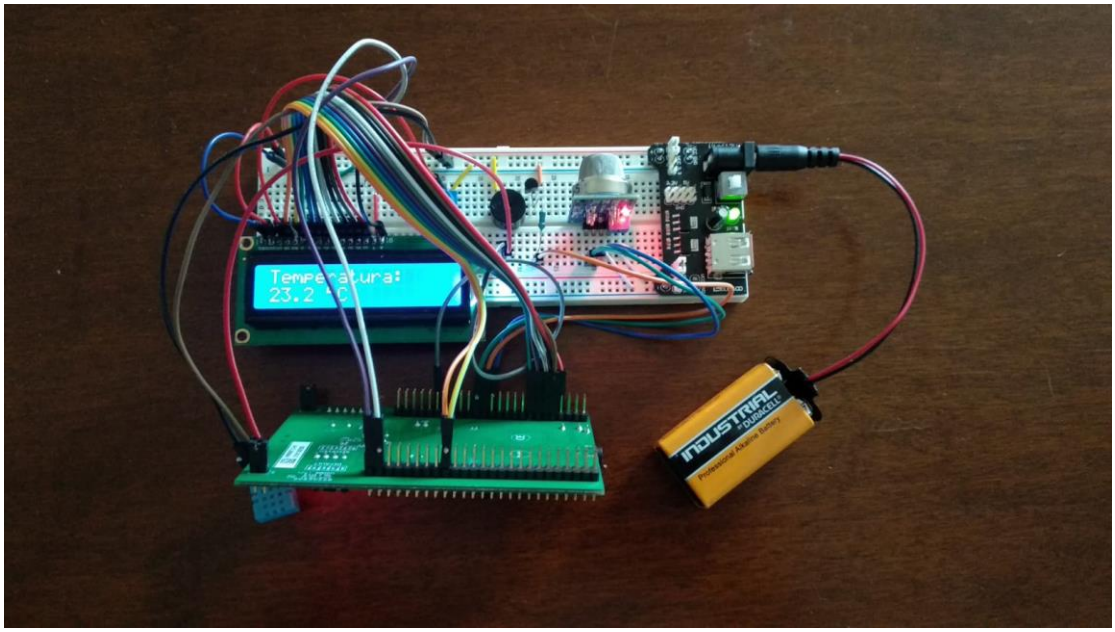
Università degli studi di Modena e Reggio Emilia
Dipartimento di Ingegneria “Enzo Ferrari”
Corso di Laurea Magistrale in Ingegneria Informatica



Progetto di un sensore di rilevamento fuga di gas naturale, su Arduino Uno R3

PROGETTO DI CORSO

Fabrizio Di Blasi | Informatica industriale ed IoT | a.a 2018/2019

Sommario:

Introduzione:

In questa relazione viene esposto il processo di progettazione e prototipazione di una centralina per il rilevamento di fughe di gas.
Tale sistema è stato concettualizzato per essere adottato all'interno di un distributore della rete stradale per il rifornimento di auto GPL.
Il suo scopo è quello di chiudere la valvola che dalla cisterna porta al bocchettone di rifornimento nel momento in cui il sensore di gas rilevi una fuga.
Il sistema ha lo scopo di allertare l'utente tramite un messaggio proveniente dal BOT Telegram (previa sua inizializzazione).
L'utente potrà "dialogare" col sensore tramite una serie di comandi molto basilari messi a disposizione e ricevere uno storico delle misurazioni.

Studio di fattibilità:

Requisiti e ambito di applicazione:

Il progetto presentato descrive un prodotto da considerarsi esclusivamente a scopo illustrativo, poiché non soddisfa ancora tutti gli stringenti vincoli di sicurezza industriale.
Il requisito di maggiore importanza è la rilevazione istantanea di un'eventuale fuga di gas e la durata maggiore possibile della batteria di alimentazione del sistema.
Essendo un'oggetto critico, è consigliato apporre un packaging ignifugo e resistente ad agenti esterni come polvere o sporcizia in modo da non compromettere il rilevamento del gas.

Vincoli:

Il prodotto finale per il corretto funzionamento dovrà essere disposto lontano da fonti di calore, in luoghi chiusi e non eccessivamente umidi.

In particolare, i vincoli del prodotto derivano direttamente dai vincoli dei singoli componenti utilizzati; rilevazioni corrette sono assicurate in ambienti compresi tra 0°C e 50°C, umidità comprese tra il 20% e il 90% e ambienti adeguatamente ossigenati (intorno al 21% per mantenere il normale regime di utilizzo).

Componenti Impiegati nel prototipo:

Per la realizzazione del prototipo sono stati usati i seguenti componenti:

Board Arduino Uno R3:

- Core
    - * ATmega328P @16Mhz
- Memoria
    - * Flash memory 32 KB
    - * SRAM 2 KB
    - * EEPROM 1 KB
- Periferiche
    - * Digital I/O Pins 14 (di cui 6 con PWM)
    - * PWM Digital I/O Pins6
    - * Analog Input Pins 6

Servomotore MG946R:

Caratteristiche:

- * Peso: 55g
- * Dimensioni: 40.7×19.7×42.9mm
- * Stall torque: 10.5kg/cm (4.8v); 13kg/cm (6v)
- * Operating speed: 0.20sec/60° (4.8v);0.17sec/60°(6.0v)
- * Alimentazione: 4.8-6.6v
- * Temperatura: 0- 55°

Sensore di Gas MQ5:

Caratteristiche:

- * Alimentazione a 5V DC
- * 150mA di assorbimento
- * Uscita TTL (D0) e analogica (A0)

La misurazione di questo sensore sono affidabili solamente se il sistema è ad una temperatura posta tra -20 °C e +70 °C ed al di sotto del 95% di umidità.
Inoltre un parametro da non sottovalutare è la concentrazione di ossigeno nell'ambiente, che in condizioni standard è del 21%. Un tasso maggiore o inferiore può causare degli errori di misura della concentrazione di gas.

Il sensore comunica col microcontrollore tramite un'uscita analogica e digitale (TTL).
L'uscita digitale assume valore logico

1 – se non c'è fuga di gas

0 – se viene rilevata la fuga di gas

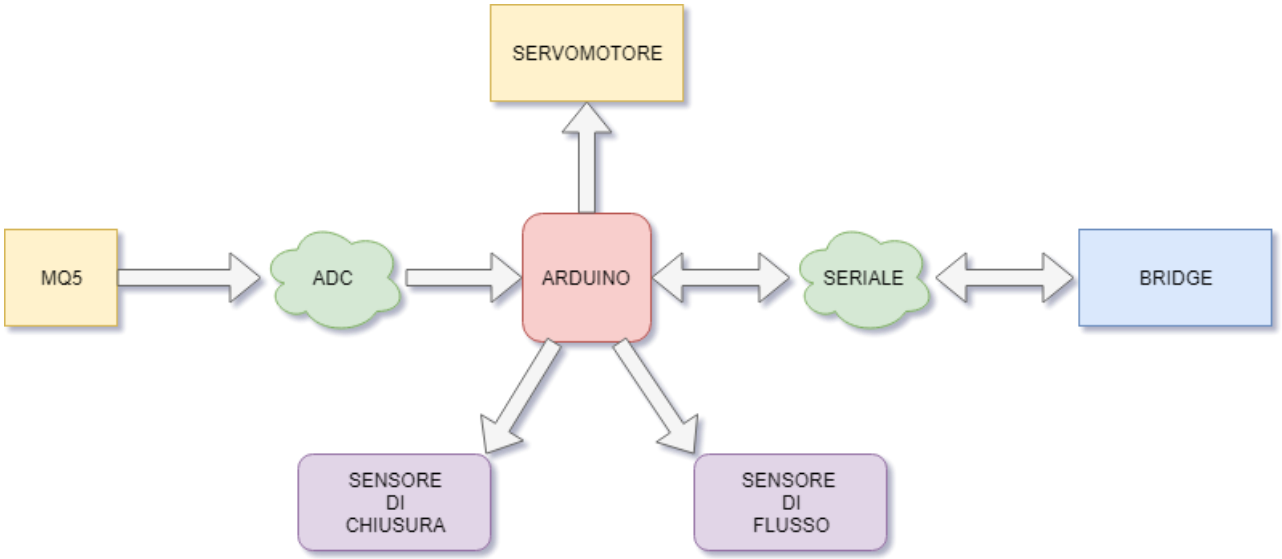
Tale segnale è utilizzato per la gestione dell'interrupt. Infatti appena avviene il rilevamento, viene mostrata sul display esclusivamente la misura del gas.
L'uscita analogica invece è quella che fornisce la misurazione in ppm della concentrazione di gas. Tale output viene fornito al convertitore ADC del microcontrollore.

Micro Switch:

Caratteristiche:
Questi switch sono stati adottati per il controllo dell'avvenuta chiusura della saracinesca e per simulare, manualmente, un flussometro posto all'interno del condotto.
Essi sono "normalmente aperti", cioè in condizione di riposo il circuito è interrotto solamente quando avviene la pressione del bottone il circuito si chiude

Implementazione:

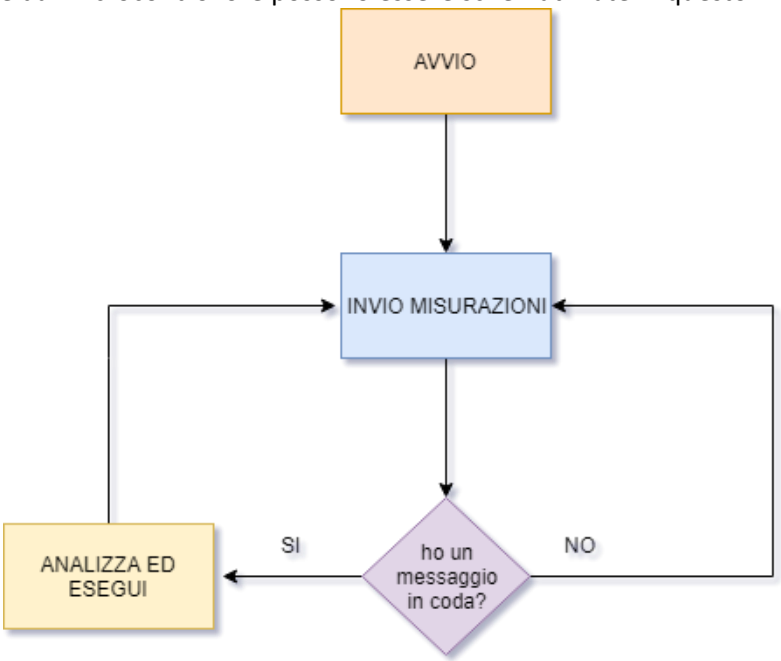
Schema a blocchi:



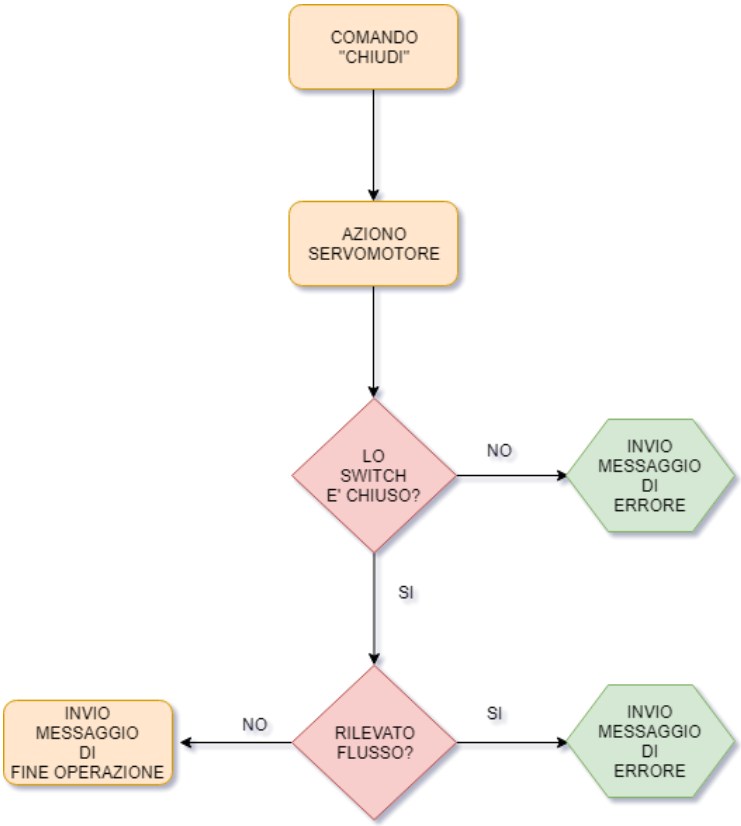
Nello schema a blocchi si può capire, tramite l'orientamento delle frecce, il verso della comunicazione tra i vari dispositivi. Si può notare come il sensore MQ5, avendo uscita analogica debba prima interagire con un ADC, nel nostro caso interno al microcontrollore, e poi tali dati potranno essere elaborati dal MCU.
Inoltre, è importante notare che, Arduino, comunica con il Bridge tramite interfaccia seriale

Degni di nota sono i due sensori di chiusura e di flusso che hanno il compito di verificare se, pur avendo inoltrato il comando, tutto è stato svolto correttamente.
Quello di chiusura è stato applicato sulla paratia che blocca il flusso del gas in modo da ottenere un feedback. L'altro, quello di flusso, simulato con uno switch premuto manualmente, serve a controllare eventuali perdite della saracinesca di chiusura del condotto.

Le funzionalità svolte dal microcontrollore possono essere schematizzate in questo modo:



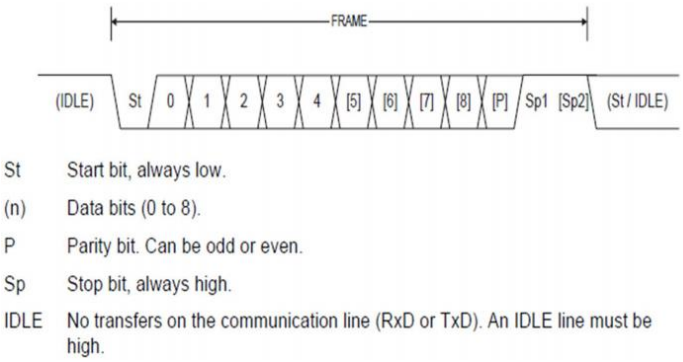
Quando il microcontrollore riceve un comando di chiusura (o apertura) si vanno ad interrogare i due sensori:



nel momento in cui il microcontrollore rileva un errore manda, tramite linea seriale, al bot telegram, il tipo di errore. Infatti Arduino è capace di inviare una stringa in modo da far capire al bot quale sensore ha inviato una risposta contradditoria.

Comunicazione Seriale Asincrona:

Arduino ed il bridge comunicano usando il protocollo seriale asincrono via USB. Il trasmettitore ed il ricevitore si sincronizzano usando i dati stessi: il trasmettitore invia inizialmente un bit di partenza, poi il dato vero e proprio (tipicamente da cinque a otto bit, con il bit meno significativo per primo), un bit opzionale di parità, e infine un tempo di stop che può avere diverse lunghezze tipiche (uno, uno e mezzo o due tempi di bit).



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

Dallo schema è possibile vedere i livelli logici del bit di start, sempre basso e quelli di stop sempre alto.
La comunicazione asincrona non prevede la comunicazione del clock, ma i due sistemi devono essere a conoscenza della *baud rate* usata dai due agenti.
Infatti, sapendo la frequenza essi potranno campionare il segnale ricevuto adeguatamente per leggere correttamente tutti i livelli logici.

Codice Sorgente:

```
1.  void loop()
2.  {
3.
4.      switch(stato){
5.          case 0:
6.              if(digitalRead(gas_din)==HIGH){//non c'è gas
7.                  ad_value=analogRead(gas_ain);
8.                  scrivi_seriale('w',ad_value);
9.              }
10.             if (digitalRead(gas_din)==LOW){//c'è gas
11.                 ad_value=analogRead(gas_ain);
12.                 scrivi_seriale('a',ad_value);
13.             }
14.             if (Serial.available() > 0) {
15.                 stato = 1;
16.                 break;
17.             }
18.             delay(500);
19.             break;
20.         case 1:
21.
22.             incomingByte = Serial.read();
23.             //applico la decisione dello stato in base al carattere ricevuto dal bot
24.             if(incomingByte == 'c') valv_stato = 0; //chiudi la valvola
25.             if(incomingByte == 'o') valv_stato = 1; //apri la valvola
26.             if(incomingByte == 'i') valv_stato = 2; //ignora emergenza
27.             //}
28.             stato=0;
29.             break;
30.     }
31.
```

In questa porzione di codice è possibile notare la *logica a stati* della prima fase di lavoro.
Arduino invia sempre un valore al bridge in questo formato:
*aXXX_* oppure *wXXX_*. La “a” viene inviata solamente quando l’uscita digitale TTL è a livello 0, in stato di normale funzionamento viene inviato “w”. Quando viene rilevato un dato in attesa da seriale, proveniente dal bridge, cambia lo stato e controlla il byte ricevuto. Il carattere “_” serve a comunicare che il messaggio è finito.

I possibili valori che può assumere il byte proveniente dal bridge sono 3:

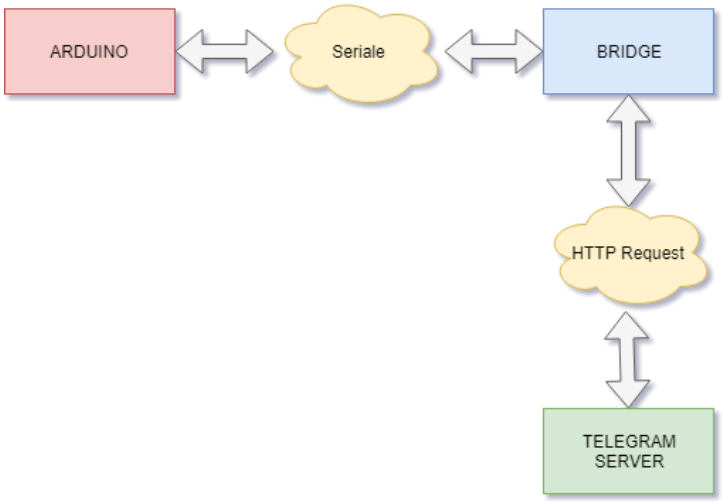
- ‘’c’’: comunica l’intenzione di chiudere la valvola
- ‘’o’’: comunica l’intenzione di aprire la valvola
- ‘’i’’: Ignora l’allarme

```
switch(valv_stato){
        case 0:
            //chiudo la valvola
            if(valvola == 1){
                valvola = 0;
                chiudi_valvola();
                // gli switch sono normalmente aperti.
                // se premo lo switch il circuito si chiude e passa corrente

                //if(digitalRead(close_sensor) == LOW || digitalRead(flow_sensor) == HIG
                if(digitalRead(close_sensor) == LOW ){
                    /*
                    Se il circuito è aperto c’è un errore
                    */
                    scrivi_seriale('e',0);
                    scrivi_seriale('f',0);// f = flow
                    // "f0_" = errore per il controllo del flusso
                }else{
                    scrivi_seriale('T',0);//True
                }

            }
            break;
        case 1:
            //apro la valvola
            if(valvola == 0){

                valvola = 1;
                apri_valvola();

                if(digitalRead(close_sensor) == HIGH){
                    /*
                    se il circuito è aperto c’è un errore
                    */
                    scrivi_seriale('e',1);
                    scrivi_seriale('f',1);// f = flow
                    // "f0_" = errore per il controllo del flusso
                }else{
                    scrivi_seriale('T',1);//True
                }

            }
            break;
        case 3:
            //ignoro l'emergenza
            break;
    }
```

In quest’ultimo *switch-case* è possibile vedere che il microcontrollore azionerà un servomotore che andrà a chiudere la valvola in base al comando ricevuto. Come si può vedere, in caso di errore nella *chiusura/apertura* della saracinesca si manda un messaggio di errore a seconda del sensore che fornisce un feedback errato. Invece se tutto funziona normalmente invia un carattere di check al bot con *“T”*.

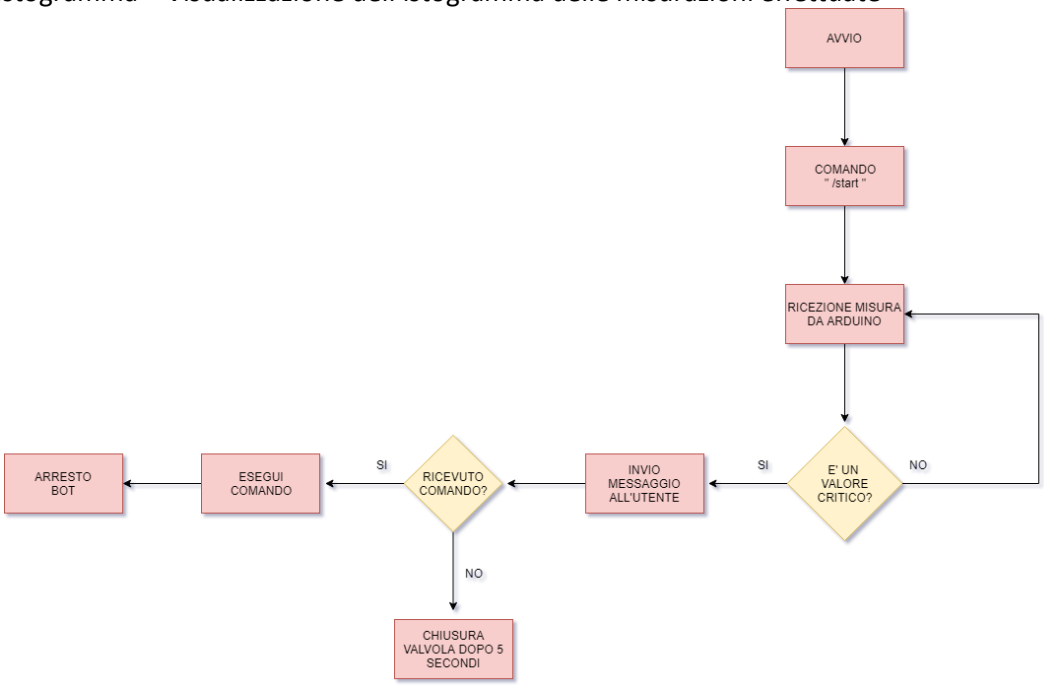
Il Bridge:

Il Bridge, in questo prototipo viene simulato da un Notebook, ma è possibile implementarlo tramite una Raspberry Pi 3, oppure la sua versione più minimale come la Pi Zero, aggiungendo le necessarie periferiche I/O.
Il Bridge esegue il cuore di tutto il prototipo, ovvero lo script Python del BOT Telegram



Il BOT telegram è stato scritto in python utilizzando la libreria pyTelegramBotAPI.
Grazie all'uso di questa libreria ogni comando affidato al bot sarà gestito con un thread, garantendo maggiore efficienza nei tempi di risposta.
Lo script python ha il compito di intercettare i dati dalla seriale, ricevere i comandi da parte dell'utente e inviare di conseguenza il messaggio ad Arduino.
I comandi disponibili per l'utente sono questi:

- /help - Visualizza la lista comandi disponibili
- /start - Avvia il controllo del sistema
- /chiudi - Chiude la valvola
- /apri - Apre la valvola
- /ignora - Ignora il messaggio di emergenza
- /istogramma – Visualizzazione dell'istogramma delle misurazioni effettuate



Dallo schema a blocchi si può capire il funzionamento del comando "/start". Una volta avviato si entra in un ciclo di controllo che terminerà quando il sistema rileverà un valore di gas sopra la norma.
Quando accade ciò l'utente avrà a disposizione 5 secondi, terminati questi, in assenza di risposta, il bot chiuderà automaticamente la valvola.

Fault Analysis:

Il sensore MQ5 è tarato per rilevare come allarme valori di gas superiori a 500 ppm, ma per anticipare l'aumento del gas o gestire errori di comunicazione del sensore, si è inserita una fault analysis basata su una gaussiana.
Infatti, si sono raccolti 3000 valori di misurazione in condizione di normale funzionamento, e solo per brevi periodi di tempo sono stati inseriti dei valori di gas sopra la norma.
In questo modo si è centrata la gaussiana nella parte ottimale di funzionamento, per ogni nuova misurazione si andrà a calcolare la probabilità, se è al di sotto di una soglia il sistema si arresta e chiude il condotto.

Script Python del comando "/start":

```
@bot.message_handler(commands=['start'])
def start_control(message):
    global chiuso , media, sigma,stato,VAL_MAX,ignora

    SOGLIA =calcola_probabilita(VAL_MAX,media,sigma)

    flag = 0
    fuga = 0
    if stato == 1:
        bot.send_message(message.chat.id, "IMPOSSIBILE : SONO GIA' IN ASCOLTO")
        return
    if stato == 0:

        stato = 1
        while True:
            while ser.inWaiting() > 0:

                data_str = ser.read(ser.inWaiting()).decode('ascii')
                val=estrai_valore(data_str)
                if int(val) >= 0 or val != '':
                    prob = calcola_probabilita(int(val), media, sigma)
                else:
                    print("errore calcolo probabilità")
                    prob = calcola_probabilita(0, media, sigma)

                if data_str[0] == 'w' and flag == 0:
                    print("tutto ok")
                    flag =1
                    fuga = 0
                    '''''
                    flag serve a far entrare una sola volta nel ciclo il programma
                    in modo da mandare un solo messaggio all'utente
                    '''
                    time.sleep(0.5)
                    bot.send_message(message.chat.id, "I sistemi funzionano normalmente. R
esto in attesa")
                '''''
                Qui controllo oltre al carattere iniziale, anche
se la probabilità della misurazione è sotto la soglia.
                in questo modo se il sensore dovesse non funzionare più la sua uscita
analogica sarà nulla ed arduino invierà sicuramente un segnale che inizierà con 'a'
                '''
                if data_str[0] == 'a' or prob < SOGLIA:
```

```
                    print("attenzione, il bot verrà arrestato. Il condotto verrà chiuso. RILEVATO GAS")
                    fuga = 1
                    bot.send_message(message.chat.id, "Emergenza : Rilevata fuga di gas : " + val +
                                                      "\nChiudere il condotto ? /chiudi"
                                                      "\nIgnorare il problema ? /ignora"
                                                      "\n\nIL RILEVAMENTO VERRA' FERMATO!!"
                                                      "\n per continuare :"
                                                      "\n/start")
                    time.sleep(0.5)
                    markdown = """
                                *in assenza di risposta, verrà chiuso automaticamente tra 5 secondi*
                                """
                    ret_msg = bot.send_message(message.chat.id, markdown, parse_mode="Markdown")
                    break #esco dal while della ricezione da seriale
                time.sleep(0.5)

            if fuga == 1:
                '''''alla fine del while di ricezione da seriale controllo il motivo dell' uscita
                perchè è possibile o che arduino sia malfunzionante e quindi non manda più niente
                oppure
                caso principale che sono uscito perchè ho rilevato una fuga di gas'''
                break # esco dal while infinito
    stato=0
    time.sleep(5)
    if ignora == 0:
        if chiuso == 0:
            # richiamerò la funzione che mandail messaggio di chiusura ad arduino
            chiuso = 1
            markdown = """
                       *Valvola chiusa*
                       """
            bot.send_message(message.chat.id, markdown, parse_mode="Markdown")
            chiudi_valvola()
        else:
            markdown = """
                        _La valvola è già chiusa_
                        """
            bot.send_message(message.chat.id, markdown, parse_mode="Markdown")
            chiudi_valvola()
    else:
        markdown = "*messaggio di allarme ignorato*" \
                   "\nDigitare /start per ricominciare il controllo del sistema"
        bot.send_message(message.chat.id, markdown, parse_mode="Markdown")
        ignora = 0
```

Script Python del comando “/chiudi” ed “/apri”:

```
def controllo_errore_chiusura_apertura():
    mex =""
    flag = 0
    mex_letto =""
    while True:
        while ser.inWaiting() > 0:
            mex_letto = ser.read(ser.inWaiting()).decode('ascii')
        print(mex_letto)
        for i in mex_letto:
            if i == 'e':
                flag = 1
            if i == 'f':
                flag = 2
            if i == 'T':
                flag = 3
        if flag == 1 or flag == 2 or flag== 3:
            break
    if flag == 1:
        mex += "ERRORE NELLA APERTURA." \
               "NECESSARIA AZIONE MANUALE"
    if flag == 2 and flag == 1:
        mex += "\nFLUSSO NON RILEVATO"
    if flag == 3:
        mex = ""
     return mex


@bot.message_handler(commands=['apri'])
def start_control(message):
    global chiuso
    if chiuso == 1:
        chiuso = 0
        bot.reply_to(message, "apro")
        apri_valvola()
        mess = controllo_errore_chiusura_apertura()
        if mess != "":
            bot.send_message(message.chat.id, mess)
    else:
        bot.reply_to(message, "è già aperta")


@bot.message_handler(commands=['chiudi'])
def start_control(message):
    global chiuso
    if chiuso == 0:
        chiuso = 1
        bot.reply_to(message, "chiudo")
        chiudi_valvola()
        mess = controllo_errore_chiusura_apertura()
        if mess != "":
            bot.send_message(message.chat.id, mess)
    else:
        bot.reply_to(message, "è già chiusa")
```

Come si può vedere tramite la funzione “controllo_errore_chiusura_apertura()” si controlla tramite un “polling” la linea seriale finchè o non arriva un messaggio di errore, in tal caso verrà comunicato all’utente, oppure arriva il feedback di avvenuta operazione.

Conclusioni:

Questo progetto mi ha consentito di acquisire più familiarità con il mondo dei microcontrollori; imparando a utilizzare componenti e sensori da zero basandomi sulle indicazioni dei datasheet per capire come interfacciarli con la board di riferimento.
Il sistema nella sua versione finale si presenta come un oggetto utile da impiegare per tenere monitorata la presenza di gas GPL in una stazione di rifornimento sopra livelli critici, tuttavia per poterlo impiegare in ambiti reali sarebbe necessaria la sostituzione di alcuni componenti in favore di altri qualitativamente migliori, più affidabili e costosi.

Ottimizzazioni e miglioramenti futuri:

I miglioramenti fondamentali potranno essere raggiunti sostituendo la comunicazione seriale via cavo con un Bluetooth low energy ed inserendo un layer bridge vero e proprio con un sistema di calcolo performante come una Raspberry Pi3 oppure un Odroid, capaci di eseguire in modo costante consumando poca energia il bot telegram.
Ugualmente si potrebbe migliorare il prototipo istaurando una comunicazione senza fili tra arduino e tutti i sensori adottati.
Grazie all'uso di più sensori sarà possibile implementare nello script eseguito dal bridge, un sistema di majority voting, in modo da comunicare all'utente il livello di pericolosità con maggior certezza.
Inoltre, sarebbe molto più efficiente sostituire il micro-switch facente funzione di flussometro con uno vero e proprio.
Tramite l'adozione del flussometro è possibile impiegare la ventolina, con apposito adattatore per simulare il passaggio del gas, usando però uno stadio di alimentazione differente poiché essa richiederebbe una tensione di 12v, che per motivi pratici non è stata prevista nel prototipo esposto, ma è stato lasciato lo spazio sulla board.