Tipicamente impiegato nei dispositivi di rilevamento di incendi	— Tensione operativa: $3.3 - 5.5\ V$ — Range operativo: $760 - 1100\ nm$ — Angolo di rilevamento: 60°

  - Lettura ADC convertita in valore di tensione analogico:

$$V_{out} = \frac{ADC\_data}{4095} \cdot V_{cc}$$

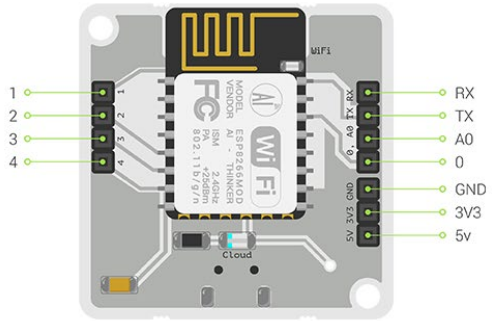
Progettazione hardware

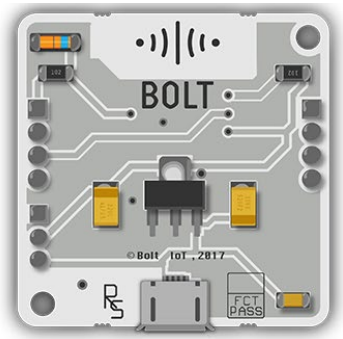
- Arduino Uno R3
  - Micro-controllore ATMega328P 16 MHz
  - 32 kB ROM Flash e 2 kB RAM
  - Convertitore ADC 10-bit
  - 6-pin analogici, 14-pin digitali
  - Protocolli seriali UART, SPI, I2C
  - Limitate capacità di elaborazione
  - Alimentazione 5V tramite USB o alimentatore
  - Programmazione via USB Arduino IDE

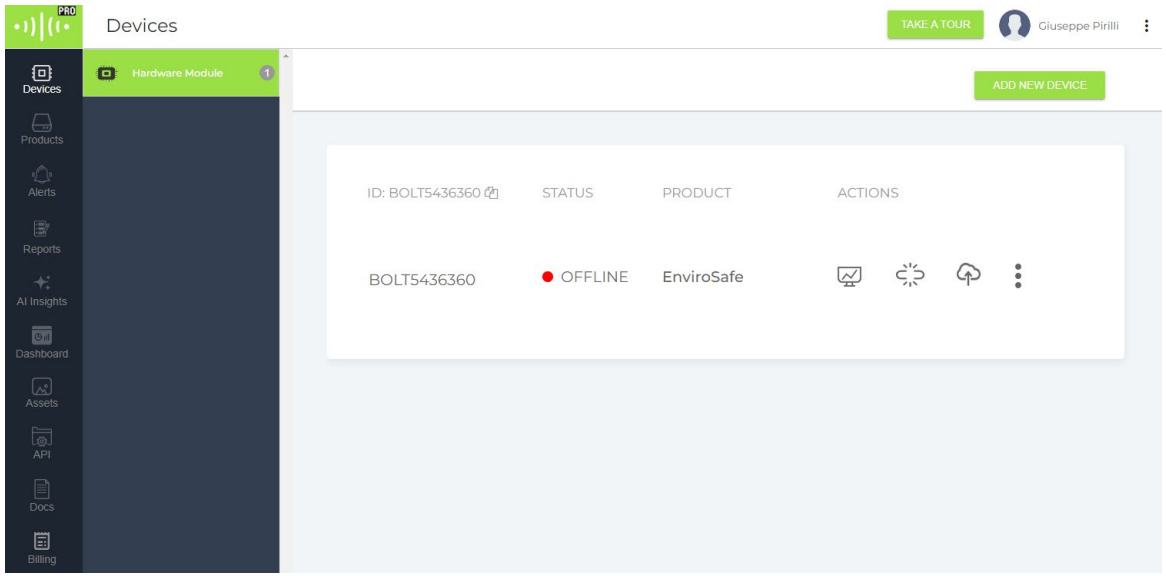


Progettazione hardware

- Bolt IoT ESP8266-12S
  - Micro-controllore Tensilica Xtensa LX106 80 MHz
  - 4 MB ROM Flash e 160 kB RAM
  - Convertitore ADC 10-bit
  - 1-pin analogico, 5-pin digitali
  - Protocollo seriale UART (limite baundrate 9600)
  - Wi-Fi 802.11 b/g/n
  - Bolt Cloud https://cloud.boltiot.com/
    1. Configurazione Remota
    2. Code Editor
    3. Smartphone App
    4. Notifiche (SMS & E-mail)
    5. Monitoraggio & Analisi
    6. Controllo remoto
  - Alimentazione 5V tramite micro-USB/pins 3,3V e GND
  - Tensione operativa 3.3V
  - Programmazione in Cloud JavaScript/HTML



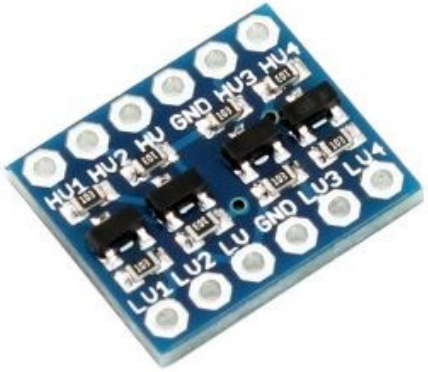


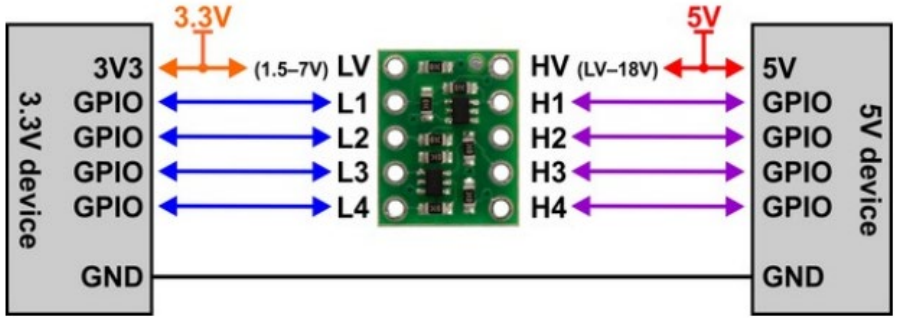


Progettazione hardware

- BSS138 Level shifter

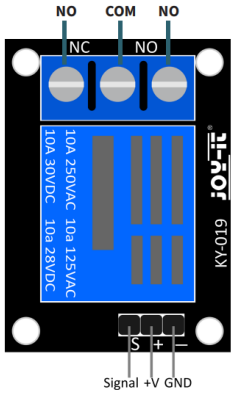
  - Utilizzato per stabilizzare i livelli di tensione dei segnali relativi alla comunicazione UART tra TelosB-Arduino e Arduino-Bolt IoT
  - Lower-Voltage (LV) supply compresa tra 1.5V - 7V
  - High-Voltage (HV) supply compresa tra LV - 18V
  - Adatto per canali bidirezionali





Progettazione hardware

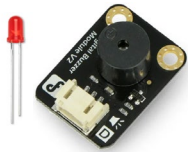
- KY-019 Relay 5V 1CH
  - Utilizzato per il controllo di attuatori azionati elettricamente
  - Segnale di controllo in input 5V
  - Tensione e corrente AC max: 10A, 250V
  - Tensione e corrente DC max: 10A, 30V
  - Segnale di controllo output AC o DC (adatto per controllo carico 220V AC)
  - Dotato di contatti normalmente aperto e normalmente chiuso



With the relay you can switch higher voltages by using a 5V output. This module can be soldered solidly or can be plugged on a breadboard.

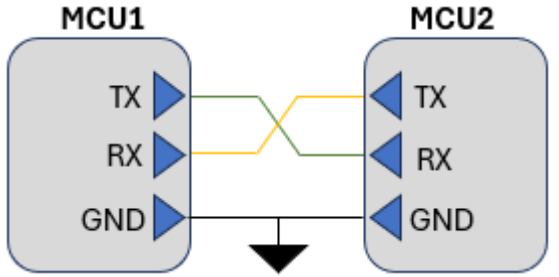
Progettazione hardware

- Attuatori

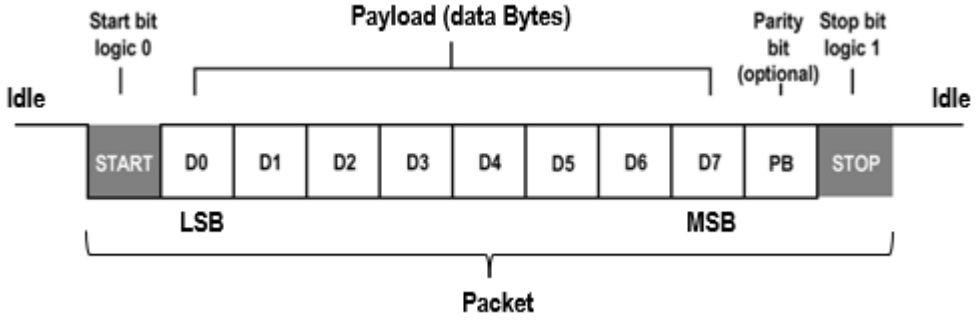
Attuatore	Caratteristiche	Applicazioni	Specifiche elettriche
Buzzer passivo con led 	N/A	Utilizzato in sistemi per la segnalazione di pericoli tramite l'emissione di un segnale acustico e luminoso	— Tensione operativa: $3.3V$ – $5V$
Sistema di ventilazione 	N/A	Utilizzato in sistemi per raffreddare o purificare l'aria in presenza di alti livelli di sostanze tossiche o infiammabili	— Tensione operativa: $12\ V$

Trasferimento dati

- Implementazione protocollo di comunicazione seriale UART per:
  - Avvio trasmissione dati TelosB-Arduino e Arduino-Bolt IoT
  - Visualizzazione messaggi di debug su monitor seriale da TelosB e Arduino (porta seriale USB)



- Caratteristiche principali :
  - Protocollo di comunicazione seriale asincrono (no segnale di clock)
  - Pacchetto e BaundRate (9600-115200) riconfigurabili
  - Master e Slave configurati con lo stesso BaundRate
  - Comunicazione di tipo Simplex, Half-duplex, Full-duplex
  - Trasmissione di ogni byte del payload bit a bit, dal LSB al MSB
  - Payload racchiuso tra bit di START e bit di STOP
  - Un bit di parità viene inviato dopo i dati (facoltativo)



Trasferimento dati

- Frame dati di lunghezza variabile:
  - Trasmissione al secondo di un pacchetto UART con payload di dimensioni pari a 6 Byte

Packet								
Start	Length	Payload						Stop
		LPG		CO		Flame		
SOP	LOP	Byte LSB	Byte MSB	Byte LSB	Byte MSB	Byte LSB	Byte MSB	EOP

  - Trasmissione ogni 30 secondi di un pacchetto UART con payload di dimensioni pari a 10 Byte

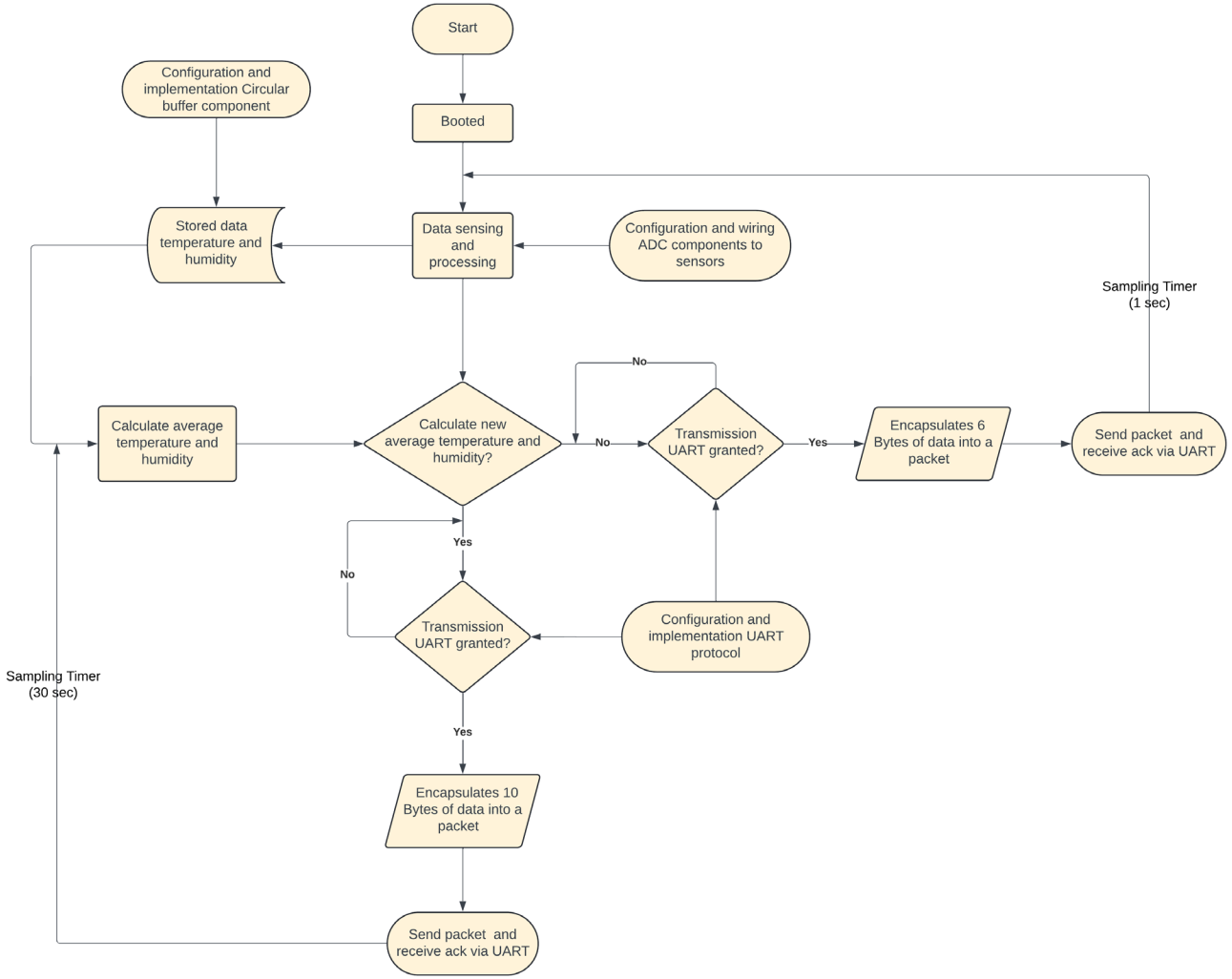
Packet										
Start	Length	Payload								Stop
		LPG		CO		Flame		Temp (avg)	RH (avg)	
SOP	LOP	Byte LSB	Byte MSB	Byte LSB	Byte MSB	Byte LSB	Byte MSB	Byte LSB	Byte MSB	Byte LSB	Byte MSB	EOP

Progettazione software

- Ambienti di sviluppo software impiegati per la programmazione:
  - TinyOS (nesC)
  - Visual Studio Code (Python)
  - Arduino IDE (C/C++)
  - Bolt Cloud (framework dedicato JavaScript)
- Struttura applicazione software:
  - TelosB firmware (sensing, elaborazione e trasmissione dati)
  - Arduino firmware (ricezione, elaborazione e trasmissione dati)
  - Applicazione Web/mobile (monitoraggio dei dati)
- Visualizzazione messaggi di debug su monitor seriale

Progettazione software

- TelosB firmware
  - Makefile
  - Module file *EnviroSafeC.nc*
  - Top level Configuration file *EnviroSafeAppC.nc*



Progettazione software

Makefile

- Definisce il componente top level dell'applicazione nesC
- Definisce le regole generali di compilazione
- Specifica le directory di librerie e componenti aggiuntivi
- Definisce delle costanti cruciali per l'applicazione nesC

```
# Definizione file sorgente
COMPONENT=EnviroSafeAppC

# Directory librerie esterne
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -I$(TOSDIR)/chips/msp430/usart

# Directory componenti personalizzati esterni
PFLAGS += -I/opt/tinyos-main/apps/EnviroSafe/Components/Modules/ADC12bit
PFLAGS += -I/opt/tinyos-main/apps/EnviroSafe/Components/Modules/uartConfig
PFLAGS += -I/opt/tinyos-main/apps/EnviroSafe/Components/Interfaces
PFLAGS += -I/opt/tinyos-main/apps/EnviroSafe/Components/Modules

# Definizione costanti
CFLAGS += -DSAMPLING_TIME0=1200
CFLAGS += -DSAMPLING_TIME1=30000+50
CFLAGS += -DACK_DIM=2
CFLAGS += -DSOP=60
CFLAGS += -DEOP=62
CFLAGS += -DLOP_ZERO=6
CFLAGS += -DLOP_ONE=10
CFLAGS += -DUART0_DIM=10
CFLAGS += -DUART1_DIM=14
CFLAGS += -DDIM_CIRC_BUFFER=50
CFLAGS += -DDIM_WINDOW=25
CFLAGS += -DBYTE_SIZE=8

include $(MAKERULES)
```

Progettazione software

Configuration file *EnviroSafeAppC*

- Definisce l'architettura dell'applicazione nesC, specificando quali componenti interagiscono e come le interfacce tra questi vengono collegate

- Componenti dell'applicazione:
  - *EnviroSafeC*, implementa la logica dell'applicazione
  - *SerialPrintfC*, gestisce la stampa di messaggi su seriale (debug)
  - *MainC*, gestisce la fase di avvio dell'applicazione
  - *TimerMilliC*, gestisce il campionamento periodico
  - *CircularBufferC*, per memorizzare i dati campionati periodicamente
  - *SensirionSht11C, per accedere alle letture del sensore integrato*
  - *Msp430adc#C,* per accedere alle letture dei sensori esterni
  - *MQ#InterpolationC*, gestisce la conversione in ppm dei dati di gas
  - *LedsC*, per feedback visivo
  - *Msp430Uart0C* e *uartConfigC*, gestiscono la comunicazione via UART

```
configuration EnviroSafeAppC {
}


implementation {

    components uartConfigC, EnviroSafeC, SerialPrintfC;

    components MainC;
    EnviroSafeC.Boot -> MainC;

    components new TimerMilliC() as Timer0;
    components new TimerMilliC() as Timer1;
    EnviroSafeC.SamplingTimer0 -> Timer0;
    EnviroSafeC.SamplingTimer1 -> Timer1;

    components LedsC;
    EnviroSafeC.Leds -> LedsC;

    components new CircularBufferC(DIM_CIRC_BUFFER) as CB_temperature;
    components new CircularBufferC(DIM_CIRC_BUFFER) as CB_humidity;
    EnviroSafeC.CB_temp -> CB_temperature;
    EnviroSafeC.CB_hum -> CB_humidity;

    components MQ6InterpolationC;
    components MQ7InterpolationC;
    EnviroSafeC.ppmPropane -> MQ6InterpolationC;
    EnviroSafeC.ppmCO -> MQ7InterpolationC;

    components new SensirionSht11C() as TemperatureDriver;
    components new SensirionSht11C() as HumidityDriver;
    components new Msp430adc0C() as ADC0;
    components new Msp430adc1C() as ADC1;
    components new Msp430adc2C() as ADC2;
    EnviroSafeC.RawPropane -> ADC0;
    EnviroSafeC.RawCO -> ADC1;
    EnviroSafeC.RawFlame -> ADC2;
    EnviroSafeC.RawTemp -> TemperatureDriver.Temperature;
    EnviroSafeC.RawHumidity -> HumidityDriver.Humidity;

    components new Msp430Uart0C();
    Msp430Uart0C.Msp430UartConfigure -> uartConfigC;
    EnviroSafeC.Resource -> Msp430Uart0C;
    EnviroSafeC.UartStream -> Msp430Uart0C;
}
```

Progettazione software

Module file *EnviroSafeC*

- Diverse interfacce utilizzate

```
/*
* @uses
* @desc Interfacce utilizzate dall'applicazione.
*/
uses {
    interface Boot;                 // Per eseguire azioni quando il sistema si avvia
    interface Leds;
    interface Timer<TMilli> as SamplingTimer0;
    interface Timer<TMilli> as SamplingTimer1;

    // Interfacce per acquisire i valori grezzi delle grandezze fisiche d'interesse
    interface Read<uint16_t> as RawTemp;
    interface Read<uint16_t> as RawHumidity;
    interface Read<uint16_t> as RawPropane;
    interface Read<uint16_t> as RawCO;
    interface Read<uint16_t> as RawFlame;

    interface CircularBuffer as CB_temp;        // Gestione dei buffer circolari per i dati di temperatura e umidita'
    interface CircularBuffer as CB_hum;
    interface MQ6Interpolation as ppmPropane;   // Per determinare la concetrazione di gas propano (in ppm)
    interface MQ7Interpolation as ppmCO;        // Per determinare la concetrazione di CO (in ppm)

    // Gestione della comunicazione UART
    interface Resource;
    interface UartStream;

}
```

Progettazione software

Boot

```
/*
* @event Boot.booted
* @desc Evento innescato all'avvio del sistema.
*/
event void Boot.booted() {

    /*
    * @call interface.command
    * @desc Inizializzazione dei buffer circolari e avvio di timers periodici.
    */
    call CB_temp.init();
    call CB_hum.init();
    call SamplingTimer0.startPeriodic(SAMPLING_TIME0);
    call SamplingTimer1.startPeriodic(SAMPLING_TIME1);
}
```

Timer

```
/*
* @event SamplingTimer0.fired
* @desc Evento innescato quando il timer scade.
******* Controllo led e lettura del valore grezzo di temperatura.
*/
event void SamplingTimer0.fired() {
    .
    .
    .
}

/*
* @event SamplingTimer1.fired
* @desc Evento innescato quando il timer scade per calcolare la media dei valori di temperatura e umidita' relativa.
******* Successivamente, i dati vengono formattati per renderli idonei alla trasmissione via UART.
*/
event void SamplingTimer1.fired() {
    .
    .
    .
}
```

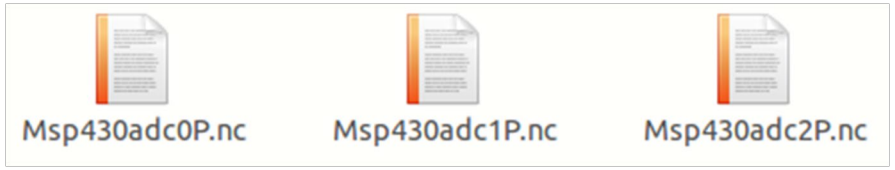
Progettazione software

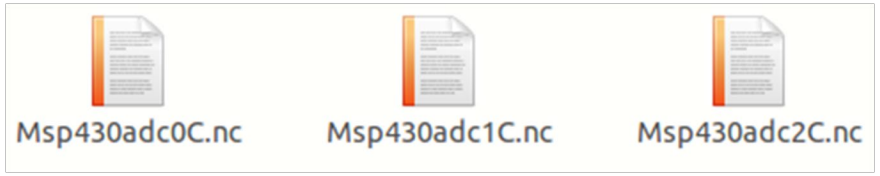
Read

```
event void SamplingTimer0.fired() {
        .
        .
    call RawTemp.read();
}

/*
* @event RawTemp.readDone
* @desc Evento innescato al termine della lettura del valore temperatura.
*/
event void RawTemp.readDone(error_t code, uint16_t value) {

    if(code == SUCCESS) {
        TempCelsius = -43.9+0.01*value;
        call CB_temp.putElem(TempCelsius);
        call RawHumidity.read();
    }
}

/*
* @event RawHumidity.readDone
* @desc Evento innescato al termine della lettura del valore umidita' relativa.
*/
event void RawHumidity.readDone(error_t code, uint16_t value) {

    if(code == SUCCESS) {
        RelativeHumidity = -2.0468+0.0367*value-(1.5955E-6*value*value);
        call CB_hum.putElem(RelativeHumidity);
        call RawPropane.read();
    }
}

/*
* @event RawPropane.readDone
* @desc Evento innescato al termine della lettura del valore LPG.
*/
event void RawPropane.readDone(error_t code, uint16_t value) {

    if ( code == SUCCESS ) {

        // Usa l'interfaccia MQ6Interpolation per calcolare il ppm
        call ppmPropane.Interpolate(value, &MQ6_Rs, &ratioPROPANE, &ppmPROPANE);

        call RawCO.read();
    }
}
```

```
/*
* @event RawCO.readDone
* @desc Evento innescato al termine della lettura del valore CO.
*/
event void RawCO.readDone(error_t code, uint16_t value) {

    if ( code == SUCCESS ) {

        // Usa l'interfaccia FermionCOInterpolation per calcolare il ppm
        call ppmCO.Interpolate(value, &MQ7_Rs, &ratioCO, &ppmMonoOxide);

        call RawFlame.read();
    }
}

/*
* @event RawFlame.readDone
* @desc Evento innescato al termine della lettura del valore Flame.
*/
event void RawFlame.readDone(error_t code, uint16_t value) {

    if ( code == SUCCESS ) {

        flameValue = (float)value/4095*3.3;
            .
            .
            .
    }
}
```

Progettazione software






```
#include "Msp430Adc12.h"

module Msp430adc0P {

  provides interface AdcConfigure<const msp430adc12_channel_config_t*>;

}


implementation {

  const msp430adc12_channel_config_t config = {

      inch: INPUT_CHANNEL_A0,
      sref: REFERENCE_AVcc_AVss,
      ref2_5v: REFVOLT_LEVEL_NONE,
      adc12ssel: SHT_SOURCE_ACLK,
      adc12div: SHT_CLOCK_DIV_1,
      sht: SAMPLE_HOLD_4_CYCLES,
      sampcon_ssel: SAMPCON_SOURCE_SMCLK,
      sampcon_id: SAMPCON_CLOCK_DIV_1
  };

  async command const msp430adc12_channel_config_t* AdcConfigure.getConfiguration() {
    return &config;
  }
}
```

Analog VCC (AVcc)	1	2	UART Receive (UART0RX)
Analog Input 0 (ADC0)	3	4	UART Transmit (UART0TX)
Analog Input 1 (ADC1)	5	6	I2C Clock (I2C_SCL) Shared Digital I/O 4 (GIO4)
Analog Input 2 (ADC2) Exclusive Digital I/O 1 (GIO1)	7	8	I2C Data (I2C_SDA) Shared Digital I/O 5 (GIO5)
Analog Ground (Gnd)	9	10	Analog Input 3 (ADC3) Exclusive Digital I/O 0 (GIO0)

```
generic configuration Msp430adc0C() {

  provides interface Read<uint16_t>;

}


implementation {

  components new AdcReadClientC();
  Read = AdcReadClientC;

  components Msp430adc0P;
  AdcReadClientC.AdcConfigure -> Msp430adc0P
}
```

Progettazione software

CircularBuffer

```
event void Boot.booted() {

    call CB_temp.init();
    call CB_hum.init();
        .
        .
}

event void RawTemp.readDone(error_t code, uint16_t value) {
            .
            .
        call CB_temp.putElem(TempCelsius);
            .
            .
}

event void RawHumidity.readDone(error_t code, uint16_t value) {
            .
            .
        call CB_hum.putElem(RelativeHumidity);
            .
            .
}

/*
* @event SamplingTimer1.fired
* @desc Evento innescato quando il timer scade per calcolare la media dei valori di temperatura e umidita' relativa
******* Successivamente, i dati vengono formattati per renderli idonei alla trasmissione via UART.
*/
event void SamplingTimer1.fired() {

    if (firstIteration) {
        shiftValue = 0;          // Nessun shift nella prima iterazione
        firstIteration = FALSE;      // Cambia lo stato per le iterazioni successive
    } else {
        shiftValue = DIM_WINDOW;     // Shift completo nelle iterazioni successive
    }

    call CB_temp.getWindow(window_temp, DIM_WINDOW, shiftValue);
    call CB_hum.getWindow(window_hum, DIM_WINDOW, shiftValue);
    Average(window_temp, &avg_temp);
    Average(window_hum, &avg_hum);
        .
        .
}
```

```
/*
* @void
* @desc Funzione utilizzata per il calcolo della media dei dati acquisiti al secondo.
*/
void Average(uint16_t* window, uint16_t* avg) {
    uint8_t i;
    sum = 0;
    for (i=0; i<DIM_WINDOW; i++) {
        sum += window[i];
    }
    *avg = sum/DIM_WINDOW;
}
```

Progettazione software

CircularBuffer.nc

```
interface CircularBuffer {

  command uint16_t size();

  command void init();

  command void putElem(uint16_t value);

  command uint16_t latestElem();

  command void getWindow(uint16_t* window, uint16_t wSize, uint16_t shiftSize);
}
```

CircularBufferC.nc

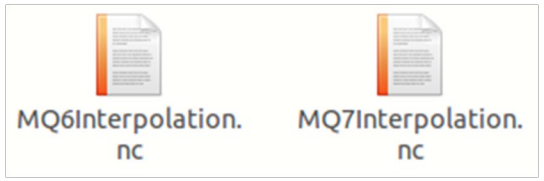
```
generic module CircularBufferC(uint16_t dim_buffer ) {

  provides interface CircularBuffer ;
}

implementation {

  uint16_t buffer[dim_buffer];
  uint16_t free_pos = 0;
  uint16_t i_wStart = 0;
  uint16_t i;

  command uint16_t CircularBuffer.size() {

    return dim_buffer;
  }

  command void CircularBuffer.init() {

    memset(buffer, 0, sizeof(buffer));
    free_pos = 0;
    i_wStart = 0;
  }

  //Ragioniamo in termini dell'ultima posizione libera, quindi l'elemento nella posizione
  //precedente rappresenta l'ultimo elemento inserito
  command uint16_t CircularBuffer.latestElem() {

    if (free_pos == 0)
       return buffer[dim_buffer-1];
    else
       return buffer[free_pos-1];
  }

  command void CircularBuffer.putElem(uint16_t value) {

    buffer[free_pos] = value;
    free_pos = (free_pos + 1) % dim_buffer;
  }

  command void CircularBuffer.getWindow(uint16_t* window, uint16_t wSize, uint16_t shiftSize) {

    i_wStart = (i_wStart + shiftSize) % dim_buffer;

    if (wSize > dim_buffer) {
          wSize = dim_buffer;       // Limita wSize alla dimensione del buffer
       }

    for (i=0; i<wSize; i++) {
       window[i] = buffer[(i_wStart + i) % dim_buffer];
    }
  }
}
```

Progettazione software

MQ#Interpolation

```
/*
* @event RawPropane.readDone
* @desc Evento innescato al termine della lettura del valore LPG.
*/
event void RawPropane.readDone(error_t code, uint16_t value) {

    if ( code == SUCCESS ) {

        call ppmPropane.Interpolate(value, &MQ6_Rs, &ratioPROPANE, &ppmPROPANE);
            .
            .
            .
    }
}

/*
* @event RawCO.readDone
* @desc Evento innescato al termine della lettura del valore CO.
*/
event void RawCO.readDone(error_t code, uint16_t value) {

    if ( code == SUCCESS ) {

        call ppmCO.Interpolate(value, &MQ7_Rs, &ratioCO, &ppmMonoOxide);
            .
            .
            .
    }
}
```

Progettazione software




```
/*
 * @interface MQ6Interpolation
 * @desc Definizione di un'interfaccia che dichiara un comando per eseguire l'interpolazione
 ******* lineare sulla curva di sensibilita' del sensore e ricavare la concentrazione di LPG,
 ******* espressa in ppm, in funzione del rapporto Rs/R0.
 */
interface MQ6Interpolation {

    command void Interpolate(uint16_t ADCdata, float* Rs, float* ratio, uint16_t* ppm);

}
```

```
/*
* @header
* @desc Contiene le costanti necessarie per l'algoritmo di interpolazione.
*/

#ifndef MQ6CONSTANTS_H
#define MQ6CONSTANTS_H

#define MQ6_NUM_POINTS 18


   const float MQ6_Vcc = 3.3;
   const float MQ6_RL = 10; // Resistenza di carico [kΩ]
   const float MQ6_R0 = 39; /* Resistenza sensore Rs in aria pulita determinata durante la calibrazione [kΩ]
                               R0 = Rs/(rapporto Rs/R0 a 200 ppm)*/


  // Punti noti sulla curva di sensibilità LPG per l'interpolazione
  const float MQ6sensitivityCurvePROPANE[MQ6_NUM_POINTS][2] = {

      {200, 2},   // 200 ppm -> Rs/R0 ~ 2
      {300, 1.8},
      {400, 1.6},
      {500, 1.45},
      {600, 1.4},
      {700, 1.3},
      {800, 1.2},
      {900, 1.1},
      {1000, 1},
      {2000, 0.73},
      {3000, 0.64},
      {4000, 0.58},
      {5000, 0.5},
      {6000, 0.48},
      {7000, 0.45},
      {8000, 0.42},
      {9000, 0.4},
      {10000, 0.39}
  };

#endif
```


```
#include "MQ6Constants.h"

module MQ6InterpolationC {

  provides interface MQ6Interpolation;

}

implementation {

  command void MQ6Interpolation.Interpolate(uint16_t ADCdata, float* Rs, float* ratio, uint16_t* ppm) {

    uint16_t i;
    float MQ6_Vout;
    float ppm_float;

    MQ6_Vout = (float)ADCdata/4095 * MQ6_Vcc;        // Conversione del valore ADC in tensione
    *Rs = MQ6_RL * ((MQ6_Vcc/MQ6_Vout)-1);           // Calcolo Rs (con RL ~ 10kΩ)
    *ratio = *Rs/MQ6_R0;                             // Calcolo rapporto Rs/R0

    // Implementazione dell'interpolazione lineare sulla curva di sensibilita' del sensore
    for (i = 0; i < (MQ6_NUM_POINTS-1); i++) {
      if (*ratio >= MQ6sensitivityCurvePROPANE[i+1][1] && *ratio <= MQ6sensitivityCurvePROPANE[i][1]) {
        ppm_float = MQ6sensitivityCurvePROPANE[i][0] +
                    (MQ6sensitivityCurvePROPANE[i+1][0] - MQ6sensitivityCurvePROPANE[i][0]) *
                    (*ratio - MQ6sensitivityCurvePROPANE[i][1]) /
                    (MQ6sensitivityCurvePROPANE[i+1][1] - MQ6sensitivityCurvePROPANE[i][1]);

        *ppm = (uint16_t) ppm_float;                 // Restituisce solo la parte intera
      }
    }

    // Se ratio fuori dal range, restituisci il valore limite
    if (*ratio > MQ6sensitivityCurvePROPANE[0][1]) {

    ppm_float = MQ6sensitivityCurvePROPANE[0][0];
        *ppm = (uint16_t) ppm_float;
    }

    if (*ratio < MQ6sensitivityCurvePROPANE[MQ6_NUM_POINTS - 1][1]) {
    ppm_float = MQ6sensitivityCurvePROPANE[MQ6_NUM_POINTS - 1][0];
        *ppm = (uint16_t) ppm_float;
    }
  }
}
```

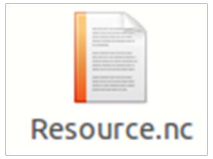
Progettazione software

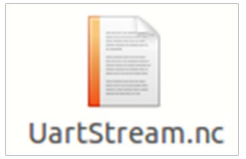
Resource
&
UartStream

```
/*
* @event RawFlame.readDone
* @desc Evento innescato al termine della lettura del valore Flame.
*/
event void RawFlame.readDone(error_t code, uint16_t value) {

    if ( code == SUCCESS ) {
                .
                .
                .
        // Richiesta di accesso al bus UART prima di trasmettere
        call Resource.request();
                .
                .
                .
    }
}

/*
 * @event Resource.granted
 * @desc Evento innescato quando il bus UART diventa disponibile per la trasmissione.
 ******* I dati vengono immagazzinati in un array di byte per renderli idonei alla trasmissione via UART.
 */
event void Resource.granted() {
                .
                .
                .
    // Transmissione: Invio array di byte via Uart da telosB ad Arduino
    if(call UartStream.send(UartData, currentUartDataSize) == SUCCESS) {
        call Leds.led0On();
    }

    // Ricezione: Invio ack via Uart da Arduino a telosB
    if(call UartStream.receive(receivedData, ACK_DIM) == SUCCESS) {
        call Leds.led2On();
    }
}
```

```
/*
 * @async_event UartStream.sendDone
 * @desc Evento asincrono innescato quando l'operazione di trasmissione é completata.
 */
async event void UartStream.sendDone(uint8_t* buf, uint16_t len, error_t error) {

    // Rilascia il bus UART dopo aver trasmesso il pacchetto dati
    call Resource.release();
}

/*
 * @async_event UartStream.receivedByte
 * @desc Evento asincrono innescato quando la trasmssione di un byte é completata.
 */
async event void UartStream.receivedByte(uint8_t byte) {

    // Rilascia il bus UART dopo aver trasmesso un byte
    call Resource.release();
}

/*
 * @async_event UartStream.receiveDone
 * @desc Evento asincrono innescato quando l'operazione di ricezione é completata.
 */
async event void UartStream.receiveDone(uint8_t* buf, uint16_t len, error_t error) {

    // Rilascia il bus UART dopo aver ricevuto l'ack
    call Resource.release();
}
```

Progettazione software






```
interface Resource {
  async command error_t request();
  async command error_t immediateRequest();
  event void granted();
  async command error_t release();
  async command bool isOwner();
}
```

```
interface UartStream {
  async command error_t send( uint8_t* buf, uint16_t len );
  async event void sendDone( uint8_t* buf, uint16_t len, error_t error );
  async command error_t enableReceiveInterrupt();
  async command error_t disableReceiveInterrupt();
  async event void receivedByte( uint8_t byte );
  async command error_t receive( uint8_t* buf, uint16_t len );
  async event void receiveDone( uint8_t* buf, uint16_t len, error_t error );
}
```

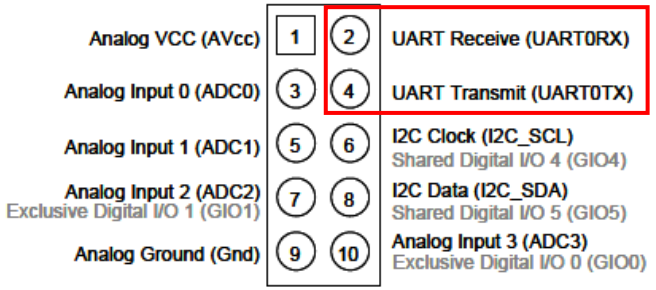
Progettazione software

uartConfigP.nc

```
#include "msp430usart.h"

module uartConfigP {

  provides interface Msp430UartConfigure;

}

implementation {

  msp430_uart_union_config_t msp430_uart_38400_config = {{
      ubr : UBR_1MHZ_38400,
      umctl : UMCTL_1MHZ_38400,
      ssel : 0X02,
      pena : 0,
      pev : 0,
      spb : 1,
      clen : 1,
      listen : 0,
      mm : 0,
      ckpl : 0,
      urxse : 0,
      urxeie : 1,
      urxwie : 0,
      utxe : 1,
      urxe : 1,

  }};

  async command msp430_uart_union_config_t* Msp430UartConfigure.getConfig() {
    return &msp430_uart_38400_config;
  }

}
```

uartConfigC.nc

```
configuration uartConfigC {

  provides interface Msp430UartConfigure;

}


implementation {

  components uartConfigP;
  Msp430UartConfigure = uartConfigP;

}
```



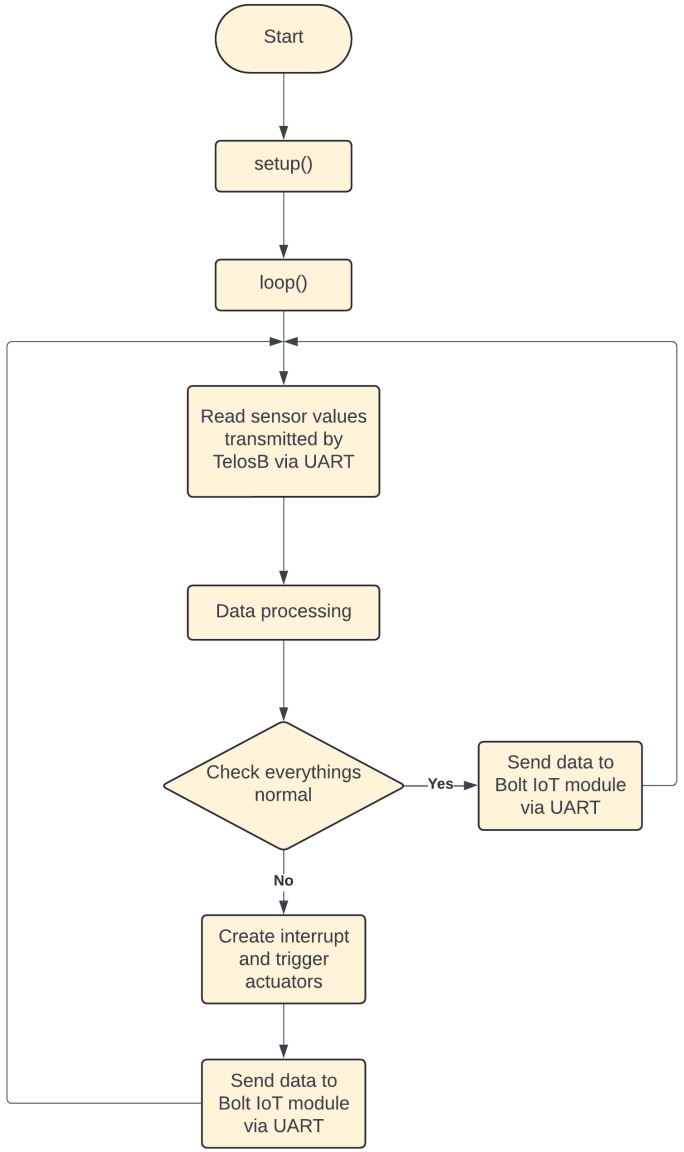
Progettazione software

```
event void Resource.granted() {

    if (sendToUart1Data) {

        // Memorizza i dati nell'array di byte per l'invio quando si calcolano le medie (ogni 30 secondi)
        Uart1Data[0] = SOP;                     // Inizio pacchetto dati
        Uart1Data[1] = LOP_ONE;                 // Lunghezza paccheto dati (in byte)
        Uart1Data[2] = ppmPROPANE & 0xFF;       //LSB
        Uart1Data[3] = ppmPROPANE >> 8;         //MSB
        Uart1Data[4] = ppmMonoOxide & 0xFF;
        Uart1Data[5] = ppmMonoOxide >> 8;
        Uart1Data[6] = flameValue & 0xFF;
        Uart1Data[7] = flameValue >> 8;
        Uart1Data[12] = EOP;                    // Fine pacchetto dati

        UartData = Uart1Data;
        currentUartDataSize = UART1_DIM;

        sendToUart1Data = FALSE;  // Resetta la flag

        // Ricezione acknowledgment trasmesso da Arduino
        ArduinoData = receivedData[1] << 8 | receivedData[0];

    } else {

        // Memorizza i dati nell'array di byte per l'invio ogni secondo
        Uart0Data[0] = SOP;
        Uart0Data[1] = LOP_ZERO;
        Uart0Data[2] = ppmPROPANE & 0xFF;
        Uart0Data[3] = ppmPROPANE >> 8;
        Uart0Data[4] = ppmMonoOxide & 0xFF;
        Uart0Data[5] = ppmMonoOxide >> 8;
        Uart0Data[6] = flameValue & 0xFF;
        Uart0Data[7] = flameValue >> 8;
        Uart0Data[8] = EOP;

        UartData = Uart0Data;
        currentUartDataSize = UART0_DIM;

        // Ricezione acknowledgment trasmesso da Arduino
        ArduinoData = receivedData[1] << 8 | receivedData[0];
    }
        .
        .
        .
}
```

```
event void SamplingTimer1.fired() {

    if (firstIteration) {
        shiftValue = 0;              // Nessun shift nella prima iterazione
        firstIteration = FALSE;      // Cambia lo stato per le iterazioni successive
    } else {
        shiftValue = DIM_WINDOW;     // Shift completo nelle iterazioni successive
    }
                .
                .
                .
    Uart1Data[8] = avg_temp & 0xFF;
    Uart1Data[9] = avg_temp >> 8;
    Uart1Data[10] = avg_hum & 0xFF;
    Uart1Data[11] = avg_hum >> 8;

    sendToUart1Data = TRUE;
}
```

Progettazione software

- Arduino firmware
  - Sketch *EnviroSafe.ino*



Progettazione software

```
// Librerie per la gestione delle comunicazioni seriali UART
#include <SoftwareSerial.h>
#include <AltSoftSerial.h>
#include <boltiot.h>

#define SOP '<'
#define EOP '>'
#define MAX_LENGTH 18   // Lunghezza massima del payload, può essere aumentata se necessario
#define Tens_Pin A3     // Utilizzato come ack
#define Buzzer_Pin 3    // Pin per il controllo del buzzer sonoro
#define Led_Pin 4       // Pin per il controllo del led associato al buzzer sonoro
#define Fun_Pin 5       // Pin per il controllo del sistema di ventilazione

AltSoftSerial telosbSerial;              // Sfrutta i pin 8-RX, 9-TX per la comunicazione seriale UART
//SoftwareSerial boltSerial(10,11);      // Permette di creare una porta seriale su qualsiasi coppia di pin digitali

unsigned long previousMillis = 0;        // Memorizza l'ultimo tempo in cui è stato cambiato lo stato del buzzer
const long interval = 300;               // Intervallo di tempo per alternare suono acceso/spento (300 ms)

bool buzzerState = false;                // Mantiene lo stato del buzzer (acceso/spento)

byte Transmit_Data[2];
uint16_t ack;

// Variabili globali per temperatura e umidità inizializzate a zero
uint16_t Temperature = 0;
uint16_t humidity = 0;

void setup() {

  // Interfaccia seriale hardware
  Serial.begin(38400);                           // Comunicazione con monitor seriale (pc)

  // Instanza di interfacce seriali software
  telosbSerial.begin(38400);                     // Comunicazione con telosB
  //boltSerial.begin(9600);
  boltiot.begin(10,11);                         // (10-RX, 11-TX) Comunicazione con il modulo Bolt a 9600 baud

  pinMode(Buzzer_Pin, OUTPUT);
  pinMode(Led_Pin, OUTPUT);
  pinMode(Fun_Pin, OUTPUT);
  digitalWrite(Buzzer_Pin, LOW);
  digitalWrite(Led_Pin, LOW);
  digitalWrite(Fun_Pin, LOW);
}
```

Progettazione software

```
void loop() {
  // Flag per inizio e fine pacchetto
  bool started = false;
  bool ended = false;

  uint8_t inData[MAX_LENGTH];
  byte index = 0;
  uint8_t packetLength = 0;     // Variabile per memorizzare la lunghezza del payload

  // Lettura pacchetto dati dal bus UART
  while (telosbSerial.available() > 0) {
    uint8_t inChar = telosbSerial.read();

    if (inChar == SOP) {
      index = 0;
      started = true;
      ended = false;
    } else if (inChar == EOP) {
        ended = true;
        break;
    } else {
        if (started) {
          if (index == 0) {
            packetLength = inChar;
            // Memorizzazione dati nel buffer
          } else if (index < packetLength + 1 && index < MAX_LENGTH) {
              inData[index - 1] = inChar;
          }
          index++;
        }
    }
  }
```

Progettazione software

```
if (started && ended && index == packetLength + 1) {

  // Processo i dati ricevuti
  Serial.println("");
  Serial.println("");
  Serial.println("Pacchetto completo ricevuto, elaborazione dei dati...");
  uint16_t Propane = word(byte(inData[1]), byte(inData[0]));  // I primi 2 byte sono relativi all'LPG
  uint16_t CO_ppm = word(byte(inData[3]), byte(inData[2]));
  uint16_t flame = word(byte(inData[5]), byte(inData[4]));
  Serial.print("LPG: ");
  Serial.print(Propane);
  Serial.print(" ppm, CO: ");
  Serial.print(CO_ppm);
  Serial.print(" ppm, Flame: ");
  Serial.print(flame);
  Serial.print("V");

  // Gestione Buzzer sonoro
  if (flame < 3) {
    handleBuzzer();
  } else {
    digitalWrite(Buzzer_Pin, LOW);
    digitalWrite(Led_Pin, LOW);
  }

  // Gestione sistema di ventilazione
  if (Propane >= 1000 || CO_ppm >= 500) {
    digitalWrite(Fun_Pin, HIGH);
  } else {
      digitalWrite(Fun_Pin, LOW);
  }

  if (packetLength > 6) {
    Temperature = word(byte(inData[7]), byte(inData[6]));
    humidity = word(byte(inData[9]), byte(inData[8]));
    Serial.print(", Temp(avg): ");
    Serial.print(Temperature);
    Serial.print(" C, RH(avg): ");
    Serial.print(humidity);
    Serial.print("%");
  }
```

```
// Funzione per alternare lo stato del buzzer sonoro e del led
void handleBuzzer() {
  if (millis() - previousMillis >= interval) {
    previousMillis = millis();                          // Aggiorna il tempo dell'ultimo cambiamento
    buzzerState = !buzzerState;                         // Inverti lo stato del buzzer
    digitalWrite(Buzzer_Pin, buzzerState ? HIGH : LOW);   // Modo compatto di scrivere una condizione booleana if-else
    digitalWrite(Led_Pin, buzzerState ? HIGH : LOW);
  }
}
```

Progettazione software

```
    // Invia i dati al modulo Bolt IoT per la visualizzazione in cloud
    boltiot.processPushDataCommand(Propane, CO_ppm, Temperature, humidity, flame);

    // Reset per il prossimo pacchetto
    started = false;
    ended = false;
    index = 0;
    packetLength = 0;

    // Trasmissione segnale di acknowledgement a TelosB
    ack = 1;
    successfullyUART();
  } else {
      ack = 0;
      failureUART();
  }

  delay(1200);

}
```

```
// Invia un messaggio di ACK positivo al trasmettitore
void successfullyUART() {
  Transmit_Data[0] = ack & 0xff;
  Transmit_Data[1] = ack >> 8;
  telosbSerial.write(Transmit_Data,2);
  Serial.println(" ");
  Serial.print("ACK: ");
  Serial.print(ack);
}

// Invia un messaggio di ACK negativo al trasmettitore
void failureUART() {
  Transmit_Data[0] = ack & 0xff;
  Transmit_Data[1] = ack >> 8;
  telosbSerial.write(Transmit_Data,2);
  Serial.println(" ");
  Serial.print("ACK: ");
  Serial.print(ack);
}
```

Progettazione software

- Debuging
  - VScode *Debug.py* (TelosB debug)
  - Sketch *EnviroSafe.ino* (Arduino debug)

```python
import serial.tools.list_ports
import time

#Recupero delle porte seriali USB disponibili
ports = serial.tools.list_ports.comports()
ser = serial.Serial()

portList = []

# Visualizzazione delle porte seriali USB disponibili
for onePort in ports:
    portList.append(str(onePort))
    print(str(onePort))

if not portList:
    print("Nessuna porta seriale disponibile.")
    exit()

# Selezione della porta desiderata su cui avviare la comunicazione
val = input("Seleziona porta seriale: COM")

# Check disponibilitá della porta seriale USB selezionata
portVar = None
for x in range(0,len(portList)):
    if portList[x].startswith("COM" + str(val)):
        portVar = "COM" + str(val)
        print(f"Porta selezionata: {portVar}")

if portVar is None:
    print("Errore: Porta non trovata!")
    exit()
```

```python
# Configurazione comunicazione seriale e ricezione dati
ser.port = portVar
ser.baudrate = 115200

try:
    ser.open()              # Apertura porta seriale USB
    ser.flushInput()        # Pulisce eventuali dati residui nel buffer
except serial.SerialException as errorPort:
    print(f"Errore nell'apertura della porta: {errorPort}")
    exit()

print("\nInizio lettura, in attesa di ricevere dati...   (Digita 'CTRL+C' per interrompere)\n")

try:
    # Lettura dei dati in un ciclo infinito
    while True:
        if ser.in_waiting:  # Verifica che ci siano dati disponibili nel buffer di ricezione
            packet = ser.readline()     # Legge una riga di dati da seriale USB
            ack = ser.readline()
            print(packet.decode('utf-8'))   # Stampa i dati ricevuti convertiti in stringa
            print(ack.decode('utf-8'))

except KeyboardInterrupt:
    print("\nInterruzione manuale. Chiudo la porta seriale.\n")
    ser.close()             # Chiusura porta seriale USB
```

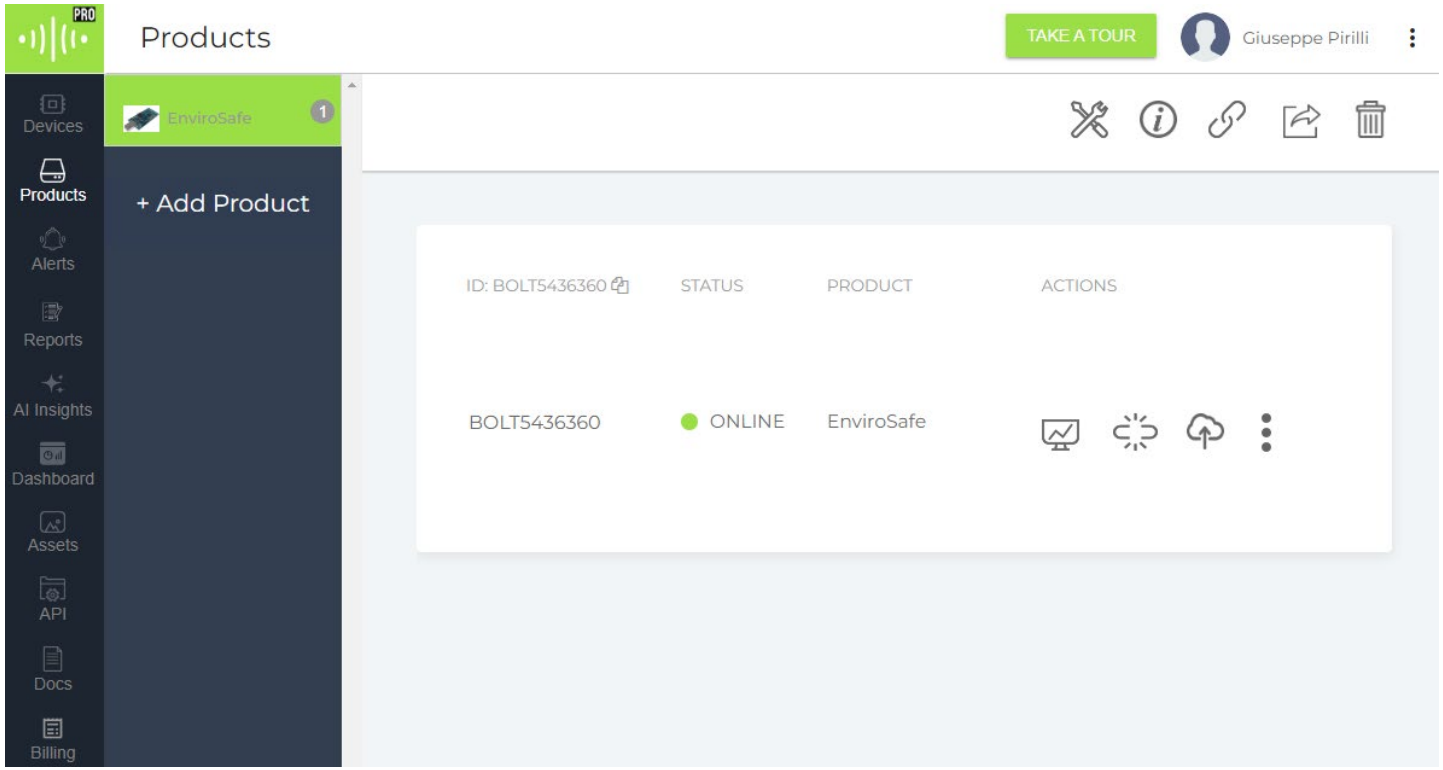
```
Serial.println("");
Serial.println("");
Serial.println("Pacchetto completo ricevuto, elaborazione dei dati...");
Serial.print("LPG: ");
Serial.print(Propane);
Serial.print(" ppm, CO: ");
Serial.print(CO_ppm);
Serial.print(" ppm, Flame: ");
Serial.print(flame);
Serial.print("V");
Serial.print(", Temp(avg): ");
Serial.print(Temperature);
Serial.print(" C, RH(avg): ");
Serial.print(humidity);
Serial.print("%");
Serial.println(" ");
Serial.print("ACK: ");
Serial.print(ack);
```

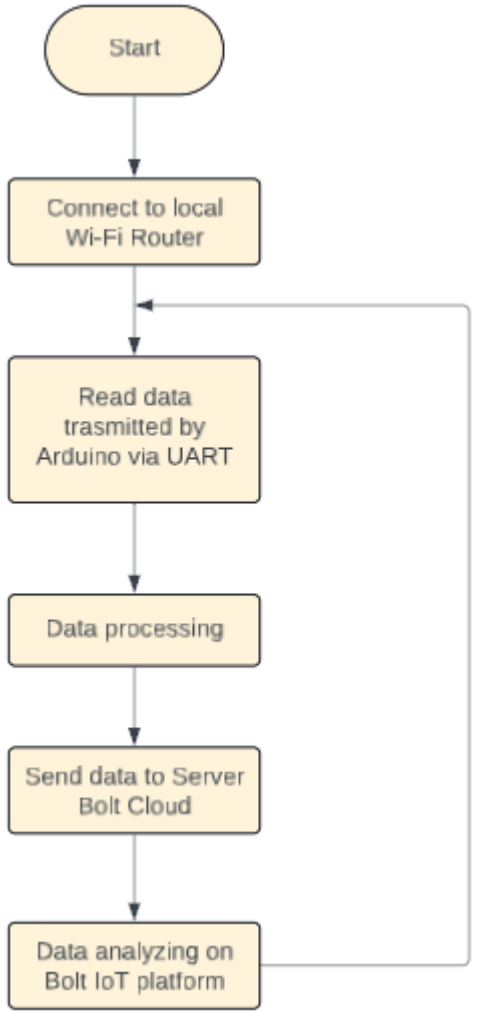
LATO TELOSB

```
printf("data_to_arduino = LPG: %d ppm, CO: %d ppm, Flame: %dV, Temp(avg): %d C, RH(avg): %d%%\n",
    ppmPROPANE, ppmMonoOxide,flameValue, avg_temp, avg_hum);
printf("ack = %d\n", ArduinoData);
printfflush();

printf("data_to_arduino = LPG: %d ppm, CO: %d ppm, Flame: %dV\n", ppmPROPANE, ppmMonoOxide, flameValue);
printf("ack = %d\n", ArduinoData);
printfflush();
```

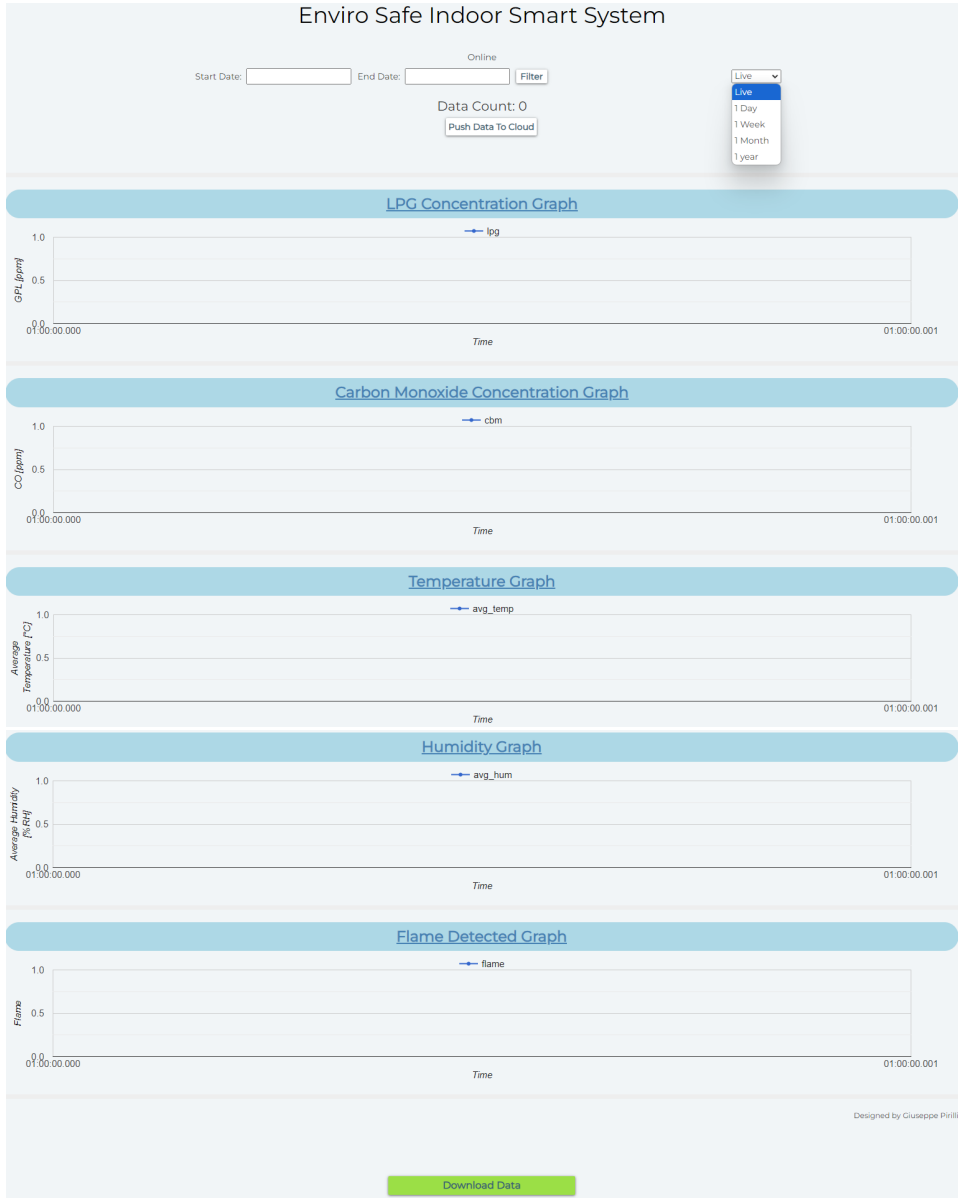
Applicazione Web/mobile

- Lato Bolt IoT: Applicazione Web/mobile



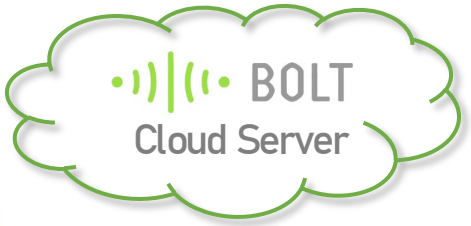


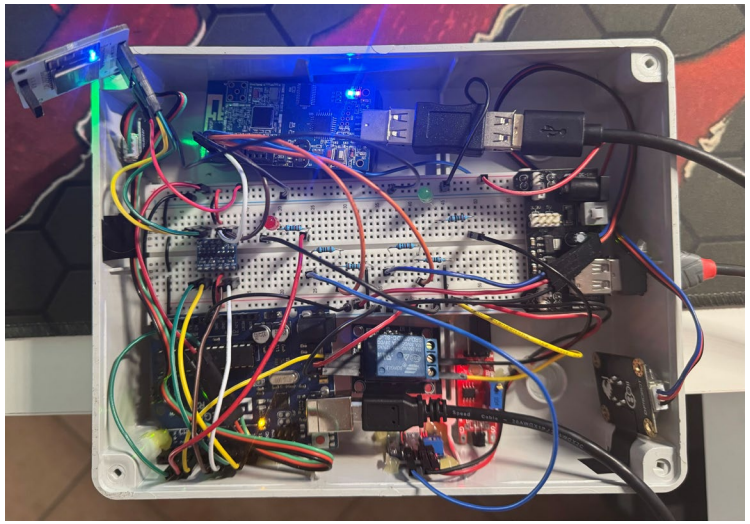
Applicazione Web/mobile



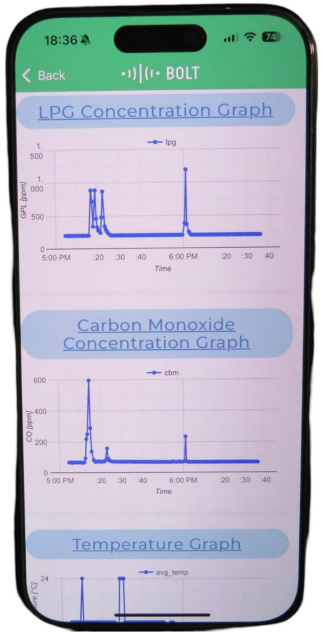
```
setChartLibrary('google-chart');
setChartTitle('Enviro Safe Indoor Smart System');


$("#muliple_graph_span").append("<hr style=border-width:7><h3 style=background-color:#ADD8E6;color:#4682B4;\
border-radius:30px;padding:10px><u><b>LPG Concentration Graph</b></u></h3>");
var lineGraph1 = new boltGraph();
lineGraph1.setChartType("lineGraph");
lineGraph1.setAxisName('Time','GPL [ppm]');
lineGraph1.plotChart('time_stamp','lpg');

$("#muliple_graph_span").append("<hr style=border-width:7><h3 style=background-color:#ADD8E6;color:#4682B4;\
border-radius:30px;padding:10px><u><b>Carbon Monoxide Concentration Graph</b></u></h3>");
var lineGraph2 = new boltGraph();
lineGraph2.setChartType("lineGraph");
lineGraph2.setAxisName('Time','CO [ppm]');
lineGraph2.plotChart('time_stamp','cbm');

$("#muliple_graph_span").append("<hr style=border-width:7><h3 style=background-color:#ADD8E6;color:#4682B4;\
border-radius:30px;padding:10px><u><b>Temperature Graph</b></u></h3>");
var lineGraph3 = new boltGraph();
lineGraph3.setChartType("lineGraph");
lineGraph3.setAxisName('Time','Average Temperature [°C]');
lineGraph3.plotChart('time_stamp','avg_temp');

$("#muliple_graph_span").append("<hr style=border-width:7><h3 style=background-color:#ADD8E6;color:#4682B4;\
border-radius:30px;padding:10px><u><b>Humidity Graph</b></u></h3>");
var lineGraph4 = new boltGraph();
lineGraph4.setChartType("lineGraph");
lineGraph4.setAxisName('Time','Average Humidity [%]');
lineGraph4.plotChart('time_stamp','avg_hum');

$("#muliple_graph_span").append("<hr style=border-width:7><h3 style=background-color:#ADD8E6;color:#4682B4;\
border-radius:30px;padding:10px><u><b>Flame Detected Graph</b></u></h3>");
var lineGraph5 = new boltGraph();
lineGraph5.setChartType("lineGraph");
lineGraph5.setAxisName('Time','Flame');
lineGraph5.plotChart('time_stamp','flame');

$("#muliple_graph_span").append("<hr style=border-width:7><h6 style=text-align:right> Designed by Giuseppe Pirilli</h6>");
```

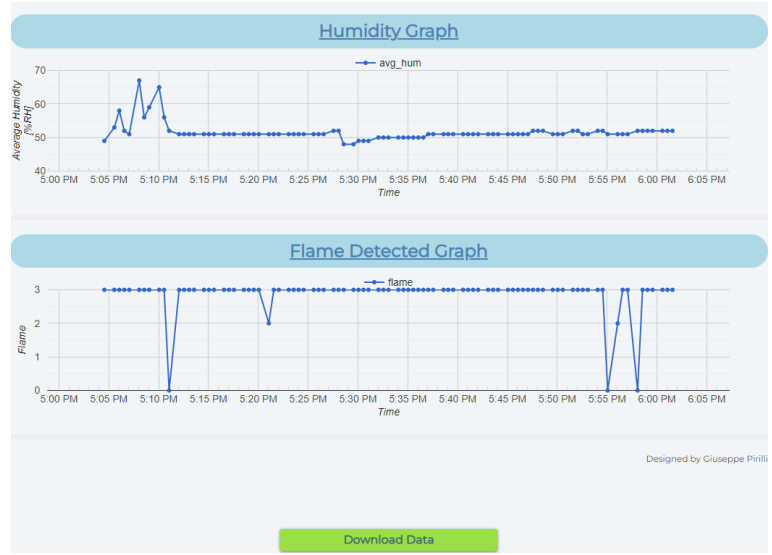
Risultati sperimentali





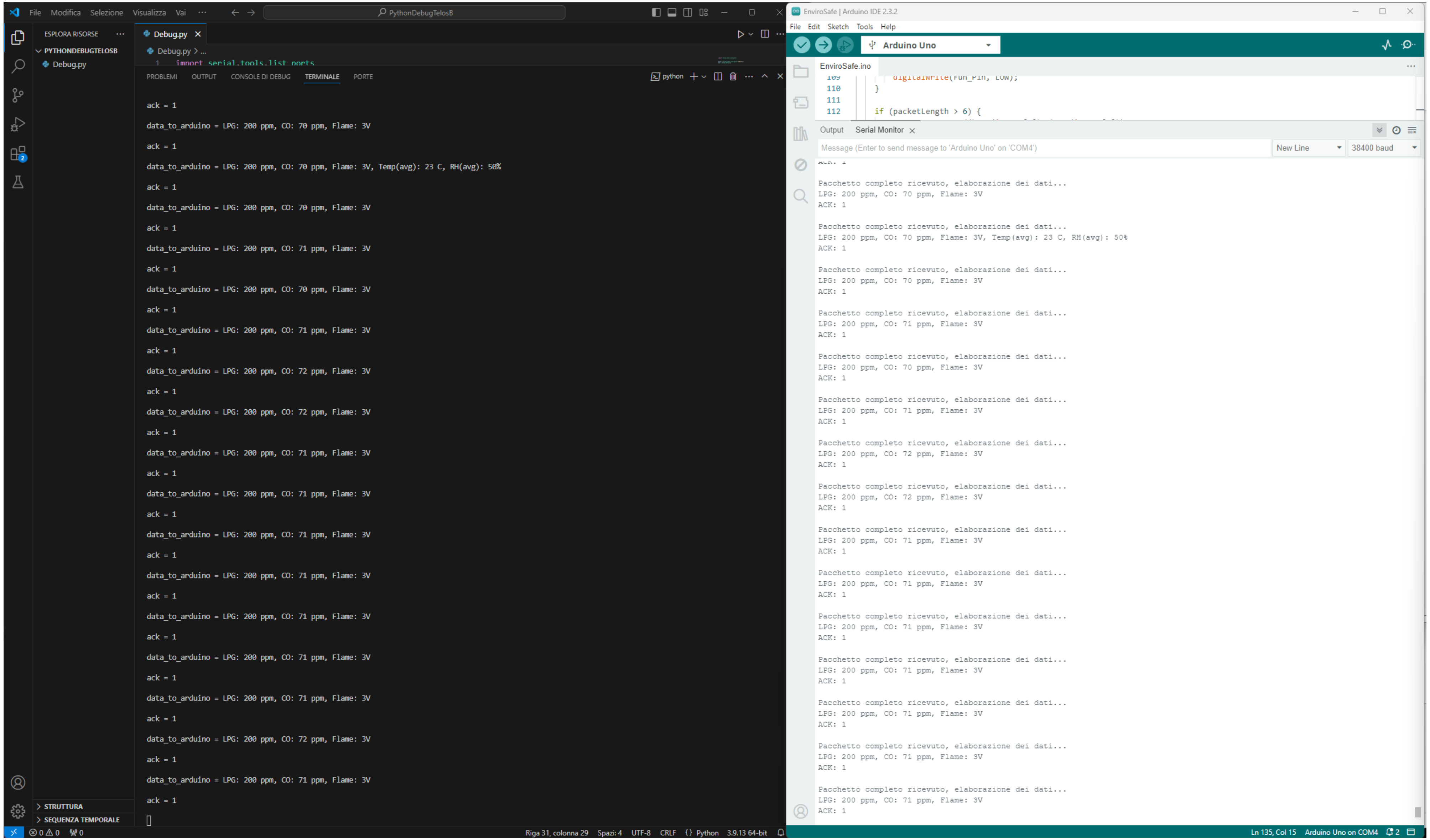
[Setup sperimentale]



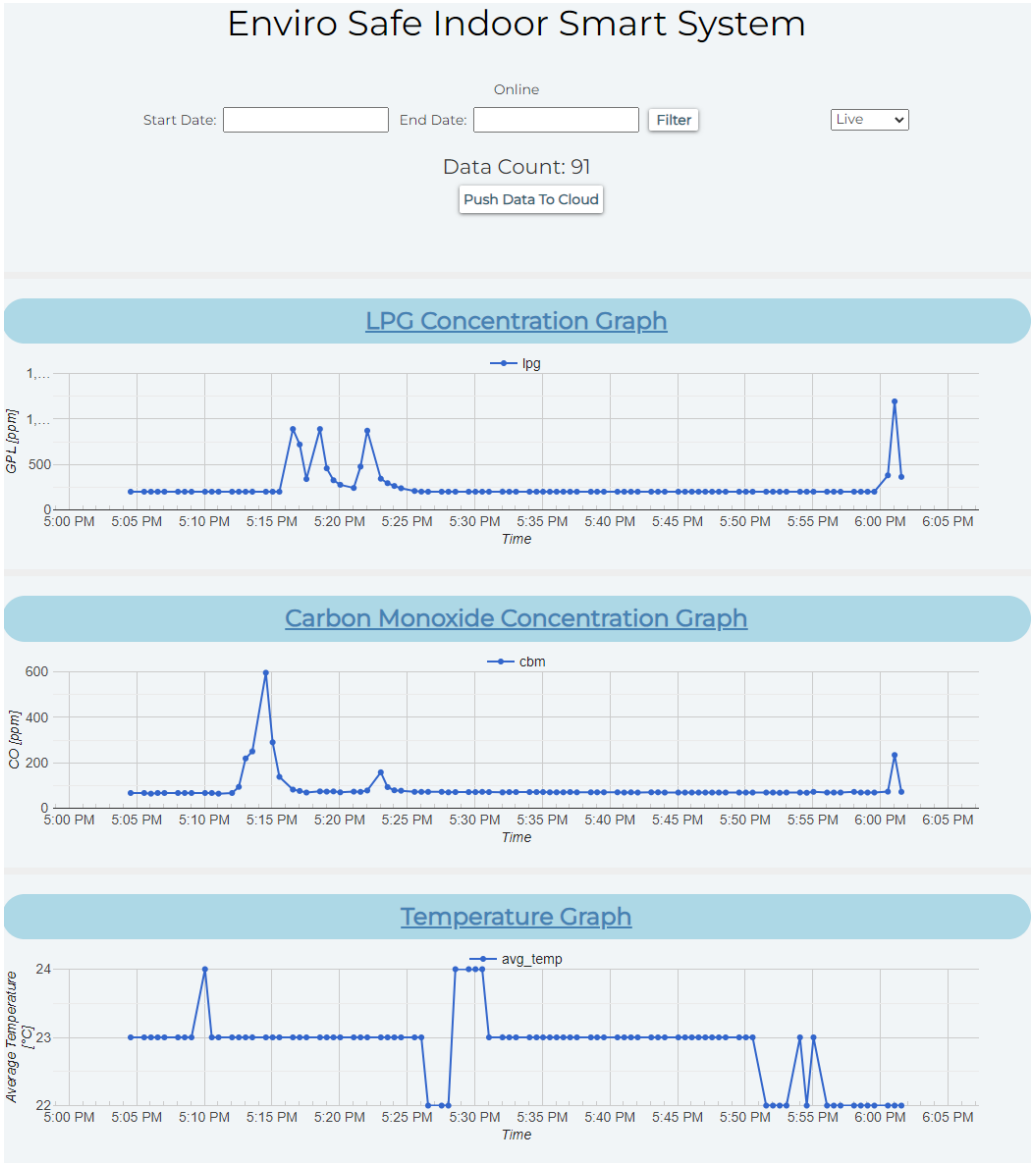


[GUI Webpage/App mobile]

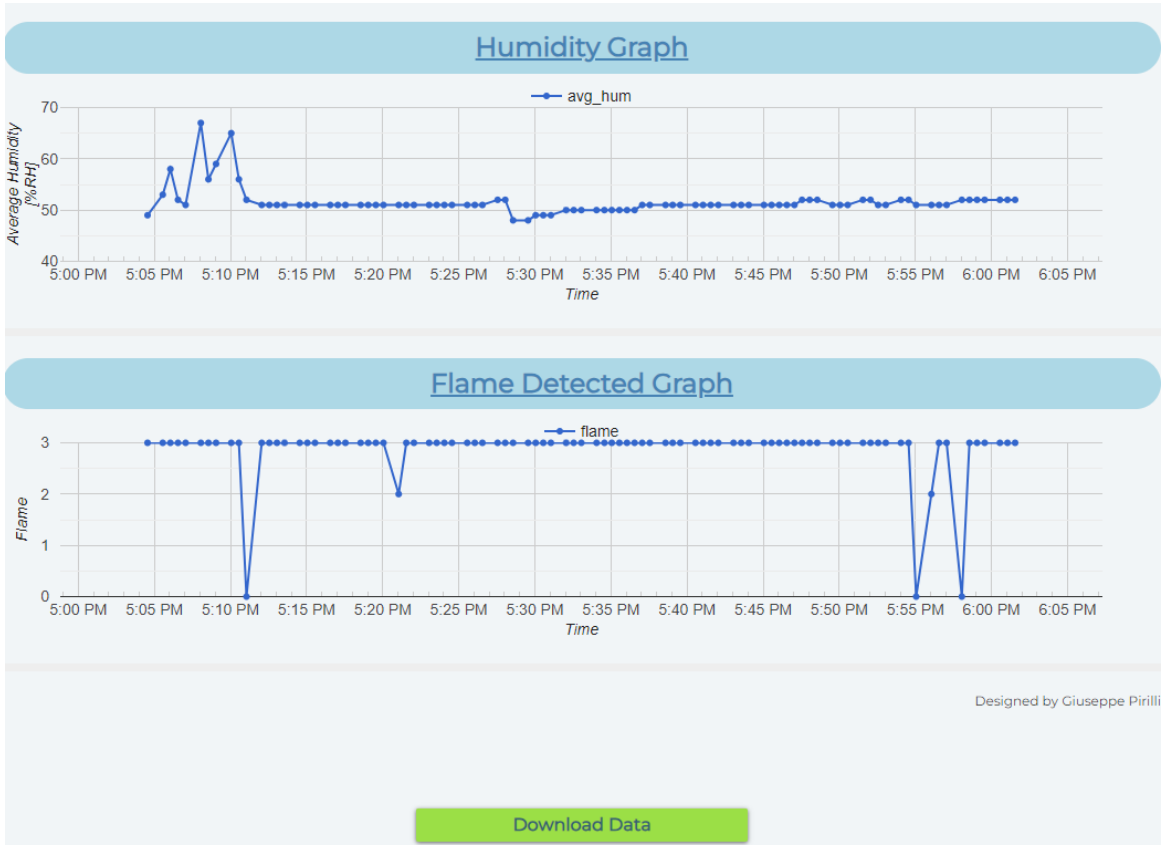
Risultati sperimentali



Risultati sperimentali



GUI Webpage



Conclusioni

- È importante disporre di sistemi domestici per rilevare tempestivamente la presenza di incendi e gas pericolosi, poiché questi possono rappresentare un rischio per la salute e la sicurezza delle persone.
- Il prototipo sviluppato rappresenta una soluzione efficace, ma migliorabile, per rilevare e mitigare i rischi associati a incendi e gas pericolosi in ambito domestico, garantendo la sicurezza delle persone.
- Un potenziale approccio per migliorare le prestazioni del sistema, in termini di efficienza, accuratezza e scalabilità, è stato identificato nell'integrazione di tecnologie IoT.
- Sistemi intelligenti basati su IoT offrono un miglioramento significativo rispetto al metodo tradizionale. Questi progressi contribuiscono a creare ambienti di vita più sicuri, sottolineando l'importanza della continua ricerca e sviluppo in questo campo.
- Nel complesso, il sistema sviluppato fornisce una soluzione economica, affidabile e comoda per rilevare e monitorare da remoto le condizioni ambientali indoor in ambito residenziale. La sua capacità di rilevare incendi e gas pericolosi, nonché la sua accuratezza e facilità d'uso, lo rendono uno strumento prezioso per garantire la sicurezza delle persone.

Sviluppi futuri

- Possibili upgrade da apportare al sistema:
  - Adattare il sistema per uso in ambiente industriale, oltre che residenziale
  - Ricerca sull'uso di un sistema di alimentazione a batteria
  - Integrare ulteriori sensori per il rilevamento di altri gas pericolosi, consentendo un monitoraggio più accurato è completo dell'ambiente
  - Inviare notifiche all'utente via SMS/e-mail qualora venga rilevata la presenza di un incendio o le concentrazioni di gas superano una soglia prestabilita
  - Controllo da remoto degli attuatori tramite piattaforma web/mobile IoT
  - Sviluppo di un'applicazione web/mobile proprietaria, senza usufruire di piattaforme IoT di terze parti
  - Dotare il sistema di modulo bluetooth per consentire il monitoraggio in loco in assenza di connessione internet tramite un'apposita applicazione pc/mobile
  - Integrare un algoritmo di machine learning addestrato sullo storico dei dati per migliorare l'accuratezza (riducendo i falsi positivi) e la reattività di previsione del sistema