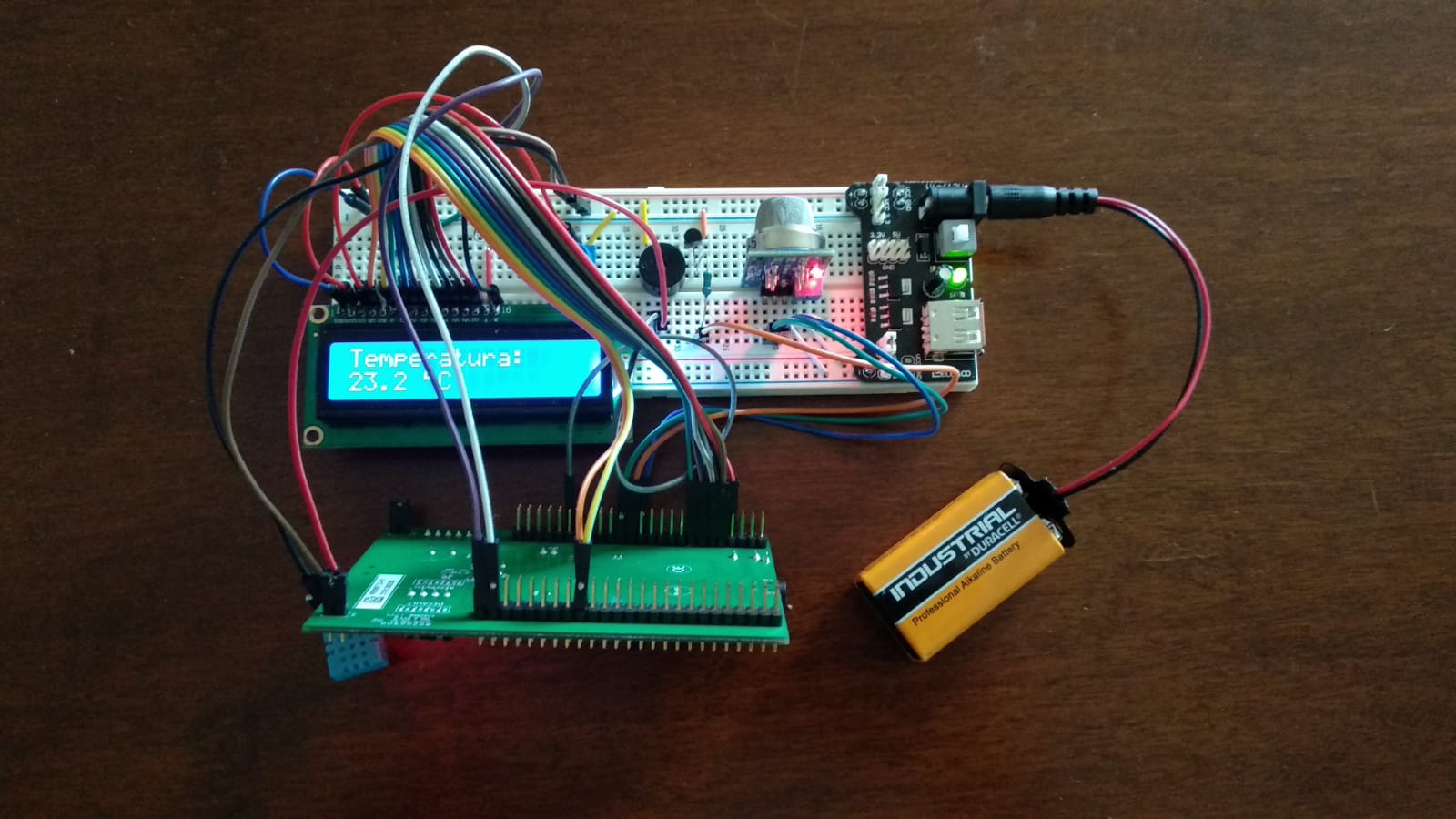
Università degli studi di Modena e Reggio Emilia

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:

1. Introduzione
2. Studio di fattibilità
   1. Requisiti e ambito di applicazione
   2. Vincoli
   3. Valutazione dei costi
3. Componenti impiegati nel prototipo
   1. Arduino Uno R3
   2. Sensore di gas MQ5
   3. Servomotore MG946R
4. Implementazione
   1. Schema a blocchi dell’applicazione
   2. Sketch Di Arduino
   3. Script python del BOT di Telegram
5. Conclusioni
   1. Ottimizzazioni e miglioramenti futuri

# 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:

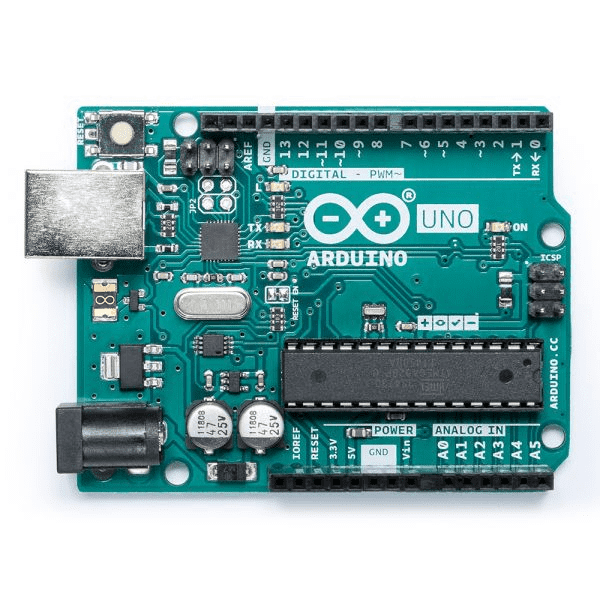
Il prodotto finale per il corretto funzionamento dovrà essere disposto lontano da fonti di calore, in luoghi chiusi e non eccessivamente umidi.

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

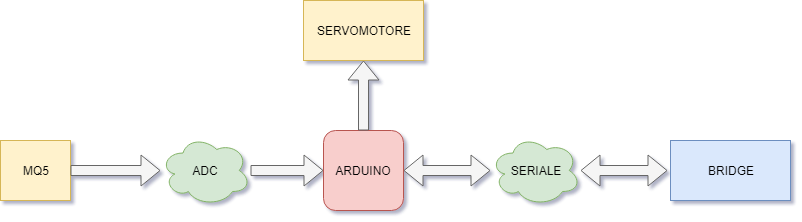
0 – se viene rilevata la fuga di gas

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.

# Implementazione:

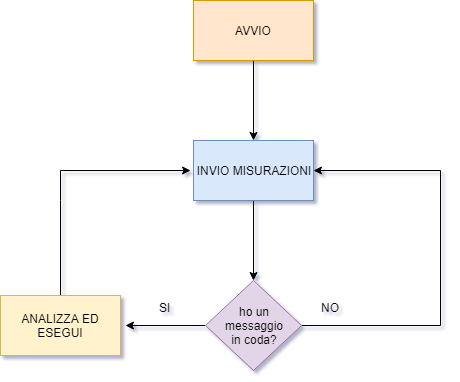
## Schema a blocchi:



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

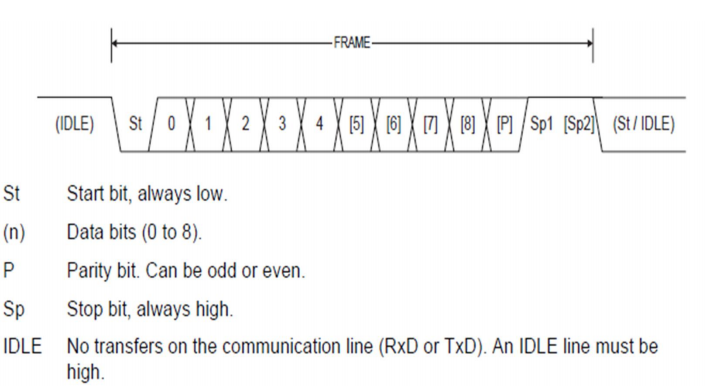
Le funzionalità svolte dal microcontrollore possono essere schematizzate in questo modo:



MANCA LA DESCRIZIONE DEL PROTOCOLLO RS232 E CODICE SORGENTE

## 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).

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. {
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:
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. }

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.

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

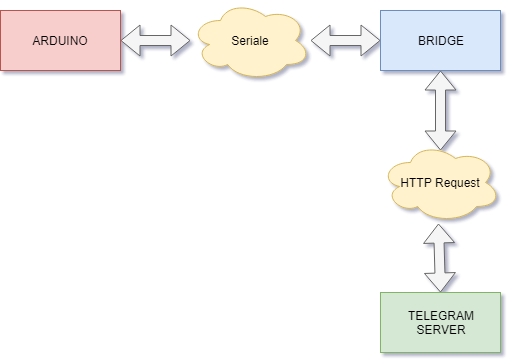
1. **switch**(valv\_stato){
2. **case** 0:
3. //chiudo la vavola
4. valvola = 0;
5. **for**(pos = 0; pos < 180; pos += 1)
6. {
7. myservo.write(pos);
8. delay(15);
9. }
10. **break**;
11. **case** 1:
12. //apro la valovla
13. **if**(valvola != 1) valvola = 1;
14. **for**(pos = 180; pos>=1; pos-=1)
15. {
16. myservo.write(pos);
17. delay(15);
18. }
19. **break**;
20. **case** 2:
21. //ignoro l'emergenza
22. **break**;
23. }
24. valv\_stato = -1;
26. delay(500);
27. }

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

## 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.

Il Bridge esegue il cuore di tutto il prototipo, ovvero lo script Python del BOT Telegram

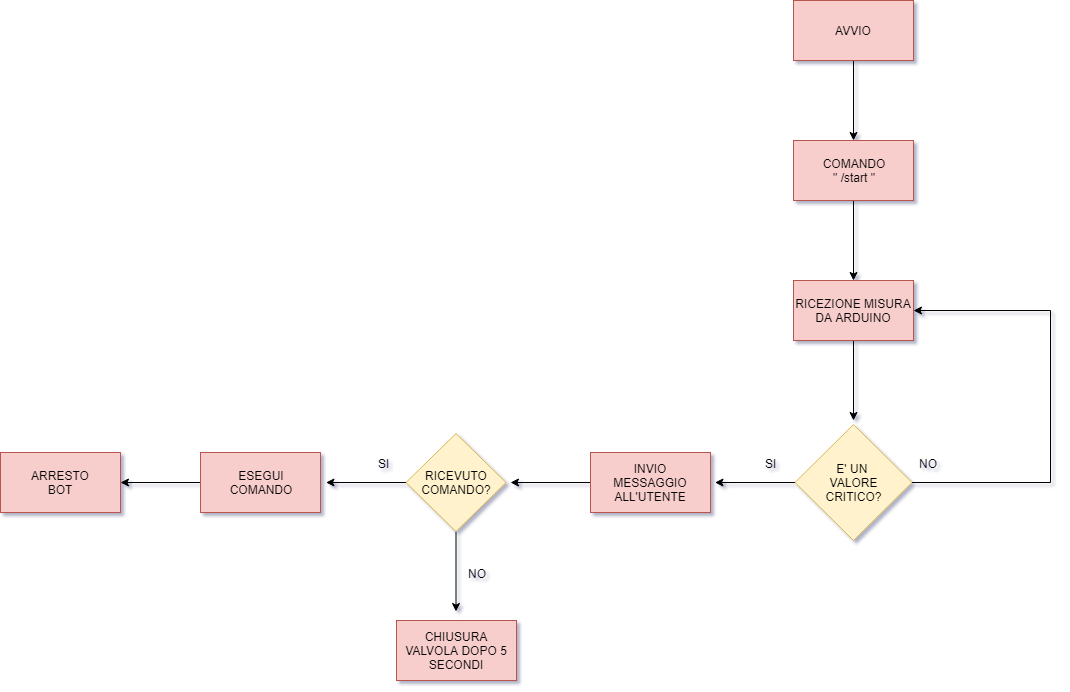


Il BOT telegram è stato scritto in python utilizzando la libreria pyTelegramBotAPI.  
Grazie all’uso di questa libreria ogni comando affidato al bot sarà gestito con un thread, garantendo maggiore efficienza nei tempi di risposta.

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.

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 bastata 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”:

1. @bot.message\_handler(commands=['start'])
2. **def** start\_control(message):
3. **global** chiuso , media, sigma,stato,VAL\_MAX,ignora
5. SOGLIA =calcola\_probabilita(VAL\_MAX,media,sigma)
7. flag = 0
8. fuga = 0
9. **if** stato == 1:
10. bot.send\_message(message.chat.id, "IMPOSSIBILE : SONO GIA' IN ASCOLTO")
11. **return**
12. **if** stato == 0:
14. stato = 1
15. **while** True:
16. **while** ser.inWaiting() > 0:
18. data\_str = ser.read(ser.inWaiting()).decode('ascii')
19. val=estrai\_valore(data\_str)
20. **if** int(val) >= 0 **or** val != '':
21. prob = calcola\_probabilita(int(val), media, sigma)
22. **else**:
23. **print**("errore calcolo probabilità")
24. prob = calcola\_probabilita(0, media, sigma)
26. **if** data\_str[0] == 'w' **and** flag == 0:
27. **print**("tutto ok")
28. flag =1
29. fuga = 0
30. '''''
31. flag serve a far entrare una sola volta nel ciclo il programma
32. in modo da mandare un solo messaggio all'utente
33. '''
34. time.sleep(0.5)
35. bot.send\_message(message.chat.id, "I sistemi funzionano normalmente. Resto in attesa")
36. '''''
37. Qui controllo oltre al carattere iniziale, anche se la probabilità della misurazione è sotto la soglia.
38. 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'
39. '''
40. **if** data\_str[0] == 'a' **or** prob < SOGLIA:
41. **print**("attenzione, il bot verrà arrestato. Il condotto verrà chiuso. RILEVATO GAS")
42. fuga = 1
43. bot.send\_message(message.chat.id, "Emergenza : Rilevata fuga di gas : " + val +
44. "\nChiudere il condotto ? /chiudi"
45. "\nIgnorare il problema ? /ignora"
46. "\n\nIL RILEVAMENTO VERRA' FERMATO!!"
47. "\n per continuare :"
48. "\n/start")
49. time.sleep(0.5)
50. markdown = """
51. \*in assenza di risposta, verrà chiuso automaticamente tra 5 secondi\*
52. """
53. ret\_msg = bot.send\_message(message.chat.id, markdown, parse\_mode="Markdown")
54. **break** #esco dal while della ricezione da seriale
55. time.sleep(0.5)
57. **if** fuga == 1:
58. '''''alla fine del while di ricezione da seriale controllo il motivo dell'uscita
59. perchè è possibile o che arduino sia malfunzionante e quindi non manda più niente
60. oppure
61. caso principale che sono uscito perchè ho rilevato una fuga di gas'''
62. **break** # esco dal while infinito
63. stato=0
64. time.sleep(5)
65. **if** ignora == 0:
66. **if** chiuso == 0:
67. # richiamerò la funzione che mandail messaggio di chiusura ad arduino
68. chiuso = 1
69. markdown = """
70. \*Valvola chiusa\*
71. """
72. bot.send\_message(message.chat.id, markdown, parse\_mode="Markdown")
73. chiudi\_valvola()
74. **else**:
75. markdown = """
76. \_La valvola è già chiusa\_
77. """
78. bot.send\_message(message.chat.id, markdown, parse\_mode="Markdown")
79. chiudi\_valvola()
80. **else**:
81. markdown = "\*messaggio di allarme ignorato\*" \
82. "\nDigitare /start per ricominciare il controllo del sistema"
83. bot.send\_message(message.chat.id, markdown, parse\_mode="Markdown")
84. ignora = 0

# Conclusioni:

Questo progetto mi ha consentito di acquisire più familiarità con il mondo dei microcontrollori; imparando a utilizzare componenti e sensori da zero basandoci 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 usando una comunicazione senza fili tra arduino ed il sensore MQ5, e dare la possibilità ad arduino di interfacciarsi con più di un sensore, comunicare i dati al bridge che con un sistema di majority voting riesca a comunicare all’utente il livello di pericolosità con maggior certezza.