
UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

---

Corso di laurea triennale

in Ingegneria Informatica e dell'Automazione

# STUDIO E SVILUPPO DI COMPONENTI DI UN SISTEMA SOTTOMARINO PER IL MONITORAGGIO DELLO STATO FISIOLOGICO DEL SUB DURANTE LA RACCOLTA DATI

## Study and development of components of an underwater system for monitoring the physiological state of the diver during data collection

Relatore:
**Prof. David Scaradozzi**

Tesi di laurea di:
**Michele Graziano de Virgilio**

Correlatori:
**Nicolò Ciuccoli**
**Veronica Bartolucci**

---

Anno Accademico 2019/2020

# Indice

# Introduzione

## Background

Fin dall'antichità l'uomo è sempre stato affascinato dall'enorme massa d'acqua che avvolgeva le terre su cui viveva. Questa attrazione lo ha portato a tentare di esplorarla, dapprima soltanto in apnea e in seguito ingegnandosi sempre di più. Risalgono al XIV secolo i primi usi, un po' più professionali della campana subacquea, letteralmente una campana rigida, utilizzata per trasportare gli uomini dalla superficie in profondità, o per compiere lavori sottomarini, o per semplice esplorazione: solitamente sono sospese da un cavo e sollevate e abbassate da un argano, da una piattaforma di supporto di superficie. Successivamente, dall'idea della campana da immersione, viene progettato il primo elmo da palombaro rifornito di aria compressa. Con l'aggiunta della prima muta impermeabile collegata a tenuta stagna, nel XIX secolo si crea il primo scafandro da immersione; anche grazie al supporto della ricerca scientifica e tecnologica, si sono compiuti passi da gigante.

Dal XX secolo si iniziarono a sviluppare attrezzature SCUBA (self-contained underwater breathing apparatus) di due tipologie, aperte o chiuse: le prime permettono una respirazione senza il ricircolo dell'aria, mentre le seconde ne prevedono l'espulsione una volta respirata.

Ovviamente i sistemi attuali sono molto più precisi e sicuri dei primi sviluppati, ma ugualmente si basano su questi principi. Quando il sistema è a ciclo aperto, si utilizza un erogatore di pressione a due stadi chiamati primo stadio e secondo stadio che sono collegati tra loro. [1] Il primo stadio può essere a pistone o a membrana ed è collegato direttamente alla rubinetteria della bombola: esso provvede ad abbassare l'alta pressione del gas (che va dai 200 ai 300 bar), che verrà portato verso il subacqueo tramite "fruste" (tubi resistenti alla pressione).

Poiché il subacqueo non può respirare aria che non sia alla stessa pressione dell'ambiente che lo circonda, con l'erogatore essa viene portata al primo stadio, di norma regolato alla pressione ambientale sommata a un valore che oscilla da 7 a 10 bar,

per poi essere portata alla pressione ambientale dal secondo stadio, che abbassa e modula la pressione del gas adattandola a quella ambientale ed è misurata tramite membrane e leveraggi al suo interno. Il secondo stadio è la parte terminale dell'erogatore, quella che è trattenuta dalla bocca del subacqueo che, così facendo, riesce a respirare senza il minimo sforzo. [2]

## Scopo della tesi

Lo scopo della tesi è fornire supporto al progetto europeo DiveSafe, presentato nel capitolo successivo, andando a studiare e sviluppare componenti, sia hardware che software, per il monitoraggio dello stato fisiologico del sub, durante immersioni per la raccolta dati. In dettaglio, sono stati adottati due profondimetri che si interfacciano con una Raspberry pi Zero W, che funge da Motherboard per il sistema integrato, per riuscire a misurare mediante un algoritmo, la frequenza di respiro del sub. Tale algoritmo può essere utilizzato per rilevare anomalie nel respiro dello stesso operatore subacqueo, sfruttando i soli segnali di pressione ambientale e di pressione di primo stadio dell'erogatore. [2]

## Panoramica della tesi

Nella prima parte della tesi, si illustra inizialmente cos'è il progetto DiveSafe nel dettaglio, per poi spostarsi sulla seconda parte della tesi, relativa alle componenti hardware in cui verrà mostrato un primo render approssimativo dello scooter subacqueo (DPV), che poi verrà assemblato con il file della DS (Docking Station) andando a progettare, realizzare e successivamente stampare in 3D un pezzo che fungerà da collegamento tra i due. In seguito, è presentata la progettazione riferita all'inserimento dei componenti all'interno del case diver, o case della "Diver Motherboard", un contenitore che verrà "indossato" dal sub e che porterà sempre con sé, in quanto conterrà la scheda Raspberry, che oltre ad essere collegata ai due profondimetri sopra menzionati, è collegata a tre sensori che monitoreranno lo stato fisiologico del subacqueo costantemente. Lo scopo del disegno del case è quello di economizzare gli

spazi, per cercare di riuscire ad inserire tutto il necessario, e in particolar modo l'agganciare esternamente i due profondimetri, che vedremo nello specifico in seguito.

La terza parte della tesi consisterà nell'elaborazione e realizzazione di un algoritmo di rilevazione del respiro, in maniera tale da riuscire a calcolare la frequenza di respiro del sub: quest'ultima verrà processata da un'applicazione presente sul tablet (come vedremo nell'insieme di componenti che fanno parte del sistema, c'è anche un tablet dotato di cover resistente ad alte pressioni) e restituirà un messaggio di avviso al sub in tempo reale.

# 1. Progetto Divesafe

DiveSafe è un progetto cofinanziato dal programma EMFF (European Maritime and Fisheries Fund) dell'Unione europea. Il titolo completo di tale progetto è: "Integrated system for scientific and environmental underwater surveys, with advanced health & safety features". [3]


**Fig. 1.1. Logo del progetto**

Come suggerito dal titolo, esso rappresenta dunque un sistema integrato per indagini svolte in ambito sottomarino a livello scientifico e ambientale, prevedendo funzionalità aggiuntive avanzate riguardanti la salute e la sicurezza dei subacquei.

Al giorno d'oggi gli scooter subacquei o DPV (Diver Propulsion Vehicle) commerciali sono sostanzialmente dei sistemi meccanico-elettrici senza una intelligenza embedded. Il principale obbiettivo di DiveSafe è l'integrazione della struttura di uno scooter DPV originale, già pronto per il mercato, con una motherboard (inserita all'interno di una Docking Station) per la gestione di sottosistemi esterni (batteria, propulsori, luci, embedded camera, etc.) creando un nuovo prodotto meccatronico capace di comunicare con la superficie, auto-localizzarsi ed interfacciare i suoi servizi ad un tablet subacqueo.

Lo scopo di DiveSafe è quindi quello di consegnare al mercato subacqueo (principalmente professionale/scientifico e in secondo luogo anche ricreativo) un prodotto innovativo che consenta ai subacquei di condurre in modo sicuro ed efficiente compiti riguardanti l'osservazione dell'ambiente marino, la documentazione fotografica

e fotogrammetrica di esso, l'esplorazione di aree sconosciute, la ricerca, la visita turistica di ampi siti sommersi, etc.

## 1.1 Architettura generale

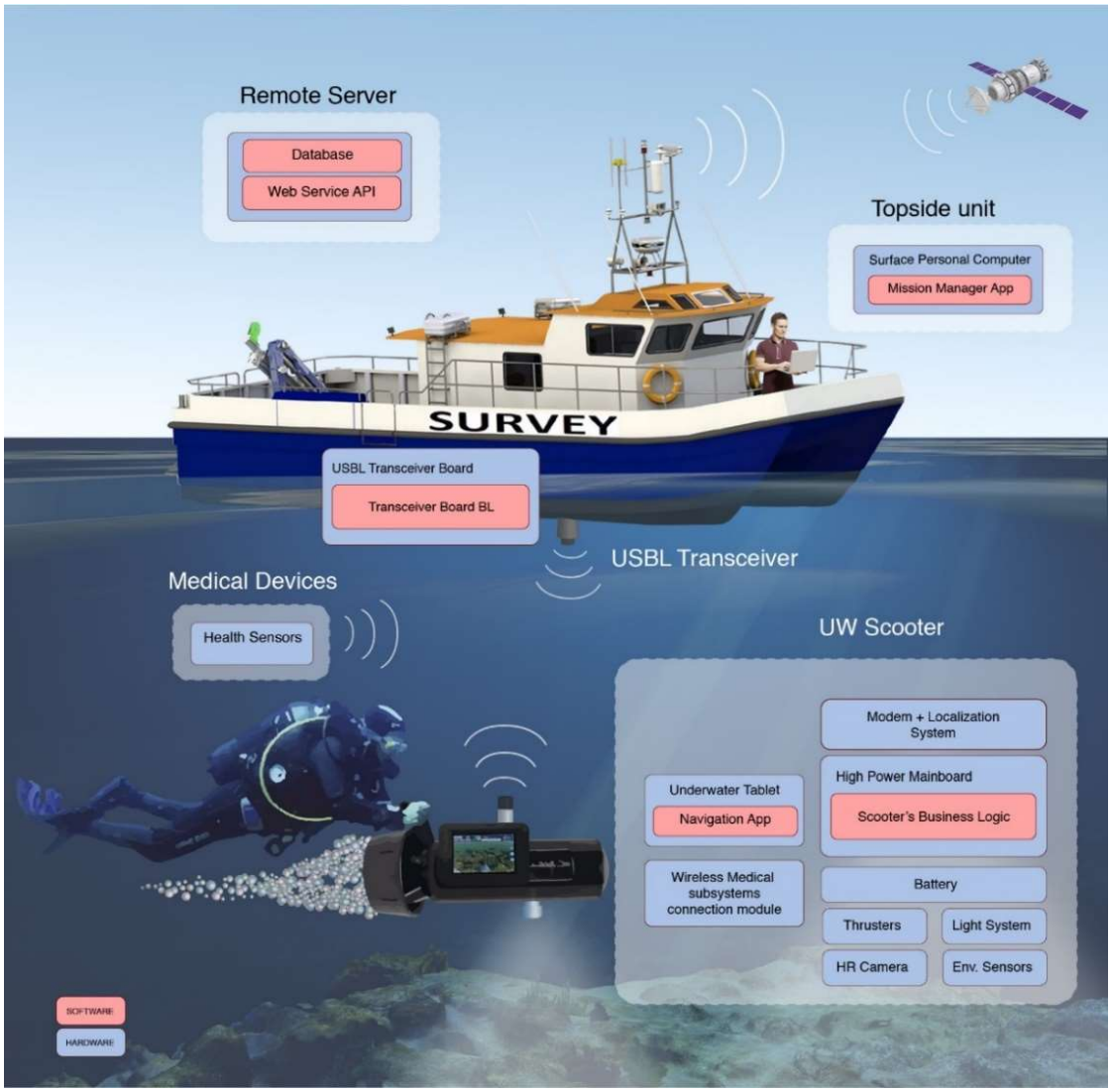



**Fig. 1.1.1. Architettura del sistema**

L'innovativo scooter DPV realizzato da DiveSafe sarà un componente di una infrastruttura generale più complessa, composta da tre blocchi. Lo schema generale del sistema distribuito DiveSafe deriva dalla configurazione tipica di un cloud system per l'Internet of Things (IoT), dove lo scooter sottomarino usato dai subacquei sarà arricchito con la capacità di raccolta e presentazione dei dati con una parziale connessione Internet.

L’infrastruttura studiata è composta da tre diversi elementi, come mostrato nella figura superiore:

1) Modulo server remoto
2) Unità Superficiale
3) Modulo Subacqueo:
    - Scooter subacqueo con il suo sistema interno
    - Dispositivi medici indossati dal subacqueo.

Il modulo Server Remoto contiene il database e gestisce il processo di ricostruzione 3D e l’infrastruttura cloud è a sua volta composta da:

- Database MySQL
- Interfaccia web per operazioni CRUD in back-end
- RESTful API per interagire con altri dispositivi
- Modulo per il motore grafico 3D.

L’unità superficiale ha le seguenti funzionalità:

- Gestisce il database
- Gestisce la comunicazione con gli altri moduli e/o con gli altri dispositivi del sistema
- Richiede una ricostruzione 3D tramite le foto acquisite in missione attraverso il modulo per il motore grafico 3D

Il modulo subacqueo è interiormente diviso in tre sistemi:

- Tablet Subacqueo (UT), collocato in un case impermeabile che garantisce inoltre una interfaccia touchscreen che supporta il subacqueo durante l’immersione
- Docking Station (DS), collocata all’interno del DPV, in grado di acquisire e scambiare dati che arrivano da diversi sensori ambientali, da una camera ad alta risoluzione (HR) e da un sistema di localizzazione acustico fisicamente

collegati a lui. È dotato inoltre di una control board embedded che gestisce tutta l'elettronica e che implementa tutta la logica. La control board è integrata ad un nodo Wi-Fi per la comunicazione con il tablet, con la superficie e con l'unità di controllo wearable del subacqueo.

- Dispositivi medici indossati dal subacqueo, per il monitoraggio della glicemia, del battito cardiaco, del respiro e per le fasi di decompressione.

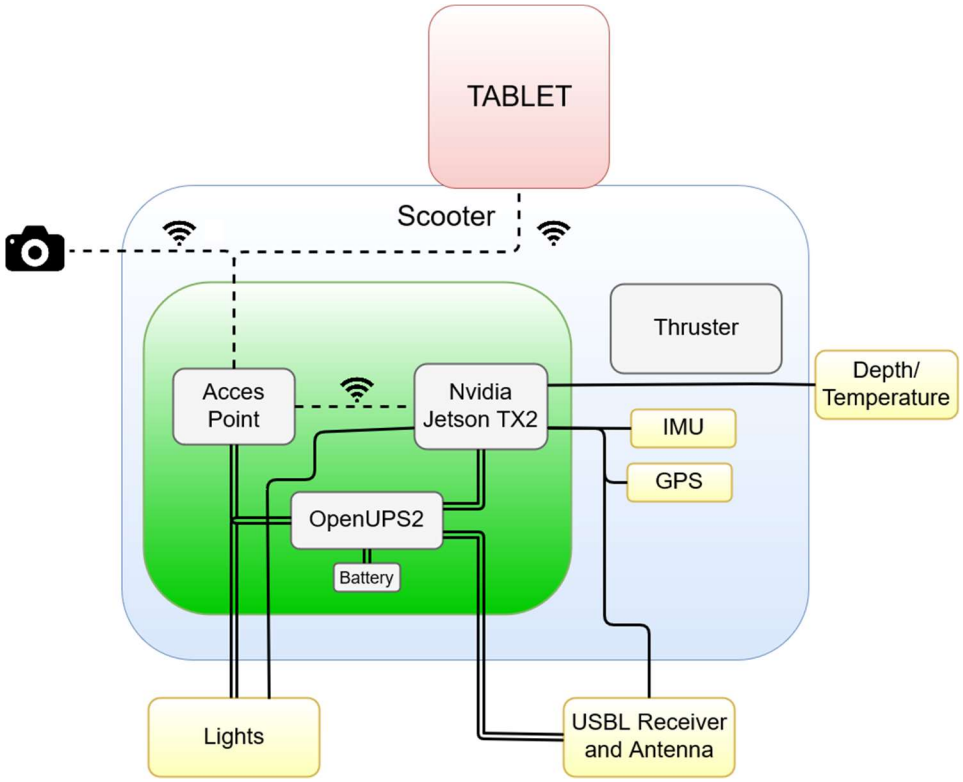



**Fig. 1.1.2. Architettura Interna DPV 1**

I requisiti fondamentali per il funzionamento della Docking Station sono:

- Router Wi-Fi che permette la connessione del tablet subacqueo alla mainboard dello scooter ed alla camera HR
- Pacco batterie Li-ion o LiFePO4 con una circuiteria speciale per il controllo del flusso di potenza e ricarica
- Connettore impermeabile con bus lines per equipaggiare la USBL ed i diversi tipi di sensori ambientali

La mainboard scelta è la NVIDIA Jetson TX2, la cui batteria integrata andrà ad alimentare a sua volta i sensori connessi ad essa.

## 1.2 Architettura del sistema sensoristico per il sub

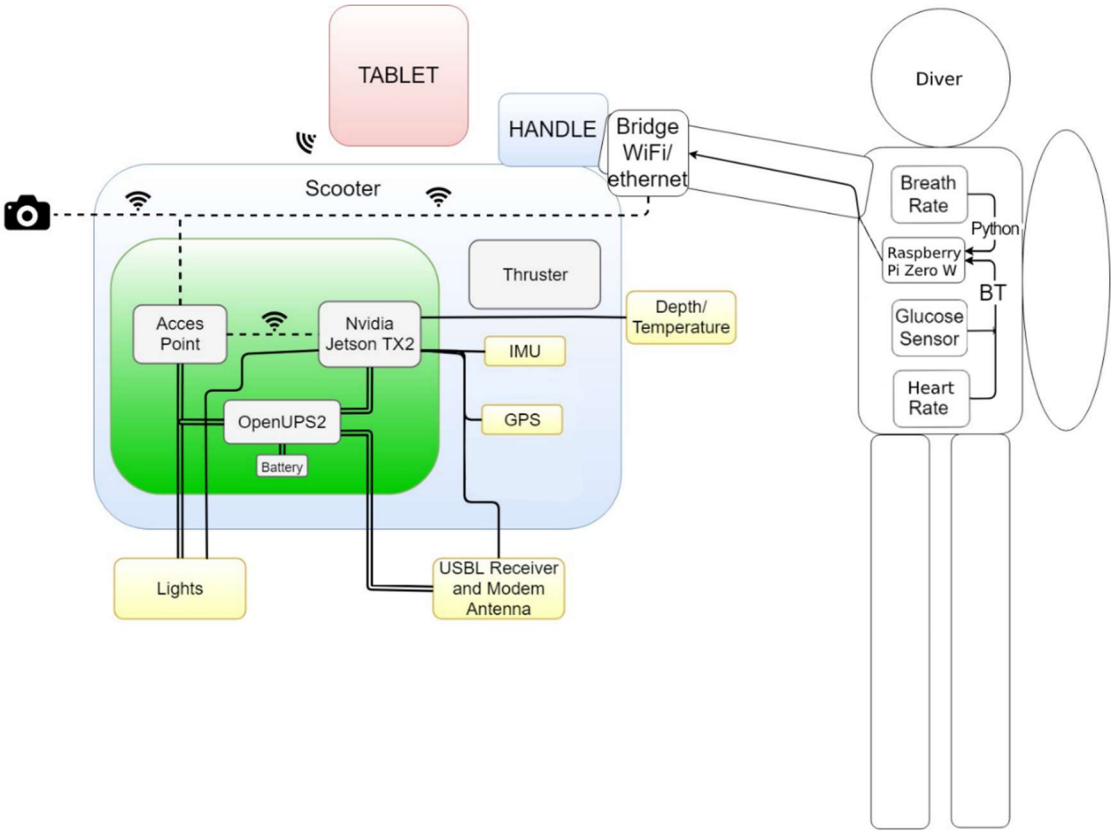



**Fig. 1.2.1 Architettura**

La comunicazione del sistema sensoristico per il subacqueo con lo scooter DPV è gestita tramite un bridge WiFi/Ethernet tra la NVIDIA Jetson TX2 e la Raspberry Pi Zero W. Quest'ultima funge da motherboard per il sistema integrato, avrà come sistema operativo Raspbian ed avrà connessa a sé tre sensori per monitorare lo stato fisiologico del subacqueo. Due sensori saranno collegati alla Raspberry tramite una delle ultime versioni del protocollo Bluetooth, il Bluetooth Low Energy BLE e sono il Glucose Sensor e l'Heart Rate Sensor. Mentre la rilevazione della frequenza di respiro, non è a carico di un sensore apposito, ma di un codice scritto in Python caricato direttamente sulla

Raspberry, ed eseguito, il quale si servirà delle misurazioni di due profondimetri per questo scopo. Inoltre, la Raspberry contiene anche l'algoritmo per le fasi di decompressione del subacqueo.

# 2. Software utilizzati

Nel corso di questo capitolo, verranno presentati brevemente gli strumenti software utilizzati per lo svolgimento di questo lavoro di tesi. In particolare, verranno dettagliati il software CATIA, fondamentale per lo svolgimento della progettazione meccanica, e il programma Visual Studio Code, utilizzato per la scrittura dell'algoritmo per il monitoraggio del respiro del subacqueo.

## 2.1 CATIA

CATIA (acronimo di Computer Aided Three dimensional Interactive Application) è una piattaforma commerciale di tipo CAD/CAE/CAM. Questo sistema è largamente utilizzato nell'ambito dell'industria tecnica ingegneristica, quali il settore aerospaziale, automobilistico, navale.


**Fig. 2.1.1. Logo di CATIA**

La progettazione con CATIA avviene in un ambiente completamente parametrico, dove ogni forma o dimensione è parametrizzata, consentendo quindi qualsiasi modifica in ogni momento, che permette al modello di aggiornarsi senza dover ridisegnare il tutto.

Per raggiungere il nostro scopo, verrà sfruttata una minima parte dell'elevata potenzialità di questo software, in particolar modo, vedremo:

- **Part design**: permetterà la creazione di un singolo pezzo nell'ambiente di sviluppo del software, col quale sarà generato un file di estensione .CATPart. Questo tipo di file sarà associato ai singoli pezzi, ad esempio l'aggancio dello scooter, come si vedrà in seguito.

- **Part assembly**: andrà a generare un file .CATProduct, il quale, nel nostro caso "punterà" ad ogni singolo .CATPart, e permetterà di avere una visione d'insieme del prodotto finale, con i singoli .CATPart assemblati, mediante vincoli particolari, che possono essere di contatto, coincidenza, offset etc.
  Un esempio di questo può essere il prodotto finale dello scooter assieme alla Docking Station.
- **Drafting**: permette di creare un foglio vuoto, utile per riportare proiezioni 2D, eventualmente in scala se si tratta di un componente abbastanza grande, con viste da ogni lato, o solo da alcuni, del singolo pezzo, o dell'insieme.
  Un esempio di questo tipo lo si può trovare nelle appendici.

## 2.2 Visual Studio Code

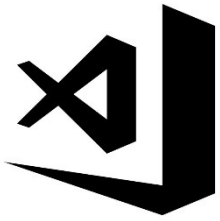


**Fig. 2.2.1. Logo di Visual Studio Code**

Visual Studio Code è un editor di codice sorgente che può essere usato con vari linguaggi di programmazione, tra cui la famiglia di linguaggi C (C, C++, C#), F#, o altri linguaggi web, tra cui PHP, Java, Ruby e molti altri. Molte delle funzioni di Visual Studio Code non sono accessibili attraverso menu o interfacce utente, ma piuttosto attraverso una finestra di comando o un file .json, ad esempio le preferenze dell'utente. Visual Studio Code poggia il suo funzionamento su *Electron*, noto framework con cui è possibile realizzare applicazioni *Node.js.*

La finestra di comando è un'interfaccia a riga di comando, che scompare appena l'utente clicca in un'area al di fuori della finestra o preme una serie di tasti per interagire con qualcosa al di fuori di essa. Sarà utile in quanto si andrà a scrivere il codice relativo all'algoritmo di misurazione della frequenza di respiro in Python. Per far questo, si è dovuto scaricare l'apposito plug-in di Python che riesce a permetterne il debugging.

# 3. Progettazione Meccanica High-End Scooter

## 3.1 Docking station

La Docking Station (DS), come visto nella presentazione di DiveSafe, è la componente che va a rendere effettivamente "intelligente" uno scooter DPV presente sul mercato, restituendo un'esperienza più sicura e sicuramente più accattivante per il sub. Dopo averne elencato le componenti presenti all'interno, vediamo come si presenterebbe senza copertura attorno per evidenziare la posizione dei componenti (fig. 3.1.1):

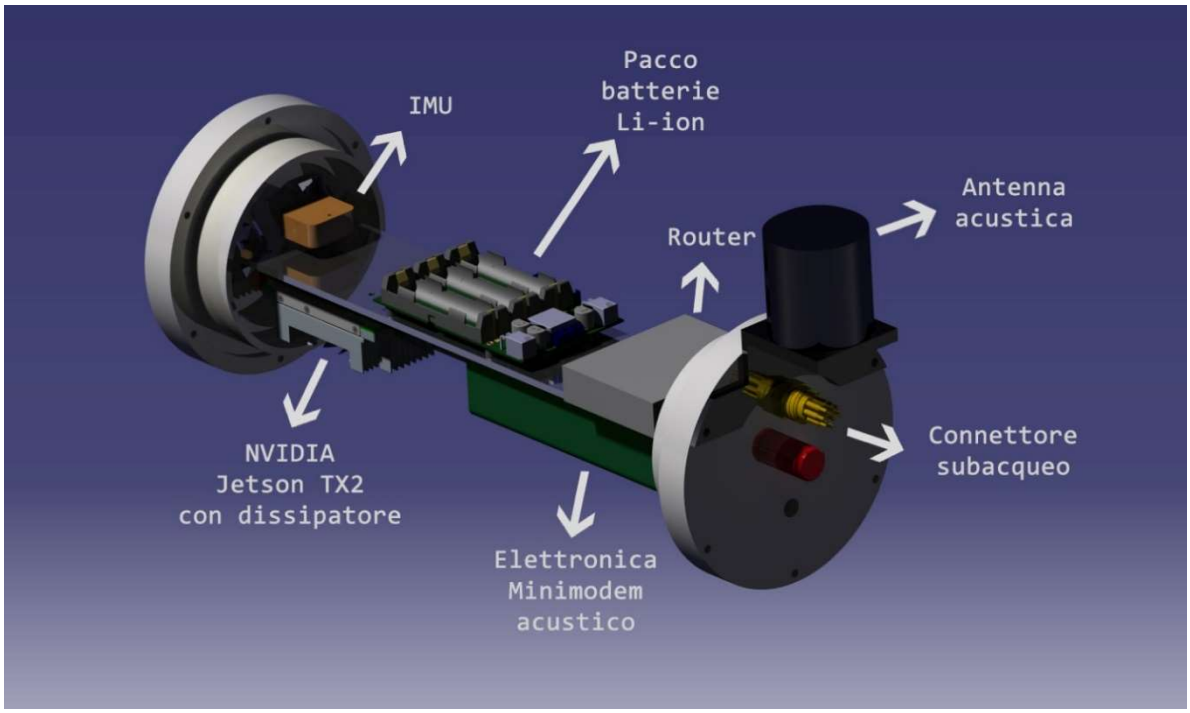



Fig. 3.1.1. Posizionamento componenti nella DS

A questo punto, per procedere con il prossimo passo che consisterà nella progettazione di un sistema di aggancio per il DS allo scooter DPV, introduciamo prima lo scooter di riferimento: quello utilizzato è il Suex XJT, prodotto appunto dalla Suex, azienda italiana con sede in Trentino, tra le più importanti al mondo ad occuparsi di questo settore.

Di seguito ne riportiamo la scheda tecnica:

| | | | |
|---|---|---|---|
| **Use:** | TECNICO / PROFESSIONALE | | |
| **Codice:** | 71381 | **Modello:** | XJt |
| **Lunghezza:** | mm 814 (inch 31,1) | **Larghezza:** | mm 814 (inch 31,1) |
| **Altezza:** | mm 340 (inch 13,4) | **Diametro corpo:** | mm 197 (inch 7,8) |
| **Peso senza batteria:** | kg 14 (lb 30,9) | **Peso con batteria:** | kg 20 (lb 44,1) |
| **Materiali costruttivi:** | Corpo in alluminio | **Forza di trazione statica max:** | N 330 (lb74,2) |
| **Velocità di punta:** | mt/min 85 (ft/min 278) | **Autonomia a velocità massima:** | min 35 |
| **Velocità di crociera:** | mt/min 45 (ft/min 45) | **Autonomia a velocità crociera:** | min 95 |
| **Profondità di collaudo:** | mt 300 (ft 984) | **Massima profondità operativa:** | mt 200 (ft 656) |
| **Galleggiabilità / assetto:** | Trim e galleggiamento neutri | **Temperatura di utilizzo in acqua:** | °C -5/+35 (°F +23/+95) |
| **Tipo di batterie:** | Nimh | **Voltaggio Nominale:** | Volt 26,4 |
| **Capacità nominale:** | Wh 370 | **Alimentazione caricabatteria:** | Volt 90/240 - 50/60 Hz |
| **Tempo massimo per la ricarica:** | h 4,5 | | |

A continuazione, si procederà col disegno in maniera approssimativa dello scooter, così da averlo in dimensioni reali al pc, all'interno del software CAD CATIA. Le uniche dimensioni di nostro interesse difatti, saranno soltanto il diametro corpo e la lunghezza.


Fig. 3.1.2. Render reale

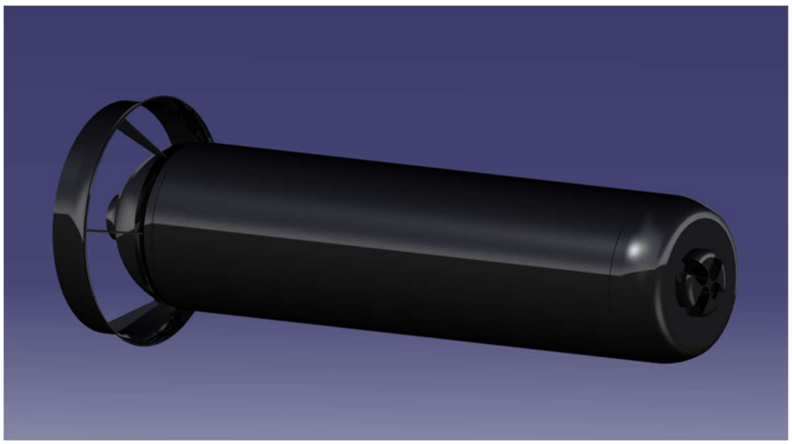


Fig. 3.1.3. Render approssimativo

## 3.2 Progettazione sistema di aggancio DS / Scooter

Verranno utilizzati due supporti “All-Purpouse” forniti da Suex, dotati di una cinghia e una molla, nativamente usati come supporto per action cam, con l’obiettivo di agganciare la DS allo scooter.

**Fig. 3.2.1. Supporto All-Purpouse**

Si è disegnato prima il supporto, come è possibile vedere in figura, rilevandone le misure col calibro, e cercando di essere il più precisi possibile. Tale componente è stato poi inserito in un file .CATProduct, che conterrà scooter, supporti e DS. In questa maniera, prendendo le misure dei fori già presenti sul DS e sfruttando quelli di default dei supporti, si è potuto realizzare un aggancio di questo genere, molto preciso:

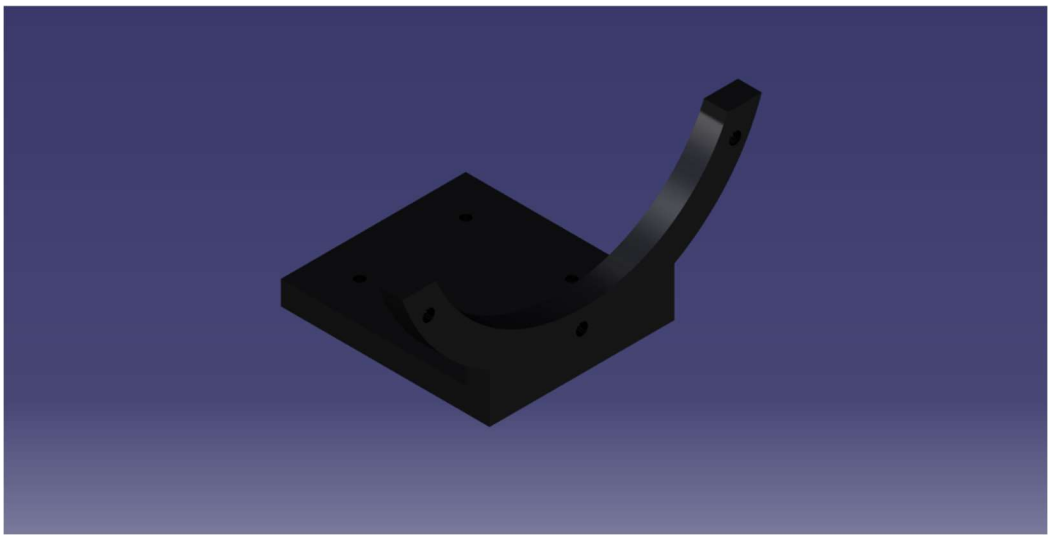


**Fig. 3.2.2. Aggancio DS / Suex XJT**

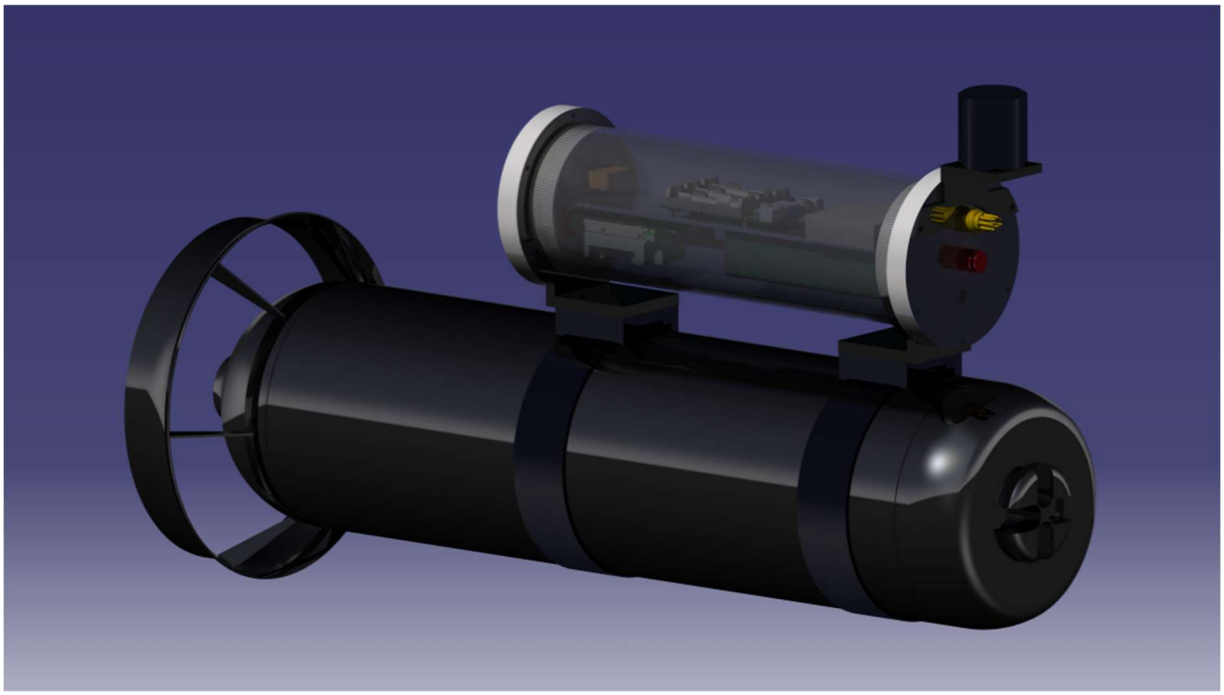


**Fig. 3.2.3. CatProduct del totale**

Una volta verificato che il pezzo era perfettamente adattabile ai supporti e alla DS, si è potuto proseguire e stampare due di questi pezzi, in 3D, con la stampante del laboratorio. Il risultato è stato il seguente:

**Fig. 3.2.4. Agganci stampati in 3D**

Per poi assemblare il tutto nella realtà, con questo risultato:

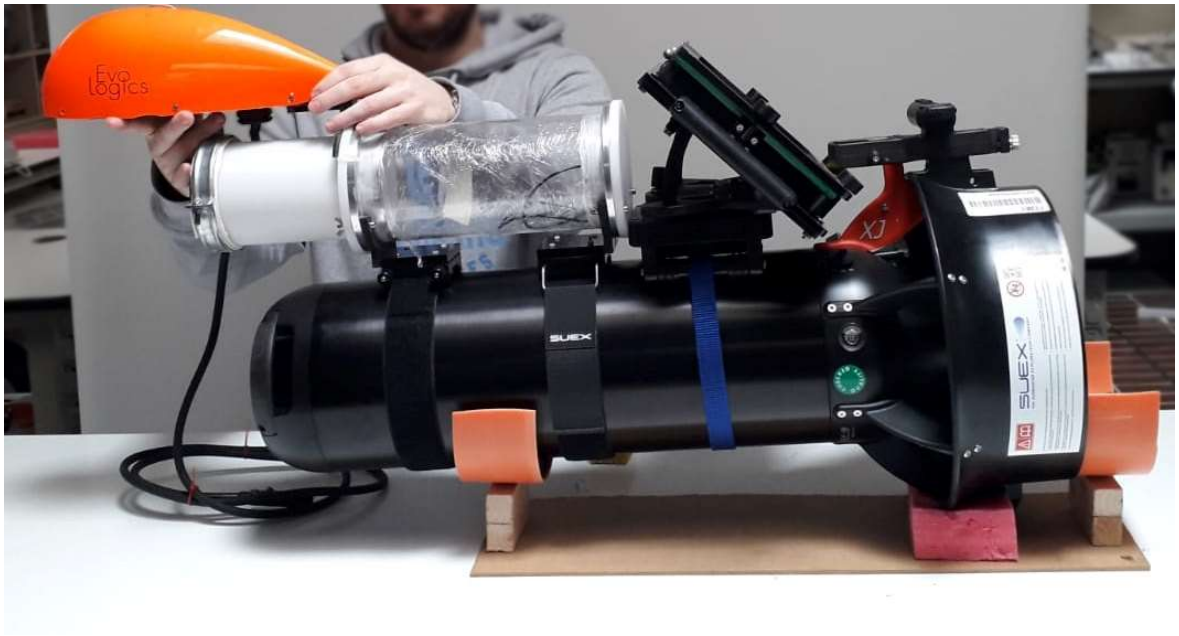



**Fig. 3.2.5. Risultato finale**

# 4. Progettazione Meccanica Case Diver Mainboard

## 4.1 Componenti

Si ripercorreranno ora i componenti che dovranno essere presenti all'interno del case:

- **MS5803-30BA**
- **Bar 30 High-Resolution 300m (MS5837-30BA)**
- **Raspberry Pi Zero W**
- **Li-Ion Battery HAT per Raspberry Pi Zero W**
- **Bridge WiFi/Bluetooth xDrip**

### 4.1.1. Raspberry Pi Zero W

La motherboard scelta per il sistema sensoristico del subacqueo è la Raspberry Pi Zero W a cui sono collegati, attraverso Bluetooth e protocollo i2c, i sensori di monitoraggio del subacqueo e i profondimetri, e la comunicazione con lo scooter DPV avviene attraverso un Bridge WiFi-Ethernet.

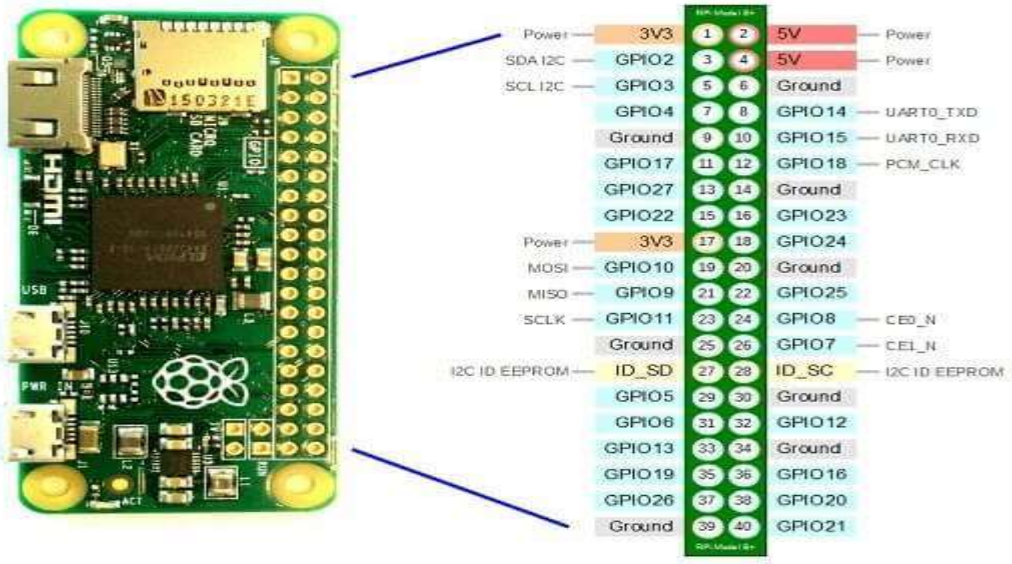



**Fig. 4.1.1.1. Raspberry Pi Zero pinout**

La Raspberry Pi Zero W (wireless) è un single-board computer, rilasciato nel 28 febbraio 2017, scelto perché ha connettività wireless LAN e Bluetooth integrati. La scheda considerata è la più piccola di tutta la serie Raspberry Pi, solo 65 x 31 x 5 mm ed è per questo comoda per essere inserita nel contenitore impermeabile che è parte del sistema integrato per il subacqueo.

**Specifiche tecniche:**

- 1GHz, single-core CPU
- 512MB RAM
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector
- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Per mantenere la Pi Zero più piccola e semplice possibile, lo spazio per la connessione GPIO viene lasciato vuoto dalla casa produttrice ma in questo caso si è deciso di saldare un connettore maschio 2x20.

La componentistica si basa su un *system-on-a-chip* di fabbricazione Broadcom BCM2835, cioè ha un sistema integrato che in un solo chip contiene processore centrale, un chipset ed eventualmente altri controller. La CPU in particolare è un 32-bit single-core ARM1176JZF-S ad 1 GHz, processore single-core con le prestazioni più alte tra i processori della famiglia Classic Arm. Introduce inoltre la tecnologia *Arm TrustZone* per permettere un'esecuzione sicura di software, attraverso un'isolazione forzata a livello hardware, costruita nella CPU. Il processore esegue

simultaneamente un sistema operativo sicuro e un sistema operativo normale da un single-core e TrustZone decide in quale dei due può far eseguire una libreria software, decidendo se è sicura o no. Creando un sottosistema di sicurezza, le risorse sono protette da attacchi software e dai comuni attacchi hardware.

Per quanto riguarda il sistema operativo, si utilizzerà Raspbian basato su Debian GNU/Linux e ottimizzato per la Raspberry Pi. Esso non è affiliato con la Raspberry Pi Foundation ma è stato creato da un team di developers interessati al single-board computer. Debian GNU/Linux è così denominata perché utilizza al suo interno programmi di utilità provenienti dal sistema operati GNU e utilizzando Linux come kernel. Nella versione "Raspbian Buster with Desktop" si ha già integrato Python 3.X.X dove la versione viene aggiornata ad ogni nuova release del sistema operativo in questione. Sarà quindi possibile eseguire gli script in Python necessari proprio su questa scheda, in questo caso specifico, la misurazione della frequenza di respiro.

### 4.1.2 Li-Ion Battery HAT per Raspberry Pi Zero W

**Fig. 4.1.2.1. Li-Ion Battery Hat**

Si tratta semplicemente di un power bank, prodotto dalla Waveshare che andrà ad alimentare la Raspberry tramite i pin GPIO, a 5V. Monta una singola batteria al litio da 3.7 V 14500 (4.2 V quando è totalmente carica).

### 4.1.3 xDrip Wifi

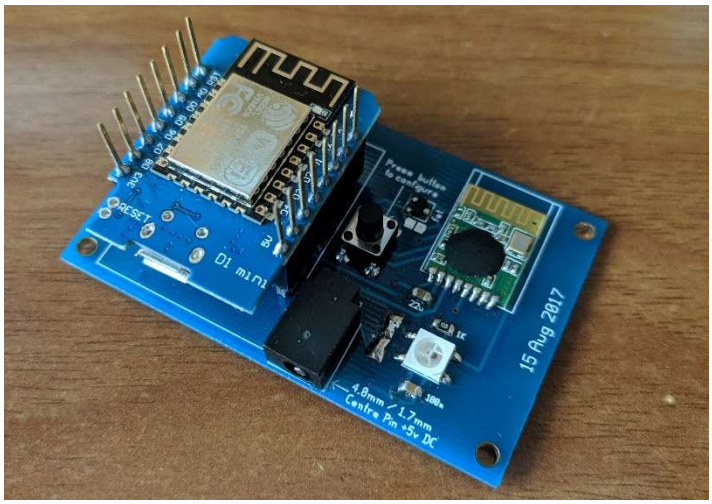



**Fig. 4.1.3.1. xDrip Wifi module**

Il modulo xDrip WiFi è un bridge Bluetooth, si comporta un ricevitore che permette la lettura dei dati inviati da un Glucose sensor in RF per renderli leggibili alla Raspberry tramite Bluetooth.

### 4.1.4 Primo sensore: MS5803-30BA

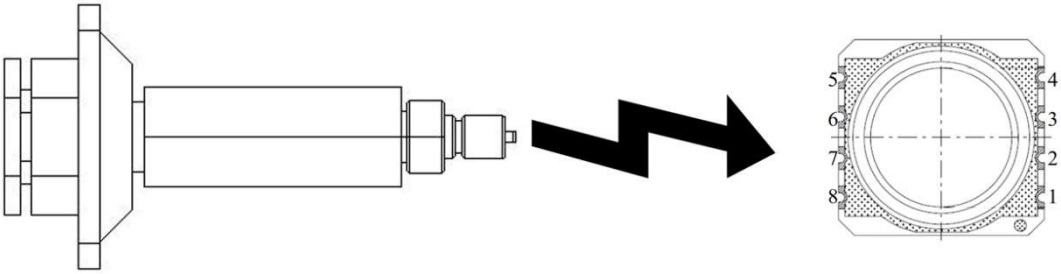



**Fig. 4.1.4.1. MS5803-30BA**

MS5803-30BA di TE Connectivity è un sensore di pressione ad alta risoluzione di nuova generazione con interfaccia bus I2C e SPI. È ottimizzato per sistemi di misurazione della profondità, con una risoluzione di profondità dell'acqua di 2,5 cm e inferiore. Il modulo sensore include un sensore di

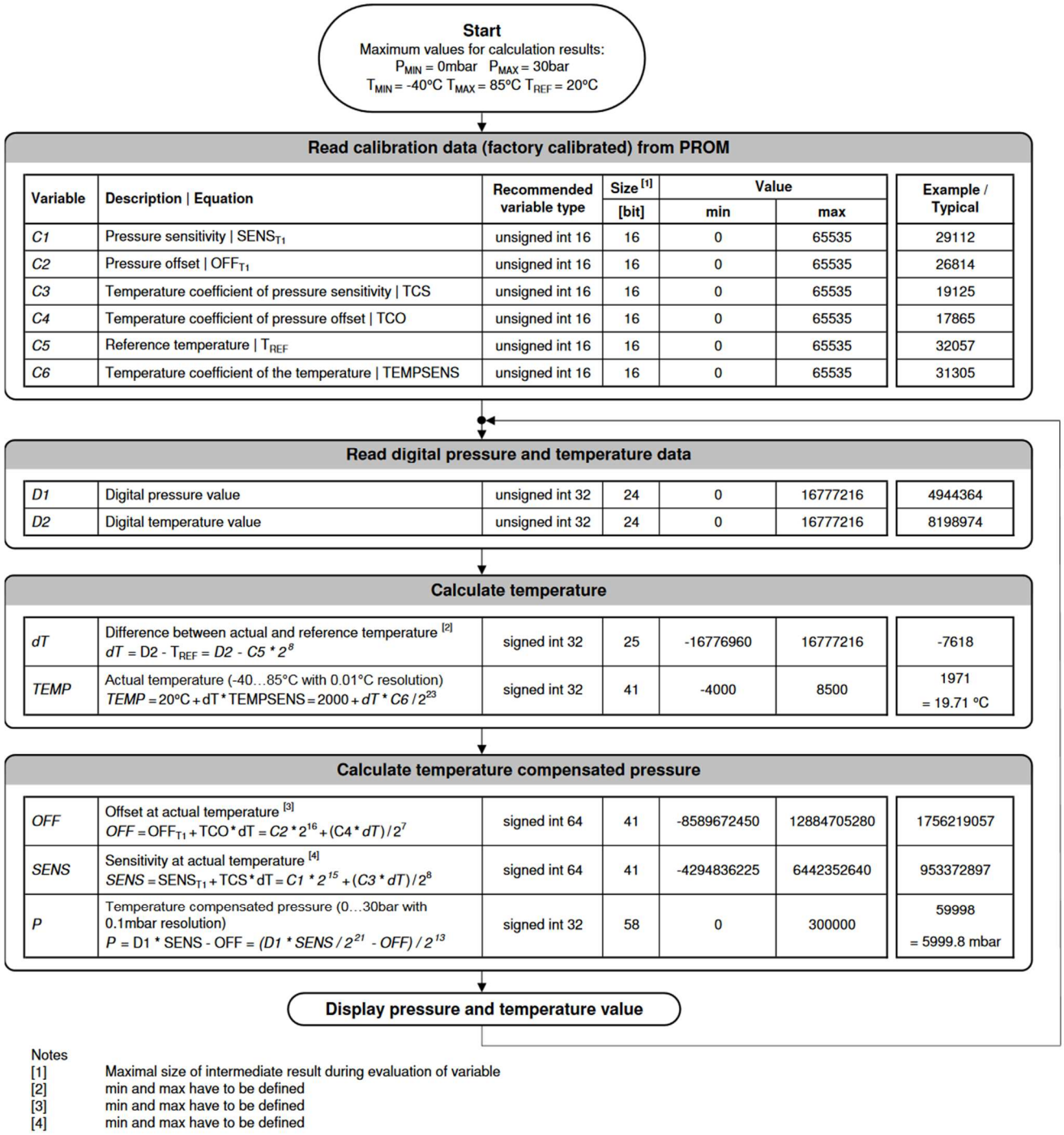
**Fig. 4.1.4.2: Foto del sensore**

pressione ad alta linearità e un ADC ΔΣ a 24 bit a potenza ultrabassa con coefficienti interni calibrati in fabbrica. Offre valori digitali precisi di temperatura e pressione a 24 bit e modalità di funzionamento diverse che consentono all'utente di ottimizzare la velocità di conversione e il consumo di corrente. MS5803-30BA può essere interfacciato con qualsiasi microcontrollore. Il protocollo di comunicazione è semplice, senza la necessità di programmare i registri interni nel dispositivo. Il gel di protezione e il tappo in acciaio inox antimagnetico rendono il modulo resistente all'acqua. Questa nuova generazione di sensori si basa sulla tecnologia MEMS. Il principio di rilevamento utilizzato risulta in un'isteresi molto bassa e un'alta stabilità del segnale di pressione e temperatura. [4]

L'MS5803-30BA è costituito da un sensore piezo-resistivo. La sua funzione principale è la conversione del voltaggio non compensato che il sensore di pressione piezoresistivo dà in output un valore a 24-bit digitale, oltre a fornire un valore digitale a 24 bit per la temperatura del sensore.

Come interfaccia seriale, permette una connessione con l'I2C e il SPI, la scelta avviene tramite il PinSelect PS il cui valore High corrisponde alla modalità I2C ed il valore Low corrisponde alla modalità SPI. Il suo funzionamento è a 3V, ma supporta da 1.8V a 3.6V.

Nella figura sottostante è possibile osservare il Flow Chart per la lettura della pressione e della temperatura e per la loro compensazione software.




| Variable | Description \| Equation | Recommended variable type | Size [1] [bit] | Value min | Value max | Example / Typical |
|---|---|---|---|---|---|---|
| *C1* | Pressure sensitivity \| $SENS_{T1}$ | unsigned int 16 | 16 | 0 | 65535 | 29112 |
| *C2* | Pressure offset \| $OFF_{T1}$ | unsigned int 16 | 16 | 0 | 65535 | 26814 |
| *C3* | Temperature coefficient of pressure sensitivity \| TCS | unsigned int 16 | 16 | 0 | 65535 | 19125 |
| *C4* | Temperature coefficient of pressure offset \| TCO | unsigned int 16 | 16 | 0 | 65535 | 17865 |
| *C5* | Reference temperature \| $T_{REF}$ | unsigned int 16 | 16 | 0 | 65535 | 32057 |
| *C6* | Temperature coefficient of the temperature \| TEMPSENS | unsigned int 16 | 16 | 0 | 65535 | 31305 |
| *D1* | Digital pressure value | unsigned int 32 | 24 | 0 | 16777216 | 4944364 |
| *D2* | Digital temperature value | unsigned int 32 | 24 | 0 | 16777216 | 8198974 |
| *dT* | Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$ | signed int 32 | 25 | -16776960 | 16777216 | -7618 |
| *TEMP* | Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$ | signed int 32 | 41 | -4000 | 8500 | 1971 = 19.71 °C |
| *OFF* | Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{16} + (C4 * dT) / 2^7$ | signed int 64 | 41 | -8589672450 | 12884705280 | 1756219057 |
| *SENS* | Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{15} + (C3 * dT) / 2^8$ | signed int 64 | 41 | -4294836225 | 6442352640 | 953372897 |
| *P* | Temperature compensated pressure (0…30bar with 0.1mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{13}$ | signed int 32 | 58 | 0 | 300000 | 59998 = 5999.8 mbar |

Notes
[1] Maximal size of intermediate result during evaluation of variable
[2] min and max have to be defined
[3] min and max have to be defined
[4] min and max have to be defined

**Fig. 4.1.4.3. Flow Chart MS5803-30BA, che esprime il funzionamento dell'algoritmo per la lettura dei dati**

Come possiamo vedere nella figura 4.1.4.1, il sensore è protetto da un involucro di ottone, il quale verrà collegato ad una frusta Gav per permettere al profondimetro di misurare la pressione dell'erogatore di primo stadio, perché lo scopo del lavoro sarà quello di utilizzarlo per rilevare questa pressione: in seguito, si analizzeranno i motivi di questa scelta.

## 4.1.5 Secondo sensore: Bar 30 High-Resolution 300m (MS5837-30BA)

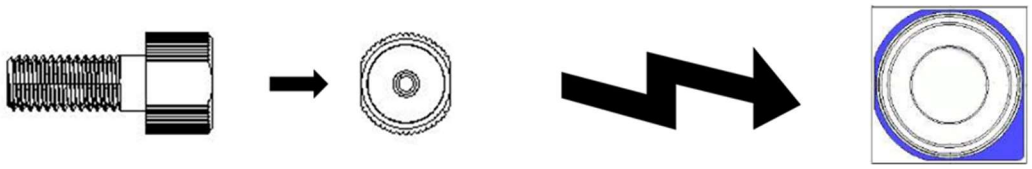


**Fig. 4.1.5.1. MS5837-30BA**

Il Bar 30 High-Resolution 300m è un sensore di alta pressione con un'alta risoluzione, può misurare pressioni fino a 30 bar (300 metri di profondità) con una risoluzione pari a 0.2 mbar; è facilmente installabile tramite il contenitore a vite impermeabile che ricopre il sensore su diversi tipi di ROV. Il sensore interno è un MS5837-30BA della Measurement Specialties, la cui organizzazione principale è TE Connectivity, che comunica tramite l'I2C. Il suo funzionamento è ad un voltaggio di 3.3V ma accetta fino a 5V in input.

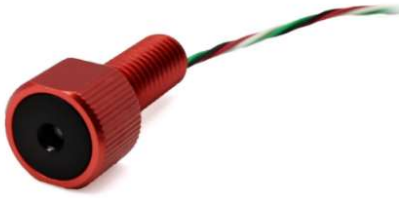


**Fig. 4.1.5.2. Bar 30 (in nero)**

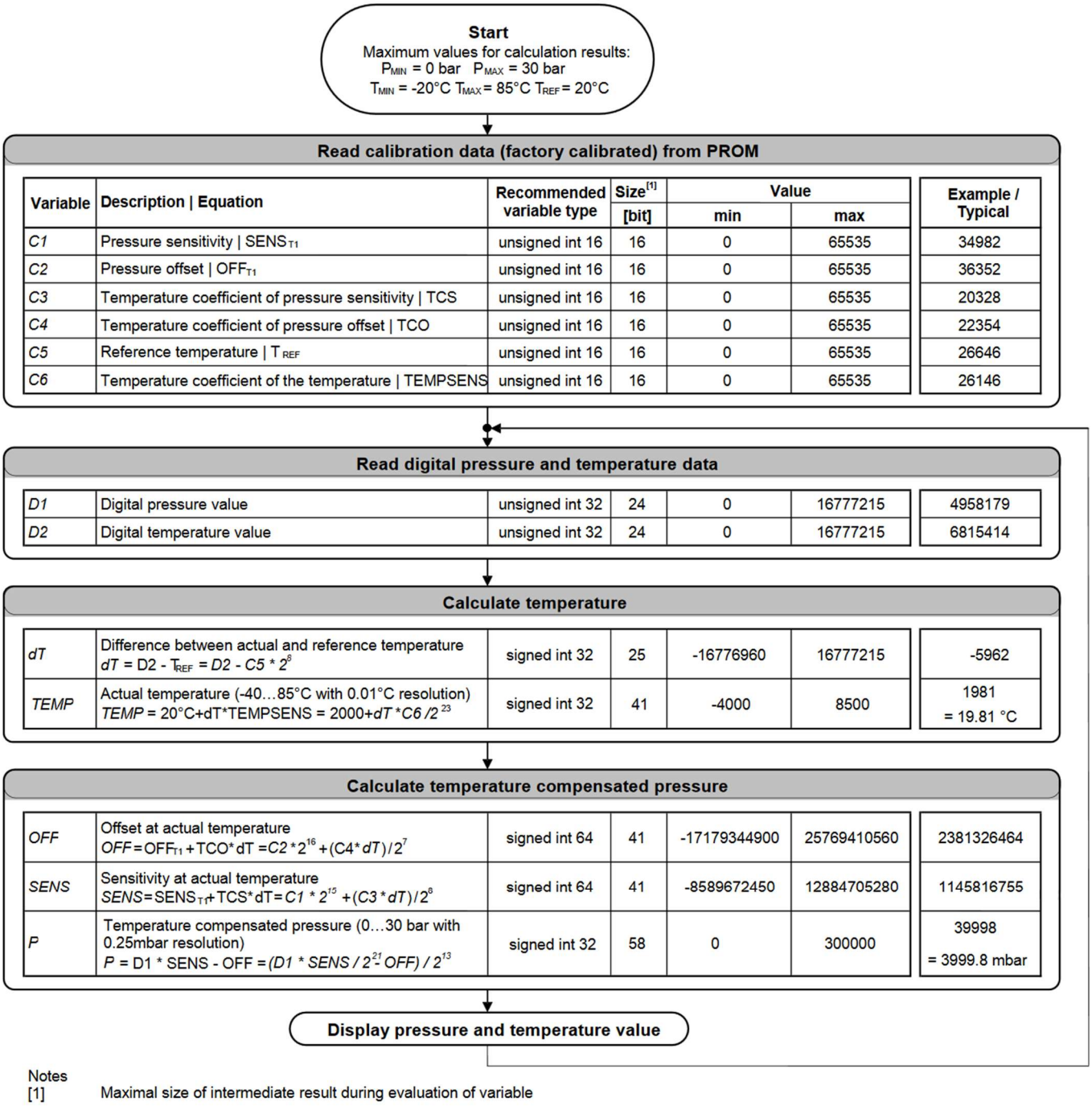
Ha come standard il connettore a 4-pin DF13 compatibile con la maggior parte delle flight controller boards come la PixHawk e la OpenPilot ma può essere facilmente utilizzato adattato a sistemi Raspberry Pi o Arduino.

**Fig. 4.1.5.3. Foto del sensore**

Simile al sensore già presentato, l'MS5837-30BA è costituito da un sensore piezo-resistivo. La sua funzione

principale è la conversione del voltaggio non compensato che il sensore di pressione piezoresistivo dà in output un valore a 24-bit digitale, oltre a fornire un valore digitale a 24 bit per la temperatura del sensore. [5]

Nella figura sottostante è possibile osservare il Flow Chart per la lettura della pressione e della temperatura e per la loro compensazione software.




| Variable | Description \| Equation | Recommended variable type | Size[1] [bit] | Value min | Value max | Example / Typical |
|---|---|---|---|---|---|---|
| *C1* | Pressure sensitivity \| $SENS_{T1}$ | unsigned int 16 | 16 | 0 | 65535 | 34982 |
| *C2* | Pressure offset \| $OFF_{T1}$ | unsigned int 16 | 16 | 0 | 65535 | 36352 |
| *C3* | Temperature coefficient of pressure sensitivity \| TCS | unsigned int 16 | 16 | 0 | 65535 | 20328 |
| *C4* | Temperature coefficient of pressure offset \| TCO | unsigned int 16 | 16 | 0 | 65535 | 22354 |
| *C5* | Reference temperature \| $T_{REF}$ | unsigned int 16 | 16 | 0 | 65535 | 26646 |
| *C6* | Temperature coefficient of the temperature \| TEMPSENS | unsigned int 16 | 16 | 0 | 65535 | 26146 |
| *D1* | Digital pressure value | unsigned int 32 | 24 | 0 | 16777215 | 4958179 |
| *D2* | Digital temperature value | unsigned int 32 | 24 | 0 | 16777215 | 6815414 |
| *dT* | Difference between actual and reference temperature $dT = D2 - T_{REF} = D2 - C5 * 2^8$ | signed int 32 | 25 | -16776960 | 16777215 | -5962 |
| *TEMP* | Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$ | signed int 32 | 41 | -4000 | 8500 | 1981 = 19.81 °C |
| *OFF* | Offset at actual temperature $OFF = OFF_{T1} + TCO * dT = C2 * 2^{16} + (C4 * dT) / 2^7$ | signed int 64 | 41 | -17179344900 | 25769410560 | 2381326464 |
| *SENS* | Sensitivity at actual temperature $SENS = SENS_{T1} + TCS * dT = C1 * 2^{15} + (C3 * dT) / 2^8$ | signed int 64 | 41 | -8589672450 | 12884705280 | 1145816755 |
| *P* | Temperature compensated pressure (0…30 bar with 0.25mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{13}$ | signed int 32 | 58 | 0 | 300000 | 39998 = 3999.8 mbar |

Notes
[1] Maximal size of intermediate result during evaluation of variable

**Fig. 4.1.5.4. Flow Chart MS5837-30BA, che esprime il funzionamento dell'algoritmo per la lettura dei dati**

## 4.2 Progettazione

In questa fase, l'obiettivo è sistemare in maniera ordinata i componenti e cercare eventuali punti di fissaggio per gli stessi al fine di dargli stabilità all'interno del case durante le immersioni. In particolar modo, è necessario trovare una giusta posizione per attaccare al case stesso, tramite delle viti la cui sede non è passante e viene filettata, il profondimetro in ottone, e fare in modo che lo spessore residuo, quindi dalla parte finale della sede all'interno del case, resti entro un certo range: tale range ovviamente varia in base al materiale, per permettergli una buona resistenza all'aumentare della pressione. Nel nostro caso il box in questione è realizzato in resina acetalica.

La resistenza del box deve essere garantita fino a 50 metri di profondità, anche se attualmente si potrà operare fino a 30 metri, in quanto i sensori adottati supportano al massimo questo valore. Per garantire l'impermeabilità, il case in questione è dotato di un tappo con un O-ring, una guarnizione piuttosto comune, usata nei progetti per via dei loro costi limitati e della loro resistenza a decine di megapascal di pressione.

La prima cosa da fare, anche in questo caso, è stata procedere disegnando dapprima il contenitore e il relativo tappo, rilevandone le misure con il calibro, e successivamente misurando anche gli altri componenti, tranne la Raspberry, per la quale sono presenti numerosi modelli già pronti in rete, già molto precisi e completi.

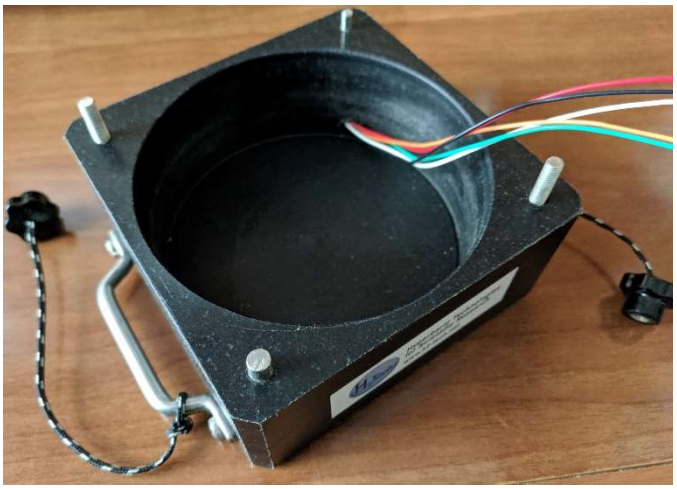


**Fig. 4.2.1. case diver aperto**

Successivamente si valuterà la possibile disposizione dei componenti interni nella realtà e una volta trovata quella più comoda e di meno ingombro, si procederà facendo ciò anche sul file del .CATProduct relativo. Si andrà poi a gestire lo spazio avanzato lungo i quattro lati interni, in maniera tale da riuscire a posizionare i due profondimetri dall'esterno, senza che vi siano ingombri con gli altri componenti, e che ci sia spazio per collegarli e

alimentarli, e simulare il fissaggio con le viti; si controllerà poi lo spessore in residuo per verificare che sia adatto a sopportare una pressione, dovuta alla profondità, elevata.

Si riporta ora il risultato finale ottenuto su CATIA tenendo conto di queste premesse e considerazioni appena fatte:

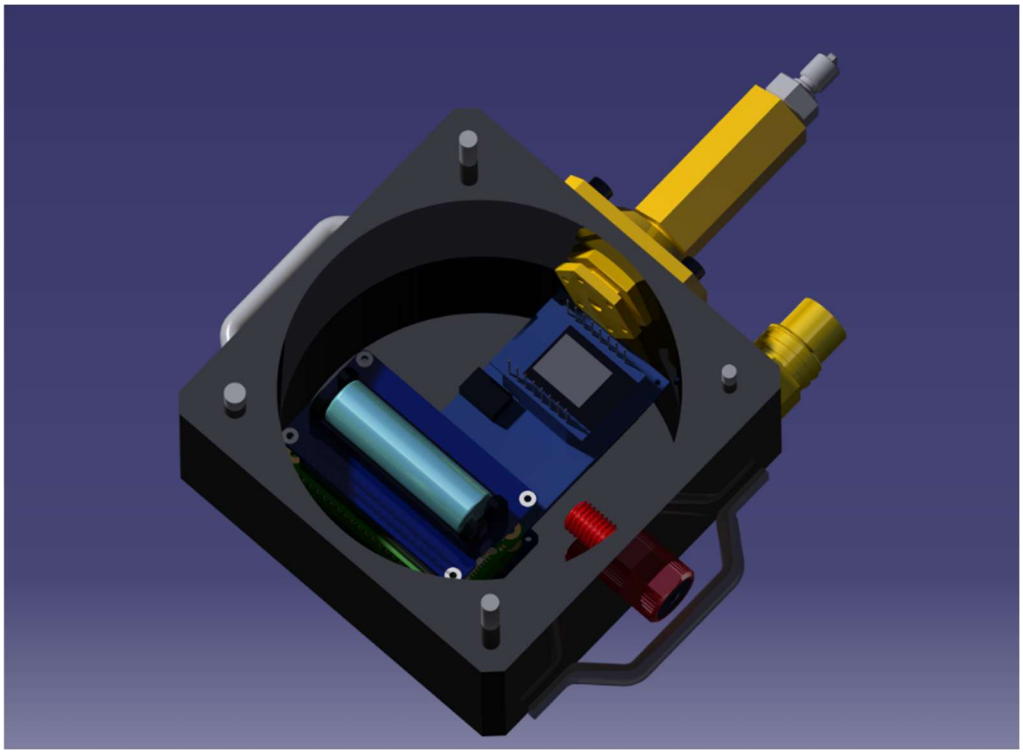


**Fig. 4.2.2. posizionamento dei componenti nel case**

Adesso si passerà ad evidenziare due dettagli di questa configurazione, una relativa al profondimetro nel guscio di ottone, e una relativa a un possibile fissaggio dei componenti come preannunciato qualche riga fa.

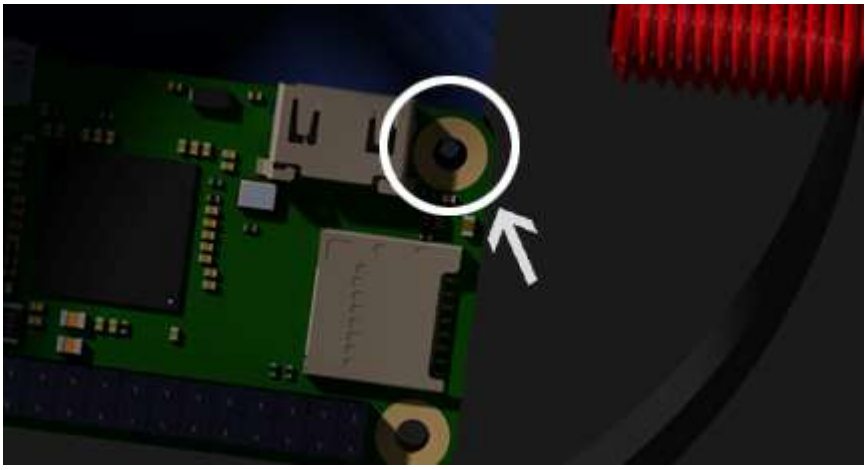


**Fig. 4.2.3. Fori concentrici (dettaglio)**

Con quest'ordine dei componenti, la Raspberry, potrebbe essere fissata, assieme al Bridge xDrip alla base del case, attraverso due fori concentrici, come si vede nella figura, con una vite simile a quelle adottate per fissare il profondimetro, altrimenti si potrebbero avere problemi all'aumentare della pressione.

In maniera analoga sono stati aggiunti degli spessori sulla simulazione di fissaggio del profondimetro ricoperto di ottone: infatti, rilevandone le misure al pc è emerso che lo spessore residuo dopo aver inserito le viti, era di circa 2.5mm. Di conseguenza verranno adottati degli spessori da 0.5mm, come si vede in foto, in grigio, dietro le viti, per rendere il case più spesso e quindi dargli più resistenza.

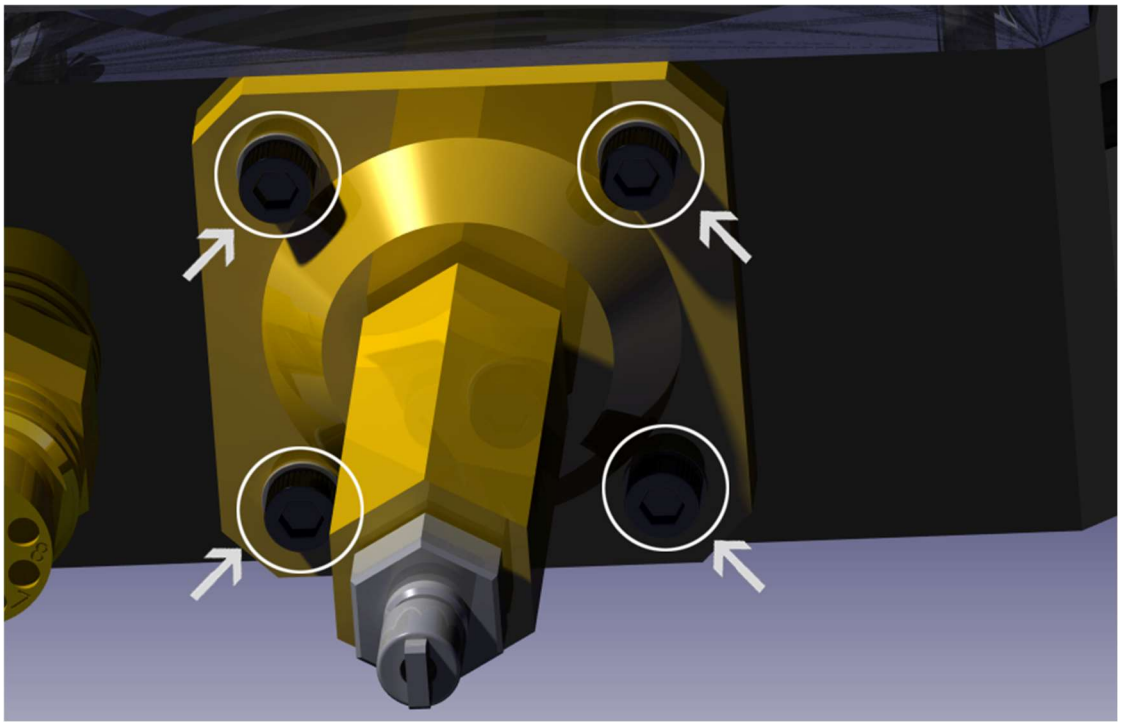


**Fig. 4.2.4. Spessori (dettaglio)**

## 4.3 Schema dei collegamenti all'interno del case

Per la rappresentazione grafica dei collegamenti sono stati usati i render preparati durante la progettazione. Si illustrano quindi nell'immagine sottostante le modalità di alimentazione tra i vari componenti e i collegamenti tra di essi. Si noti che sono stati evidenziati in nero i componenti che saranno fisicamente presenti dentro la scatola (o nel caso dei profondimetri, appena fuori). In nero, non evidenziato, invece si possono notare i sensori, o i collegamenti senza fili.

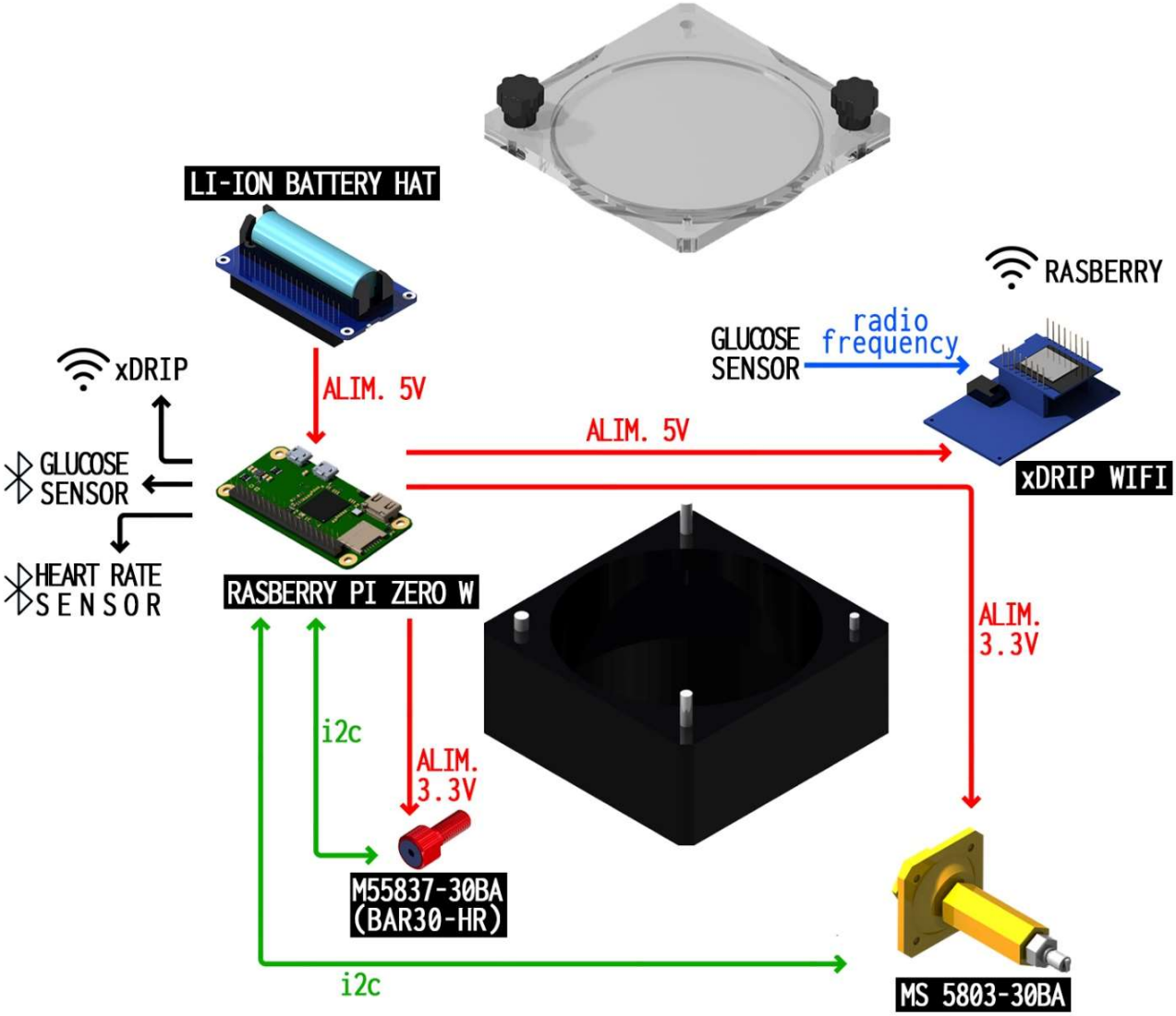



**Fig. 4.2.5. Schema dei collegamenti**

Come si può vedere, i due profondimetri sono alimentati a 3.3V in uscita dalla Raspberry, così come l'xDrip Wifi, che però lavora a 5V. A sua volta la Raspberry è alimentata a 5V da un battery pack. La comunicazione della Raspberry con il Glucose sensor e l'Heart rate sensor è via bluetooth, mentre nella comunicazione i2c la Raspberry poi farà da master per i due profondimetri (Bar30-HR e l'MS5803-30BA). Oltre a tutto questo la Raspberry è collegata tramite Wifi all'xDrip Wifi, un bridge Wifi-Bluetooth, che prende il segnale in radiofrequenza del Glucose sensor e lo emette in bluetooth per la Raspberry.

# 5. Breathing Monitor e risultati

## 5.1 Stato dell'arte

L'annegamento è la principale causa di morte nelle immersioni con autorespiratore subacqueo (SCUBA). [2] Un sub in immersione è esposto a numerosi rischi, e la maggior parte di questi derivano delle leggi fisiche a cui è sottoposto, che hanno effetti diversi rispetto a quelli che si avrebbero sulla terraferma, mentre i pericoli relativi a un'attrezzatura difettosa o usata in malo modo sono piuttosto rari. Questi avvenimenti, o meglio, quelli più frequenti, possono essere riassunti nelle patologie da decompressione o PDD**,** che possono essere di due tipi:

- la malattia da decompressione (MDD), che è causata da bolle di gas nei tessuti;
- l'embolia gassosa arteriosa (EGA), che è causata da bolle di gas nel circolo sanguigno.

Dato che ci si occuperà della gestione della frequenza di respiro di un sub, andremo a trattare brevemente la seconda tipologia, che corrisponde ad una sovradistensione polmonare acuta.

### 5.1.1 Sovradistensione polmonare (Barotrauma)

La sovradistensione polmonare è la rottura di alveoli polmonari causata da un eccessivo aumento di volume dei gas presenti. Questo può avvenire principalmente se il subacqueo trattiene il fiato durante una risalita, o a causa della presenza di muco negli alveoli che impediscono l'uscita del gas. Per comprenderne il motivo, ipotizziamo una risalita da 10 metri di profondità fino alla superficie. La pressione passa da 2 bar a 1 bar, e quindi l'aria nei polmoni essendo sottoposta a metà del valore della pressione rispetto a prima, raddoppia di volume per effetto della Legge di Boyle-Mariotte, per cui a temperatura costante, la pressione assoluta e il volume di un gas perfetto sono inversamente proporzionali.

L'aumento di volume oltre alla rottura degli alveoli provoca il passaggio d'aria nella cavità pleurica con conseguente collasso del polmone (pneumotorace) o più raramente nel mediastino (pneumomediastino) o, nell'eventualità peggiore, direttamente nel circolo arterioso polmonare (embolia gassosa arteriosa- EGA).

I sintomi si manifestano immediatamente con vertigini, dolore al torace, problemi alla vista e paralisi. Nel caso peggiore in cui l'aria entra nel circolo arterioso polmonare si avranno convulsioni con fuoriuscita di schiuma rossastra da bocca e naso, debolezza muscolare, paralisi, perdita di conoscenza e arresto respiratorio, fino ad arrivare alla morte.

Si andranno ora a considerare altri tipi di problemi, causati da una cattiva qualità dell'aria che respira il sub che di conseguenza possono portare a complicazioni durante un'immersione. Ad esempio, tratteremo brevemente l'avvelenamento da diossido di carbonio e quello da monossido di carbonio.

### 5.1.2 Avvelenamento da diossido di carbonio (anidride carbonica)

L'intossicazione da diossido di carbonio può essere causata da uno qualsiasi dei seguenti fattori:

- Inadeguato riflesso centrale della respirazione (ipoventilazione)
- Una muta stretta
- Sforzo fisico abnorme
- Malfunzionamento dell'erogatore
- Immersione profonda
- Contaminazione della fornitura di aria da gas esalati (come avviene in caso di anomalia del funzionamento del depuratore del diossido di carbonio in un sistema di fornitura di aria da rebreather)

L'ipoventilazione può incrementare i livelli ematici di diossido di carbonio e causare dispnea e sedazione. Le forme gravi da avvelenamento da diossido di carbonio possono

causare nausea, vomito, vertigini, cefalea, respiro accelerato, vampate, confusione, convulsioni e perdita di coscienza. [6]

Le forme lievi da avvelenamento di diossido di carbonio possono essere sospettate quando i subacquei lamentano cefalea da immersione o consumano poca aria.

L'intossicazione da diossido di carbonio si risolve abitualmente durante la risalita; pertanto, l'emogasanalisi dopo un'immersione, tipicamente non rileva alcun aumento dei livelli di diossido di carbonio. Il trattamento si basa su una risalita graduale e sull'interruzione dell'immersione o sulla correzione della causa precipitante.

### 5.1.3 Avvelenamento da monossido di carbonio

Il monossido di carbonio può penetrare nell'apparato di respirazione autonomo della scorta di un subacqueo se la bombola è "ricaricata" in un ambiente troppo vicino a scarichi di gas tossici o se l'olio lubrificante di un compressore difettoso diventa sufficientemente caldo da poter bruciare parzialmente (vampata), producendo monossido di carbonio. [6]

I sintomi comprendono nausea, cefalea, debolezza, goffaggine e alterazioni mentali. Avvelenamenti gravi da monossido di carbonio possono causare convulsioni, sincope o coma.

Per non appesantire la trattazione, non si discuteranno altre varie problematiche, di interesse minore per gli argomenti trattati in questa tesi. In ogni caso, un elenco approssimativo di essi è il seguente: problemi cardiaci, perdita di conoscenza, perdita del controllo motorio, narcosi da azoto, narcosi da ossigeno e altre malattie da decompressione.

Sono state fatte queste premesse, per sottolineare quanto sia importante la qualità del respiro che ha un sub durante l’immersione, in particolar modo la frequenza di respiro e la regolarità di esso.

Di solito l'approccio che viene utilizzato dai sub è quello di respirare in maniera diaframmatica, spingendo lo stomaco in fuori e lasciare espandere i polmoni. Tutto ciò deve essere effettuato molto lentamente, in maniera tale da far durare ogni ispirazione da 5 a 7 secondi e ogni espirazione non meno di 7 secondi, per un ciclo respiratorio di 12-15 secondi circa e di conseguenza un tasso respiratorio molto efficiente di 4-5 atti respiratori al minuto. [7]

## 5.2 Principio fisico funzionamento

In questa sezione si esporrà il principio fisico di funzionamento dell'algoritmo e le basi su cui poggia, seguendo l'articolo di C. Altepe, S. Murat Egi, T. Ozyigit, D. R. Sinoplu, Alessandro Marroni e Paola Pierleoni. [2] In seguito, si effettuerà una simulazione di immersione dando al codice dei valori di pressione per l'uno e l'altro sensore.

Il componente fondamentale di un'immersione è il regolatore, che permette la respirazione, abbassando la pressione del gas compresso contenuto nella bombola a una pressione respirabile per il sub. Questo regolatore si basa su due fasi:

- un primo stadio fa scendere la pressione dal serbatoio generalmente classificata a 200 bar a un IP (Intermediate Pressure),
- il secondo stadio fa scendere l'IP alla pressione ambientale (PB).

Nei moderni erogatori, l'IP è impostato su una pressione assoluta di pressione ambientale + 9,6 bar mentre il sub non respira. Quando il sub inizia a respirare, si verifica un'improvvisa diminuzione del valore IP che ritorna gradualmente al suo valore nominale (PB + 9,6 bar). Gli autori hanno ipotizzato che il rilevamento di questa improvvisa diminuzione, innescata dall'inalazione possa essere utilizzato per rilevare la frequenza respiratoria e per identificare l'assenza di respirazione che porterà ad annegamento o anomalie come l'iperventilazione o il salto respiratorio.

Viene ora riportato il grafico relativo al segnale di pressione intermedia nel momento della respirazione di un sub, nel quale è possibile vedere cosa accade a seguito della respirazione, in accordo con quanto detto in precedenza:

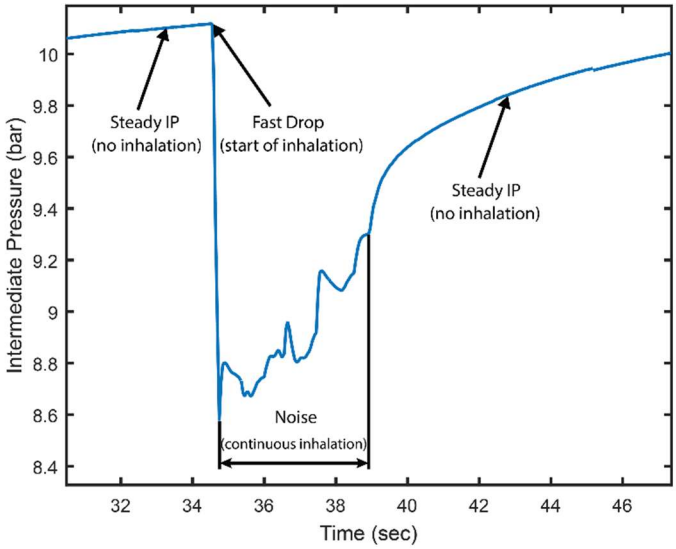


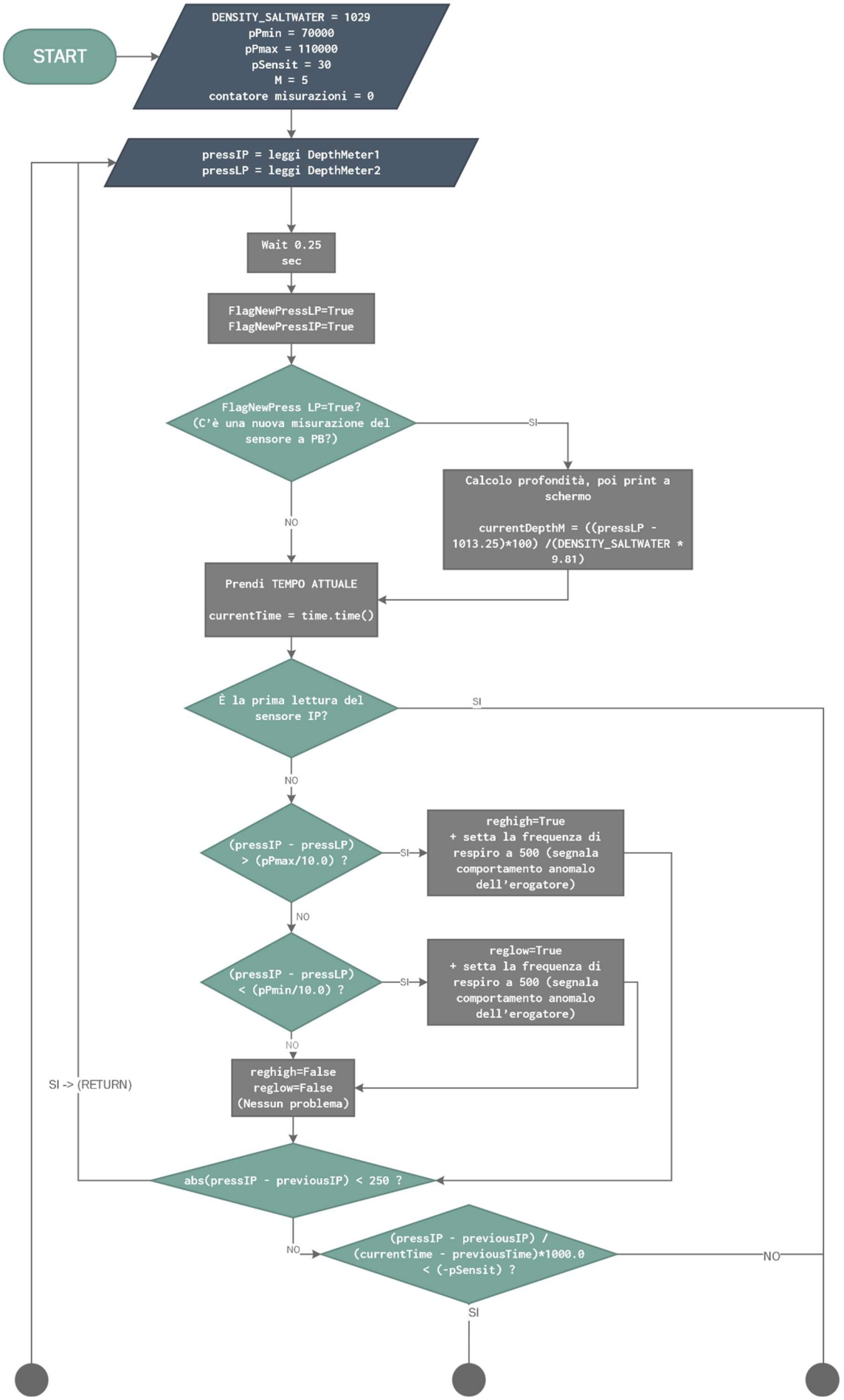

**Fig. 5.2.1. Evento di respirazione**

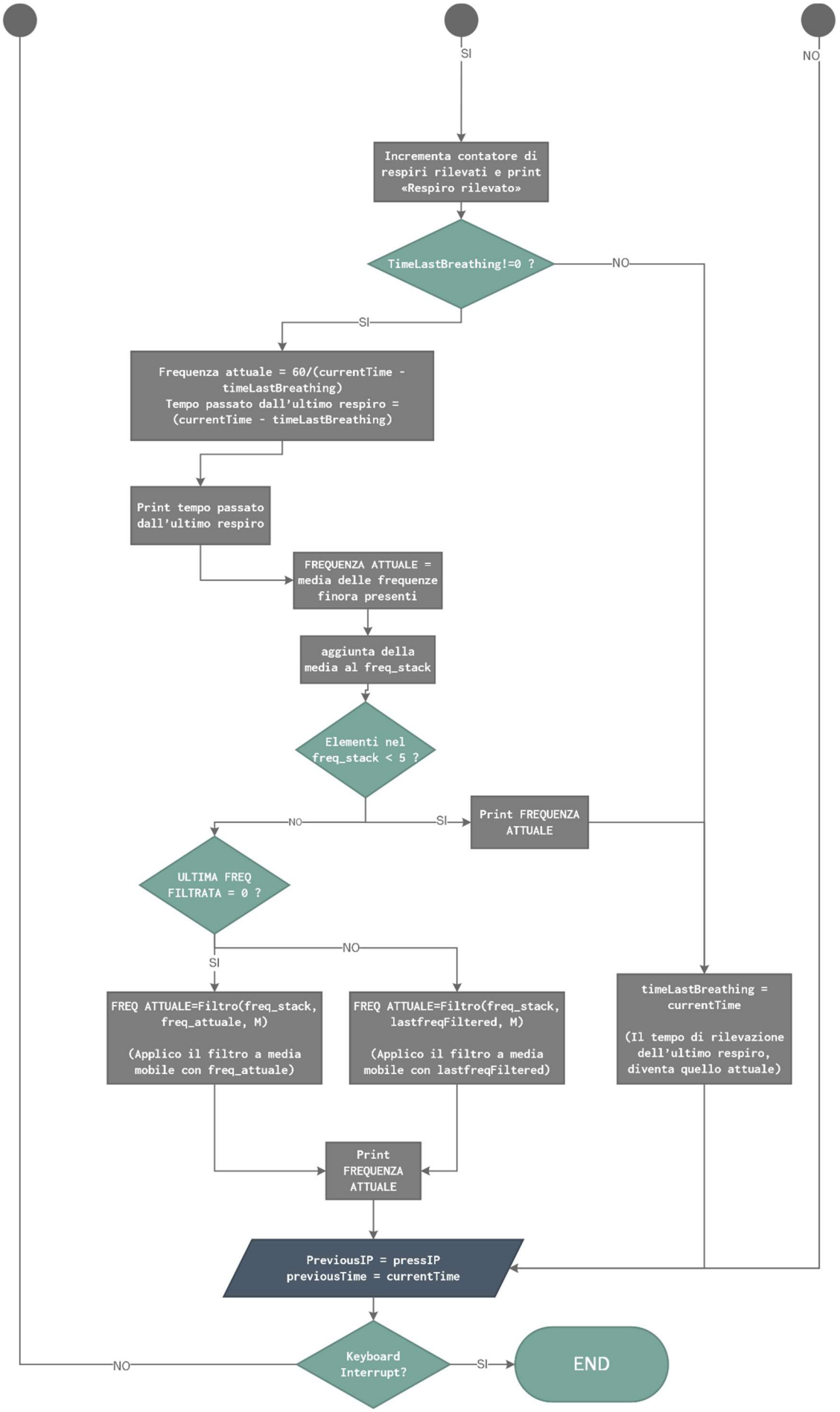
L'algoritmo è stato progettato in maniera efficiente, in modo tale da poter girare su un dispositivo elettronico costruito attorno a un microcontrollore (MSP430F5529, Texas Instruments Incorporated) e due sensori MS5837, oltre a richiedere soltanto 800 byte di memoria RAM con un ritardo massimo di 5,2 sec.

Queste problematiche non si presenteranno però nel caso trattato in questa tesi, poiché la mainboard su cui andrà a girare il codice è una Raspberry Pi Zero, che è molto più potente del microcontrollore usato e possiede un quantitativo di RAM di gran lunga maggiore.

## 5.3 Diagramma di flusso sull'algoritmo del software

In questa sezione si potrà vedere un Flow Chart sul funzionamento dell'algoritmo per la rilevazione.


START
DENSITY_SALTWATER = 1029
pPmin = 70000
pPmax = 110000
pSensit = 30
M = 5
contatore misurazioni = 0
pressIP = leggi DepthMeter1
pressLP = leggi DepthMeter2
Wait 0.25 sec
FlagNewPressLP=True
FlagNewPressIP=True
FlagNewPress LP=True? (C'è una nuova misurazione del sensore a PB?)
SI
Calcolo profondità, poi print a schermo
currentDepthM = ((pressLP - 1013.25)*100) /(DENSITY_SALTWATER * 9.81)
NO
Prendi TEMPO ATTUALE
currentTime = time.time()
È la prima lettura del sensore IP?
SI
NO
(pressIP - pressLP) > (pPmax/10.0) ?
SI
reghigh=True
+ setta la frequenza di respiro a 500 (segnala comportamento anomalo dell'erogatore)
NO
(pressIP - pressLP) < (pPmin/10.0) ?
SI
reglow=True
+ setta la frequenza di respiro a 500 (segnala comportamento anomalo dell'erogatore)
NO
reghigh=False
reglow=False
(Nessun problema)
SI -> (RETURN)
abs(pressIP - previousIP) < 250 ?
NO
(pressIP - previousIP) / (currentTime - previousTime)*1000.0 < (-pSensit) ?
NO
SI


SI
NO
Incrementa contatore di respiri rilevati e print «Respiro rilevato»
TimeLastBreathing!=0 ?
NO
SI
Frequenza attuale = 60/(currentTime - timeLastBreathing)
Tempo passato dall'ultimo respiro = (currentTime - timeLastBreathing)
Print tempo passato dall'ultimo respiro
FREQUENZA ATTUALE = media delle frequenze finora presenti
aggiunta della media al freq_stack
Elementi nel freq_stack < 5 ?
NO
SI
Print FREQUENZA ATTUALE
ULTIMA FREQ FILTRATA = 0 ?
NO
SI
FREQ ATTUALE=Filtro(freq_stack, freq_attuale, M)
(Applico il filtro a media mobile con freq_attuale)
FREQ ATTUALE=Filtro(freq_stack, lastfreqFiltered, M)
(Applico il filtro a media mobile con lastfreqFiltered)
timeLastBreathing = currentTime
(Il tempo di rilevazione dell'ultimo respiro, diventa quello attuale)
Print FREQUENZA ATTUALE
PreviousIP = pressIP
previousTime = currentTime
Keyboard Interrupt?
NO
SI
END

## 5.4 Descrizione del codice

Per iniziare, si è realizzata una classe chiamata *Algorithm* contenente le funzioni che utilizzerà l'algoritmo, per sfruttare i vantaggi della programmazione ad oggetti, (Ereditarietà, Incapsulamento, Polimorfismo), per semplificarne la lettura, rendere il *main* più comprensibile, ma specialmente per rendere il codice ordinato per procedere poi con lo svolgimento della simulazione, dove nel *main* saranno gestiti molti dati.

Si noti la presenza dell'*__init__*, un metodo speciale in Python che ci permette di inizializzare le istanze a cui in questo caso si passeranno, come argomenti, quelle variabili o strutture che dovranno essere aggiornate ad ogni ciclo. In aggiunta a ciò passeremo il consueto *self* in Python, argomento che si riferisce all'istanza e che va a definire un metodo.

```
1  class Algorithm():
2      def __init__(self, firstReading, flagNewPressLP, flagNewPressIP, reghigh, reglow,
3      freq_stack, pressIP, pressLP, currentTime, previousTime, previousIP,
4      timeLastBreathing, lastfreqFiltered, freq_attuale, cont_mis):
5          self.firstReading=firstReading
6          self.flagNewPressLP=flagNewPressLP
7          self.flagNewPressIP=flagNewPressIP
8          self.freq_stack=freq_stack
9          self.pressIP=pressIP
10         self.pressLP=pressLP
11         self.currentTime=currentTime
12         self.previousTime=previousTime
13         self.previousIP=previousIP
14         self.timeLastBreathing=timeLastBreathing
15         self.lastfreqFiltered=lastfreqFiltered
16         self.freq_attuale=freq_attuale
17         self.cont_mis=cont_mis
```

Elenchiamo la funzione di questi flag, variabili e strutture dati:

*firstReading* – Va messo a True se è la prima lettura.

*flagNewPressLP* – va messo a True quando c'è una nuova misurazione del sensore a pressione ambientale.

*flagNewPressIP* – va messo a True quando c'è una nuova misurazione del sensore a pressione intermedia.

*reghigh* – va messo a True quando ci sono problemi con l'erogatore, con la pressione maggiore di una soglia.

*reglow* - va messo a True quando ci sono problemi con l'erogatore, con la pressione minore di una soglia.

*freq_stack* – pila all'interno della quale si inseriranno le frequenze rilevate.

*pressIP* – valore della lettura del profondimetro collegato al primo stadio.

*pressLP* – valore della lettura del profondimetro a pressione ambientale.

*currentTime* – istante attuale.

*previousTime* – istante precedente.

*previousIP* – valore della lettura precedente del profondimetro collegato al primo stadio.

*timeLastBreathing* – istante dell'ultimo respiro.

*lastfreqFiltered* – valore dell'ultima frequenza filtrate dal filtro a media mobile.

*freq_attuale* – valore della frequenza attuale.

*cont_mis* – contatore dei respiri, è stato utilizzato insieme al *print* per verificare il numero di respiri totali.

### 5.4.1 *routineDiveMonitor*

```
1   def routineDiveMonitor(self):
2     #routinePressureSensor()
3     Algorithm.routineCheckDepth(self)
4     Algorithm.routineBreathingMonitor(self)
```

Questa funzione, che si vedrà poi essere richiamata nel *main*, va a richiamare altre due funzioni. Si può notare che è commentata la funzione *routinePressureSensor()* poiché attualmente si sta testando il codice e verrà effettuata una simulazione; di conseguenza la funzione per la lettura dei profondimetri non è stata scritta, ma verranno forniti valori in ingresso per i due profondimetri da due liste, una per ogni profondimetro, come si vedrà in seguito.

### 5.4.2 *routineCheckDepth*

```
1    def routineCheckDepth(self):
2       DENSITY_SALTWATER = 1029 # kg/cm^3 convenience
3       if(self.flagNewPressLP):
4          self.flagNewPressLP = 0
5          currentDepthM = ((self.pressLP - 1013.25)*100) /(DENSITY_SALTWATER * 9.81)
6          print("profondità attuale: " ,round(currentDepthM,2), "metri")
```

Questa funzione è stata costruita per fare un check iniziale sul profondimetro a pressione ambientale: se questa condizione è verificata allora il programma calcola e stampa successivamente la profondità attuale.

### 5.4.3 *routineBreathingMonitor*

```
1  def routineBreathingMonitor(self):
2     M = 5   #Moving average filter
3     pPmin = 70000 #mbar x 10 (1 byte)
4     pPmax = 110000 #mbar x 10 (1 byte)
5     pSensit = 30    #mbar/sec (2 bytes)
6
7     self.currentTime=time.time()
8     if self.firstReading:
9        self.firstReading = 0
10       print("Prima lettura dei sensori")
11    else:
12       #Check 1st Stage Regulator Performance
13       if (self.pressIP - self.pressLP) > float(pPmax)/10.0:
14          print("Reg High!")
15          self.reghigh=True
16          self.freq_attuale=500
17       elif (self.pressIP - self.pressLP) < float(pPmin)/10.0:
18          print("Reg Low!")
19          self.reglow=True
20          self.freq_attuale=500
21       else:
22          self.reghigh=False
23          self.reglow=False
24     if abs(self.pressIP - self.previousIP)<250:
25        return
```

```
26      #Breathing detection
27      if (self.pressIP - self.previousIP)/(self.currentTime - self.previousTime)*1000.0 <
28        float(-pSensit): #Breathing detected:
29         self.cont_mis+=1
30         print("\nRespiro n.",self.cont_mis,"rilevato\n")
31         if self.timeLastBreathing!=0:
32            self.freq_attuale = 60/(self.currentTime - self.timeLastBreathing)
33            intervallo_ult_res = (self.currentTime -self.timeLastBreathing)
34            print("Tempo trascorso dall'ultimo respiro ", round(intervallo_ult_res,2),"secondi")
35            self.freq_attuale = gestiscifreq(self.freq_stack, self.freq_attuale)
36            self.freq_stack.append(self.freq_attuale)
37            if len(self.freq_stack)<5:
38               print("La freq di respiro è ", round(self.freq_attuale,2),"respiri al min")
39         else:
40            if self.lastfreqFiltered==0:
41               self.freq_attuale=Filtro(self.freq_stack, self.freq_attuale, M)
42            else:
43               self.freq_attuale=Filtro(self.freq_stack, self.lastfreqFiltered, M)
44            print("La freq di respiro filtrata è ",round(self.freq_attuale,2),"respiri al min")
45            self.lastfreqFiltered=self.freq_attuale
46         self.timeLastBreathing = self.currentTime
47      self.previousIP = self.pressIP
48      self.previousTime = self.currentTime
```

In questa funzione si andrà dapprima a dichiarare il numero di punti a cui applicare il filtro a media mobile e alcune valori ottenuti empiricamente che ci serviranno in seguito. (Riga 2-5).

Si va a salvare il tempo corrente nella variabile *currentTime,* e si controlla che non sia la prima lettura. Se questa condizione è verificata, si procede al check per il funzionamento dell'erogatore. Si noti che sono presenti due flag, che indicano se l'errore è "*low*" o "*high*", nel caso in cui si vorrà specificare che il valore di maggiorazione della pressione interna all'erogatore rispetto a quella ambientale è troppo basso o troppo alto. L'errore di malfunzionamento sarà dato dal valore di *freq_attuale* che viene settato a 500.

Nel caso in cui non ci siano problemi si va avanti: c'è un filtro, il cui scopo è quello di filtrare il segnale di pressione, settato per adesso a 250 mbar. Questo valore potrà essere corretto quando si farà la prova in acqua con veri segnali di pressione; tuttavia, al momento verrà considerato accettabile.

Successivamente si va a controllare se c'è stato un respiro, controllando il valore attuale di pressione dell'erogatore con quello precedente (Riga 26-27): se questo risultato restituisce True, allora si può procedere aumentando il contatore dei respiri e stampando a schermo:

**>Respiro n. *cont_mis* rilevato**

Se è il primo respiro, (check effettuato dalla variabile *TimeLastBreathing*) si va alla Riga 46 andando ad aggiornare le variabili per prepararsi al prossimo ciclo, altrimenti si prosegue, e si va a calcolare la frequenza attuale confrontando il *CurrentTime* col *TimeLastBreathing,* si stampa il tempo trascorso dall'ultimo respiro e poi si va a fare la media delle frequenze (se sono più di una) perché saranno trattate così le prime 5 frequenze, mentre dalla quinta in poi saranno trattate con un filtro a media mobile.

Successivamente si va ad aggiungere la frequenza restituita dalla funzione al *freq_stack,* pila di frequenze che una volta completa andremo a passare al filtro a media mobile. In seguito si farà un ulteriore controllo, sulla lunghezza di questa pila, se *len(freq_stack) < 5*, allora si stampa il valore della frequenza calcolato prima e si conclude, si aggiornano le variabili per il prossimo ciclo e si va avanti. In caso contrario, si controlla, se la variabile *lastfreqFiltered* è settata al valore iniziale, cioè zero: in tal caso, si passa alla funzione del filtro a media mobile *freq_attuale,* altrimenti si può passare alla funzione la variabile *lastfreqFiltered.* Fatto ciò, si vanno ad aggiornare le variabili per prepararsi al prossimo ciclo.

### 5.4.4 Altre funzioni

Il lavoro di applicare il filtro o svolgere la semplice media aritmetica tra le frequenze lo svolgono due semplici funzioni che vedremo di seguito, fuori dalla classe:

```
1  def Filtro(pila, f1, M):
2     if len(pila)==6:
3        pila.pop(0)
4     filtered = f1 + (1/M)*(pila[-1]-pila[0])
5     return filtered
6
7
```

```
def gestiscifreq(pila, freq):
   somma_freq=0
   pila.append(freq)
   for i in range(0,len(pila)):
      somma_freq = somma_freq + pila[i]
   pila.pop(-1)
   return somma_freq/(len(pila)+1)
```

La funzione *Filtro* va ad applicare il filtro a media mobile alla frequenza, quini prende in ingresso il *freq_stack,* la *lastfrequenceFiltered* e il numero di punti per il filtro e resituisce il valore filtrato della frequenza come *filtered.*

La funzione *gestiscifreq* va a fare la media delle frequenze, aggiungendo alla lista di medie la frequenza appena calcolata, ne calcola la somma, rimuove questa frequenza aggiunta col solo scopo di avere un valor medio, e restituisce la somma diviso il *numero di frequenze+1* perché l'ultima è stata appena rimossa. L'aggiunta della media delle frequenze restituite, sarà fatta nel *main,* appena dopo la chiamata della funzione.

## 5.5 Simulazione

Dopo la scrittura dell'algoritmo si è proceduto a fare una simulazione dell'algoritmo stesso. Innanzitutto sono state aggiunte due semplici funzioni nel *main* che generano rumore bianco, che restituiscono un valore tra -30 e +30 e un valore tra -40 e +40, chiamate rispettivamente *rand* e *rand2*. Il rumore più grande verrà sommato al segnale IP, che dovrebbe essere più rumoroso, mentre quello più piccolo al segnale PB, della pressione ambientale.

Il campionamento che si è scelto è di 0.25 secondi, quindi tralasciando i tempi di esecuzione dell'algoritmo, questo vorrà dire che 4 volte al secondo i sensori mandano una misurazione di pressione. I sensori sono sostituiti da due liste di elementi in questa

simulazione. Si è pensato di farla da 5 minuti, che con questo campionamento corrispondono a 1200 misurazioni, quindi 1200 elementi per ogni lista.

Il primo passo sarà costruire i le liste PB e IP, ma poiché ogni misurazione nell'erogatore, (IP) è di solito 9,6 bar superiore alla stessa misurazione a pressione ambientale quando il sub non sta respirando, si definirà in principio la lista dei segnali PB per poi aggiungere 9600 mbar e ottenere la lista dei segnali IP.

Nello specifico, dichiareremo dapprima:

```
1  PB = [2013] #mbar, inizializzazione lista PB (pressione ambiente)
```

che rappresenta una lista con un solo valore. Questo valore corrisponderà a 10 metri di profondità, essendo vicino a 2 bar e sapendo che in acqua salata la pressione aumenta di 1 bar ogni 10 metri di profondità: si è quindi ipotizzata una situazione di questo genere.

La lista verrà riempito con una serie di cicli, così pensati:

```
1  for i in range(0,119):
2      PB.append(PB[i]+2) #discesa
3  for i in range(119,359):
4      PB.append(PB[119]) #esplorazione stazionaria
5  for i in range(359,479):
6      PB.append(PB[i]+1) #discesa più lenta
7  for i in range(479,599):
8      PB.append(PB[i]-1) #risalita lenta
9  for i in range(599,839):
10     PB.append(PB[599]) #esplorazione stazionaria
11 for i in range(839,1079):
12     PB.append(PB[i]-1) #risalita lenta
13 for i in range(1079,1199):
14     PB.append(PB[1079]) #esplorazione stazionaria
```

Ad esempio prendendo il primo ciclo *for,* si va a incrementare di 2 mbar il valore appena precedente e lo si va ad aggiungere alla lista, quindi ottenendo un aumento di pressione pari a 120 moltiplicato per 2mbar quindi 240 mbar totali alla fine del ciclo. Lo stesso ragionamento è stato fatto per la risalita, risalita lenta o discesa più lenta, mentre

l'esplorazione stazionaria corrisponde a riempire la lista con valori, senza apportarne modifiche.

A questo punto, si può costruire la lista relativa ai segnali IP, senza respiri, quindi in questo modo:

```
1  #COSTRUZIONE LISTA IP, per adesso senza respirazioni, quindi 9600mbar sopra PB
2  IP=[]
3  for i in range(0,1200):
4     IP.append(9600+PB[i])
```

per poi poter inserire il rumore bianco:

```
1  #Aggiunta rumore bianco:
2  for i in range(0,1200):
3        IP[i]=IP[i]+rand2()
4        PB[i]=PB[i]+rand()
```

Si procede dunque all'inserimento manuale dei respiri, andando a sottrarre un quantitativo di mbar da un elemento della lista dei valori IP, mentre dall'indice appena seguente si andranno a sottrarre valori di mbar di volta in volta poco inferiori, rispettando quindi il grafico nella figura 5.2.1:

```
1  ##INSERIMENTO RESPIRI CON NOISE NELLA LISTA IP:
2
3  #Primo minuto: (inserisco 4 respiri)
4  IP[60]-=750  #L'indice corrisponde a (misurazione-1) poiché le misurazioni partono
5  da 1 diversamente dagli indici che partono da 0.
6  Quindi questo primo verrà rivelato a  misurazione n.61.
7
8  #Noise poco dopo primo respiro: (di assestamento, e che ne indica il continuo del
9  respiro)
10 IP[61]-=720
11 IP[62]-=740
12 IP[63]-=680
13 IP[64]-=620
14 IP[65]-=600
15 IP[66]-=400
16 IP[67]-=500
17 IP[68]-=100
18 #ne sono 8, a 0.25sec corrispondono a 2 secondi di respiro continuo (di rumore nel
19 grafico)
```

```
20
21  #secondo respiro, lo posiziono a 120, verrà rivelato a 121, per lo stesso
22  ragionamento di prima
23  IP[120]-=800
24  #Noise poco dopo secondo respiro:
25  IP[121]-=700
26  IP[122]-=750
27  IP[123]-=800
28  IP[124]-=800
29  IP[125]-=600
30  IP[126]-=680
31  IP[127]-=700
32  IP[128]-=720
33
34  #terzo respiro (a 160)
35  IP[160]-=700
36  #Noise poco dopo terzo respiro:
37  IP[161]-=700
38  IP[162]-=700
39  IP[163]-=800
40  IP[164]-=800
41  IP[165]-=700
42  IP[166]-=680
43  IP[167]-=700
44  IP[168]-=720
45
46  #quarto respiro (a 230)
47  IP[230]-=700
48  #Noise poco dopo quarto respiro:
49  IP[231]-=700
50  IP[232]-=650
51  IP[233]-=600
52  IP[234]-=700
53  IP[235]-=680
54  IP[236]-=600
55  IP[237]-=680
56  IP[238]-=400
57  ##FINE PRIMO MINUTO
```

Sono stati inseriti 4 respiri nel primo minuto.

Un ragionamento analogo sarà effettuato con gli altri 4 minuti restanti, inserendo:

- 3 respiri nel secondo minuto;
- 4 respiri nel terzo minuto;
- 2 respiri nel quarto minuto (di durata più lunga);
- 4 respiri nel quinto minuto.

La somma totale dei respiri è di 17.

### 5.5.1 Output

Ora si valuterà l'output prodotto dal codice, relativo al primo minuto e all'ultimo respiro rilevato: tale scelta è stata intrapresa poiché sarebbe troppo dispersivo rappresentare tali informazioni per tutti i 17 i respiri.

L'obiettivo è valutare infine la frequenza di respiro e controllare se il risultato è accettabile.


**Fig. 5.5.1. Prima parte dell'output**




**Fig. 5.5.2. Seconda parte dell'output**

Come si può osservare, la frequenza di respiro alla fine del primo minuto è di circa 4 respiri al minuto. Si noti l'output all'ultimo respiro:

```
Misurazione n. 1130
profondità attuale:  9.9 metri

Misurazione n. 1131
profondità attuale:  10.06 metri

Respiro n. 17 rilevato

Tempo trascorso dall'ultimo respiro  10.16 secondi
La freq di respiro filtrata è  4.81 respiri al min

Misurazione n. 1132
profondità attuale:  9.69 metri
```

**Fig. 5.5.3. Terza parte dell'output**

Al rilevamento dell'ultimo respiro, la frequenza vale 4.8 respiri al minuto: questo risultato è in accordo con quanto espresso inizialmente in quanto sono stati inseriti 17 respiri spalmati in 5 minuti e quest'ultimo respiro è stato rilevato all'incirca alla metà del quinto e ultimo minuto. Si considera questo risultato accettabile, e la simulazione portata a termine con successo.

Si fornirà ora un'ulteriore rappresentazione grafica dei risultati con i grafici dei segnali IP e PB generati in Python, mediante la creazione del *main*.

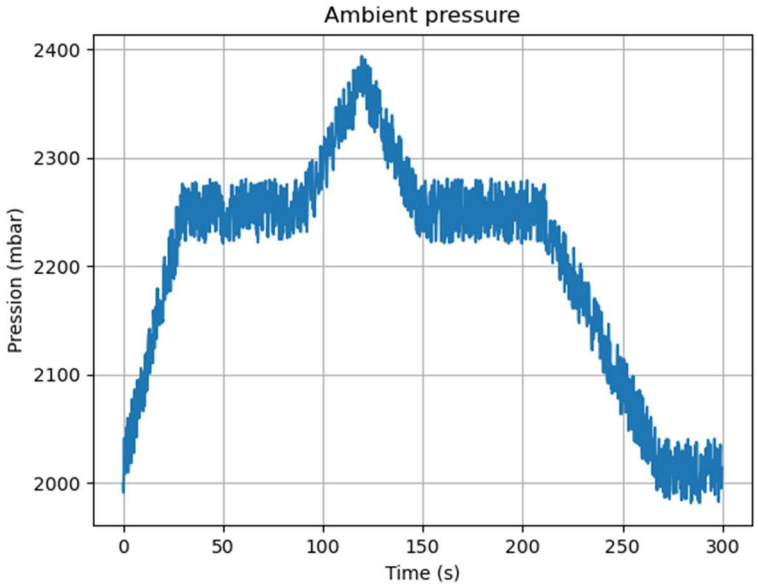



**Fig. 5.5.4. Segnale PB**

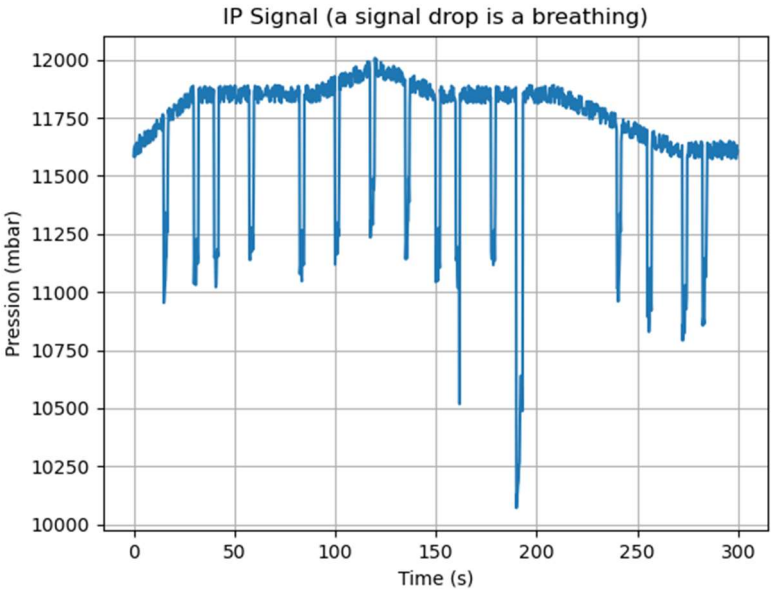



**Fig. 5.5.5. Segnale IP**

# Conclusioni e sviluppi futuri

In questo lavoro da tesi, è stata illustrata dapprima la progettazione meccanica del sistema di aggancio per lo scooter, e in seguito la parte relativa al case diver. Oltre a ciò si è trattato lo sviluppo di dell'algoritmo per il rilevamento della frequenza respiratoria del sub, esprimendone gli esiti.

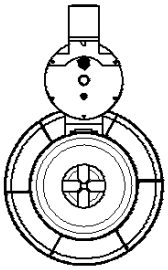
Si è consapevoli che il risultato ottenuto, per avere una maggior validità, avrebbe bisogno di un test reale da effettuare sul campo o in laboratorio, attraverso le misurazioni acquisite da profondimetri reali e di conseguenza rumori reali. Tuttavia i rumori che sono stati forniti sono abbastanza credibili, forse sono risultati essere anche troppo ampi, motivo per cui la buona riuscita del test con dati sfavorevoli è da considerarsi un buon risultato.

Il codice creato potrà essere migliorato per portare avanti lo sviluppo della sensoristica del progetto DiveSafe, per esempio inserendolo in altri script, per lavorare simultaneamente con gli altri sensori.

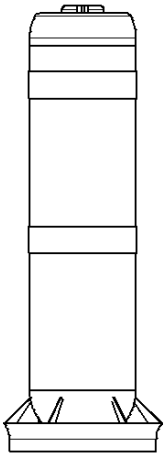
I possibili sviluppi futuri su cui si potrebbe lavorare sono molteplici; si può considerare lo studio e la verifica della stabilità in acqua per l'insieme scooter e DS oppure il miglioramento della stessa, per esempio utilizzando pesi o posizionando diversamente la DS. Un ulteriore possibilità di lavoro futuro riguarda la verifica di quanto il case diver possa essere effettivamente comodo durante un'immersione, valutando se nel caso rivedere le posizioni di cavi, fruste e di tutto ciò che possa impedire al subacqueo un'immersione comoda con un'ampia libertà nei movimenti.

# Appendici

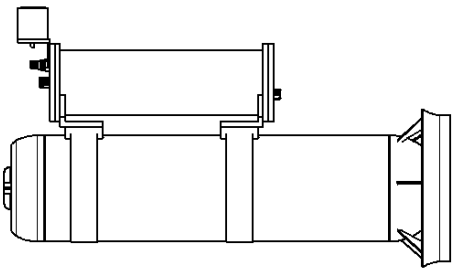
2D Sheet Scooter + DS



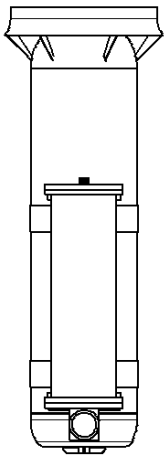
Vista frontale
Scala: 1:10



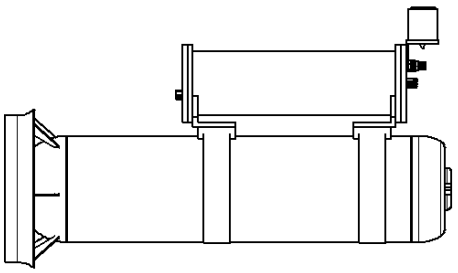
Vista dal basso
Scala: 1:10



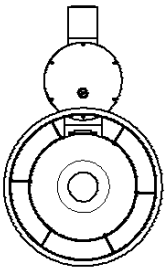
Vista da destra
Scala: 1:10



Vista dall'alto
Scala: 1:10

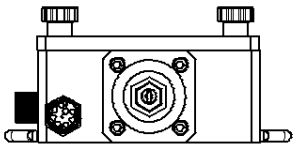


Vista da sinistra
Scala: 1:10

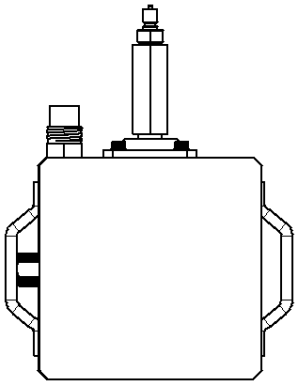


Vista posteriore
Scala: 1:10

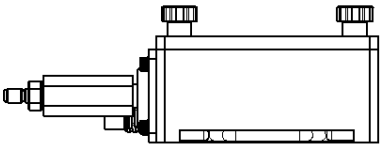
2D Sheet Case dive



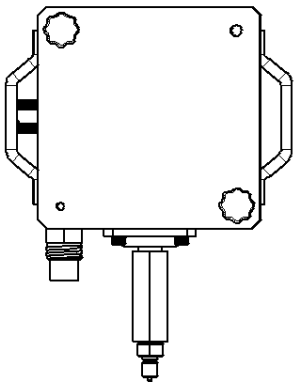
Vista frontale
Scala: 1:3



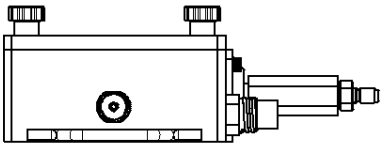
Vista dal basso
Scala: 1:3



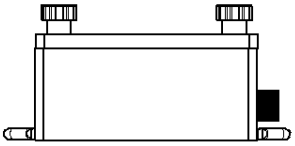
Vista da destra
Scala: 1:3



Vista dall'alto
Scala: 1:3



Vista da sinistra
Scala: 1:3



Vista posteriore
Scala: 1:3

## Bibliografia

[1] Barsky, Steven M.; Christensen, Robert W. "The Simple Guide to Commercial Diving. Hammerhead Press." (2004)

[2] Altepe, Corentin, et al. "Design and validation of a breathing detection system for scuba divers." Sensors 17.6 (2017)

[3] “About DiveSafe”, divesafe.eu/the-project/

[4] “Texas pressure sensor” te.com/usa-en/product-MS580330BA01-00.html

[5] “Bluerobotics sensor” bluerobotics.com/store/sensors-sonars-cameras/sensors/bar30-sensor-r1/

[6] Richard E. Moon. “Gas Toxicity During Diving.” (2019)

[7] Simon Pridmore, “Scuba Confidential: An Insider's Guide to Becoming a Better Diver” (2019)