UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Acquisizione dati da celle di carico per telemetria di barche a vela del Progetto Mètis Vela"

Relatore: Carlo Fantozzi

Laureando: Gianluca Rossi

ANNO ACCADEMICO 2021 – 2022

Data di laurea 21 settembre 2022

ABSTRACT

ITALIANO

Il lavoro di tesi consiste nell'integrazione di una cella di carico gestita da un microcontrollore, nel reparto elettronica del progetto universitario Mètis Vela.

Nella prima parte della discussione viene descritto il progetto in sé: da quali reparti è formato, qual è il suo scopo e come si arriva, attraverso la sinergia di tutti i reparti, a costruire una barca a vela funzionante, che parteciperà alla competizione annuale 1001 Vela Cup.

Si prosegue con la descrizione della circuiteria, delle parti della barca interessate e relative spiegazioni: cos'è una cella di carico? A cosa serve? Come si integra nel sistema già esistente?

Il cuore dell'esposizione è descrivere il lavoro svolto, che comprende: l'impermeabilizzazione della cella di carico e della scatola contenente la circuiteria, la discussione delle procedure per la gestione e la calibrazione della cella di carico tramite il software sviluppato.

La parte finale avrà lo scopo di mostrare i risultati del lavoro svolto, con aiuto di grafici e immagini.

INGLESE

The thesis' job consists of integrating a load cell handled by a microcontroller in the electronic department of University Mètis Vela project.

In the first part of the discussion, I will describe the project itself: how is it composed? What are its goals? How the sailing boat is made for competing at 1001 Vela Cup, held every year.

Then I will explain the electronic parts such as: What is a load cell? How do you integrate it in the system?

Later, I will discuss what my job was about: the waterproofing of the case containing the circuitry and the load cell, the load cell calibration and its management via the developed software running on the microcontroller.

In the final part will be the discussed the final product, with graphs and images.

INDICE

Capitolo 1

Progetto Mètis Vela

In questo capitolo verrà descritto il Progetto Mètis Vela, il progetto universitario di cui faccio parte, che mi ha permesso di sviluppare il lavoro di tesi. Verrà descritto lo scopo del progetto, assieme ai reparti che lo compongono. Nella parte finale verranno elencate le imbarcazioni prodotte negli anni.

Figura 1.1: logo Mètis Vela.

1.1 Descrizione progetto Mètis

Mètis Vela è il progetto dell'Università di Padova che si occupa della progettazione e costruzione di barche a vela sostenibili, tramite l'uso di materiali riciclabili.
Il progetto è gestito da studenti universitari provenienti da differenti corsi di studio, accomunati dalla passione per le barche a vela e l'acqua.

Possiamo trovare lo studente di Ingegneria Meccanica, che dopo aver acquisito conoscenze teoriche di progettazione in aula, può applicare quanto acquisito, con lo sviluppo delle componenti della barca a vela del Mètis.
Possiamo poi trovare lo studente, che indipendentemente dal corso di studio che frequenta, ha voglia di mettersi in gioco, svolgendo qualsiasi mansione necessaria.
È per questo che tutti gli studenti, a prescindere dal loro corso di studio, se appassionati e motivati, possono apportare un contributo al progetto.
Tramite la sinergia e la passione di ogni membro del progetto, si arriva alla costruzione di barche a vela da competizione.
La conoscenza viene trasferita alle nuove "leve" dai membri più esperti del gruppo, in modo da tramandare le tecniche perfezionate nei vari anni di appartenenza alla squadra.
Ogni anno il Mètis partecipa alla regata 1001 Vela Cup. È una competizione di barche a vela alla quale partecipano università italiane ed estere. La regata consiste nell'attraversare il tratto di mare o di lago, delimitato da boe di segnalazione, nel minor tempo possibile, utilizzando solo il vento come "propellente". L'organizzatore della regata definisce il regolamento che le barche di ciascuna squadra devono rispettare: massimo 4.60 metri di lunghezza, larghezza non superiore ai 2.10 metri e superficie velica consentita di 33 m^2. Si impone a ciascun partecipante l'uso di materiali riciclabili per la realizzazione dello scafo, e l'alluminio per le altre parti necessarie alla navigazione, come per esempio l'albero.

1.2 Flotta Mètis

La flotta Mètis è costituita da cinque barche a vela, costruite dal 2007 al 2019. La flotta verrà arricchita con la nuova imbarcazione Aletheia, che è attualmente in costruzione.

Le varie imbarcazioni si distinguono tra loro dal tipo di materiale e tecnica di progettazione utilizzata.

La prima imbarcazione patavina prodotta nell'anno accademico 2007-2008 è stata Argo. Ha gettato le basi del progetto Mètis utilizzando materiali riciclabili, che le hanno permesso di conquistare il terzo posto alla competizione velistica 1001 Vela Cup.

L'anno successivo, grazie all'esperienza acquisita nella costruzione di barche a vela, gli studenti del Mètis producono, con il compensato marino [1], Aura.
Aura è stata realizzata come evoluzione di Argo, alleggerendo l'ossatura interna in favore di un fasciame più resistente e rigido nella parte esterna.
Si è aggiudicata il primo posto nella competizione velistica 1001 Vela Cup.

Il 2012 è l'anno di svolta per il progetto. Areté è stata prodotta in fibra di lino con la tecnica dell'infusione. Questa è una tecnica utilizzata per la realizzazione di materiali compositi, che si basa sulla stesura delle fibre (di lino per Areté) sullo stampo, con l'apporto della resina, solo dopo aver messo in forma le fibre.

Nel 2015 gli ingegneri del Mètis danno alla luce la quarta barca del progetto: Ate.
La collaborazione con l'azienda Procotex, azienda produttrice di fibre unidirezionali di lino, già in precedenza partner per la realizzazione di Areté, fornisce al Mètis una nuova tipologia di fibra di lino dotata di un peso specifico inferiore.

L'ultima barca utilizzabile della flotta, Athena, sviluppata nel 2019, risultò all'epoca una delle prime barca a vela in Europa ad essere prodotta interamente con materiali riciclabili.
Per la sua costruzione la squadra si è affidata ad Arkema, azienda francese che fornì una speciale resina termoplastica [2].

Figura **1.2**: flotta Mètis Vela. L'ordine di costruzione è da destra verso sinistra.

1.3 Reparti del progetto

Il progetto Mètis è suddiviso in quattro reparti, che si occupano della sua gestione a tutto tondo. Ogni aspetto del progetto è seguito da studenti appassionati, che ogni anno ritagliano un po' del loro tempo personale per far funzionare questa "macchina" Mètis Vela.

Il **reparto di progettazione** si occupa del design e della verifica strutturale di ogni componente dell'imbarcazione, che viene poi prodotta dal settore tecnico.
Il progettista si occupa della creazione delle varie componenti della barca attraverso software CAD, e dei relativi test attraverso simulazioni.
Lo scopo del progettista è quello di rendere la barca il più veloce possibile in termini di prestazioni in gara, rispettando allo stesso tempo i regolamenti delle competizioni a cui il Mètis prende parte.
Qui figura anche la squadra di elettronica, che si occupa della costruzione dei sensori e del relativo codice per la rilevazione dei vari parametri di navigazione in acqua quali: direzione del vento, posizione in tempo reale dell'imbarcazione grazie alla tecnologia GPS e velocità della barca.
Durante la competizione la squadra di elettronica si occupa del monitoraggio dei dati spediti dai vari sensori, in modo da produrre grafici e documentazione utili per comprendere il comportamento in tempo reale dell'imbarcazione.
Il mio ruolo all'interno della squadra è quello di affiancare gli altri colleghi elettronici nella gestione del sistema, ampliandolo con nuova sensoristica.

I **cantieristi** si occupano della costruzione effettiva delle varie parti che andranno a comporre la barca. Oltre ad acquisire le competenze tipiche di un cantiere navale gli studenti cercano di trovare i migliori materiali che garantiscano elevate performance, rispettando allo stesso tempo l'ambiente.
La costruzione viene svolta dagli studenti stessi, coadiuvati dagli studenti veterani della squadra Mètis. La sede di costruzione è il cantiere del dipartimento di Ingegneria Industriale dell'Università di Padova.

Il **reparto di amministrazione** si occupa di gestire i fondi universitari stanziati per il progetto Mètis.
Poiché i soldi stanziati dall'università per il progetto sono limitati, gli studenti del reparto di amministrazione sono alla continua ricerca di partner esterni interessati alla sponsorizzazione del progetto Mètis.
Il reparto di amministrazione gestisce inoltre i vari canali social per pubblicizzare il progetto e darne visibilità.

I canali social attivi sono:

- Sito web [3]
- Pagina Instagram [4]
- Pagina Facebook [5]

Il compito dell'**equipaggio** è quello di testare e collaudare le imbarcazioni portandole al limite, in modo da capirne i punti forti e deboli, con delle sedute di allenamento velico vero e proprio.
Gli allenamenti permettono di prepararsi al meglio alla competizione 1001 Vela Cup.
I membri dell'equipaggio sono persone sportive e allenate, poiché la navigazione dell'imbarcazione richiede abilità natatorie e di resistenza.

Capitolo 2

Il sistema SailTrack

Nel seguente capitolo verrà descritto il sistema elettronico composto dai vari sensori presenti in barca e dal sensore di terra. Verranno dettagliate le tecnologie hardware e software presenti nel sistema SailTrack, ponendo particolare enfasi al nodo della rete sviluppato per la tesi: SailTrack Strain che si occupa della gestione della cella di carico.

F
i

Figura 2.1: logo SailTrack.

2.1 Descrizione sistema

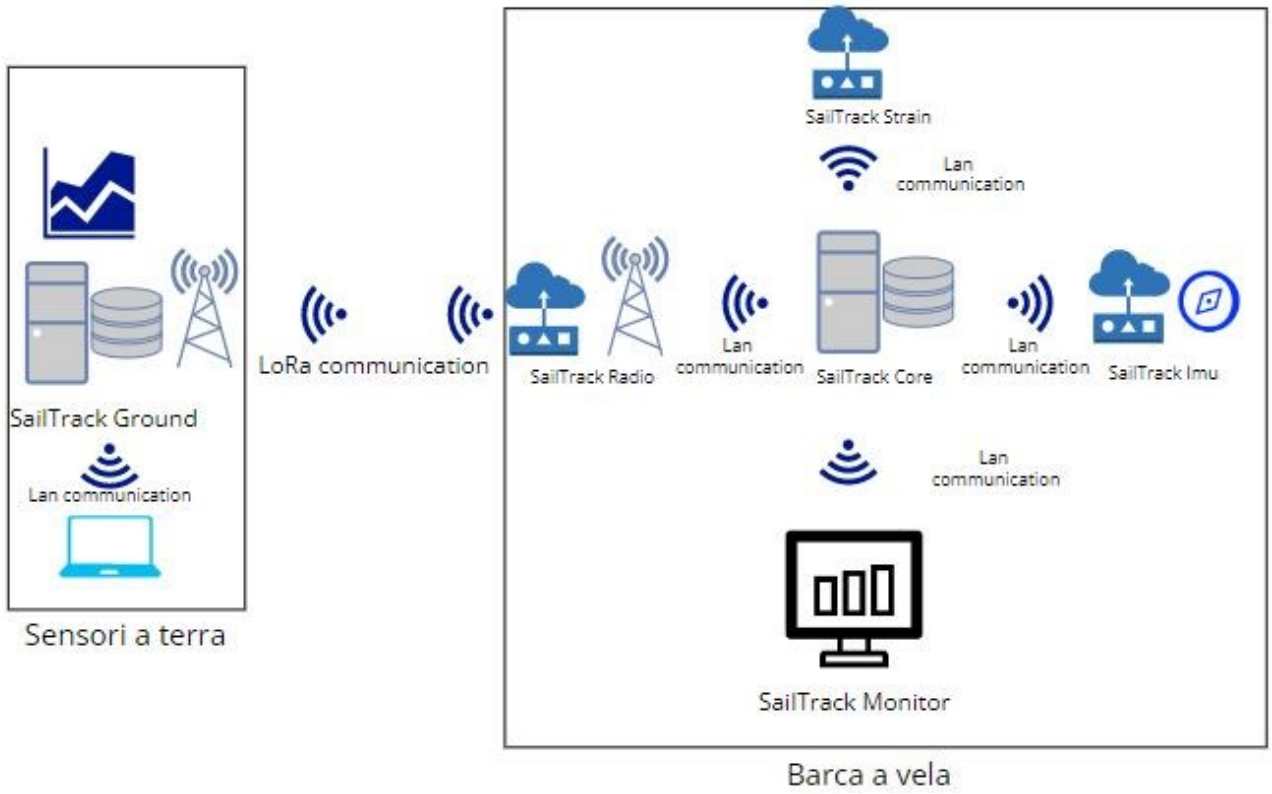

Figura 2.2: sistema SailTrack composto dal sistema di terra e della sensoristica presente sulla barca a vela.

SailTrack è il sistema distribuito composto da sensori eterogenei, **che "dialogano" tra di loro** attraverso comunicazione wireless, usando il protocollo MQTT [6] e il protocollo I^2C.

SailTrack, gestito da un sistema operativo Real-Time, è composto da due parti principali: i sensori presenti a bordo della barca e la sensoristica di terra.

2.1.1 Protocollo MQTT

MQTT è l'acronimo di Message Queuing Telemetry Transport e indica un protocollo di trasmissione dati TCP/IP [7] basato su un modello di pubblicazione e sottoscrizione, che opera attraverso un apposito broker dei messaggi.
È basato sul paradigma Publish-Subscribe, ossia sullo scambio di messaggi asincroni in formato Json tra vari client, che a seconda delle necessità possono sia produrre messaggi che riceverli.
I messaggi MQTT per essere pubblicati devono essere definiti in uno "scope" chiamato topic. Il mittente dichiara in quale topic intende spedire il messaggio; il destinatario per ricevere tale messaggio deve iscriversi allo stesso topic.
Il broker MQTT è colui che si occupa di smistare i vari messaggi MQTT ai vari destinatari. La lista dei topic presenti è gestita dal broker.

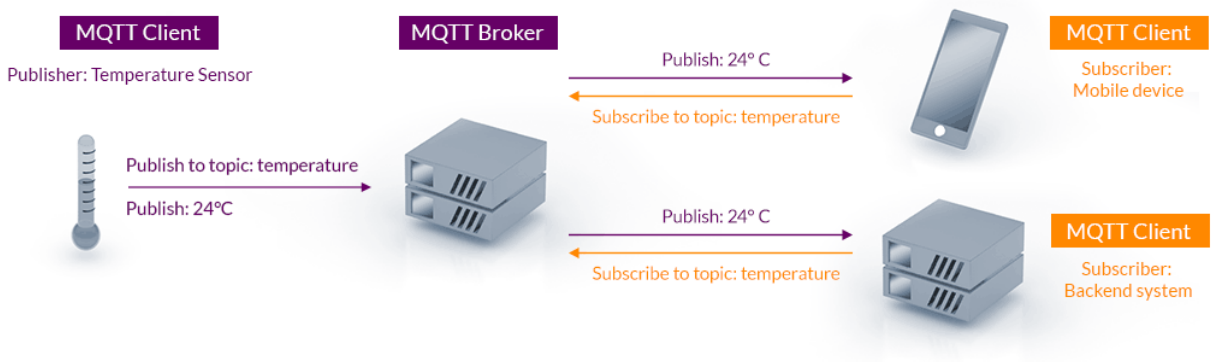

Figura 2.3: protocollo MQTT composto dai client e del broker. Fonte: https://mqtt.org/

Il protocollo MQTT è adatto al sistema sopra descritto perché i dispositivi connessi hanno un bitrate ridotto e consumano poca energia; per questi motivi il protocollo MQTT è lo standard nell' Internet Of Things.

2.1.2 Protocollo I^2C

La comunicazione fisica tra le varie **componenti di ciascun sensore** è gestita dal protocollo I^2C.
È un tipo di comunicazione seriale bidirezionale sviluppata dall'azienda Philips nel 1982 [8].
Le due linee di comunicazione bidirezionale, chiamate "SCL" (**S**erial **CL**ock) e "SDA" (**S**erial **DA**ta), trasportano rispettivamente la tempistica di sincronizzazione (chiamata anche "clock") e i dati.
Per la rappresentazione dei valori logici "1" e "0", si usano l'alimentazione del circuito ($\mathbf{V_{dd}}$) e la massa ($\mathbf{V_{ss}}$).
A causa dei disturbi elettromagnetici, questi valori di tensione vengono modificati, generando un'alterazione del segnale originale presente in ingresso. È necessario impostare le suddette linee in uno stato perfettamente conosciuto, che non possa portare a stati di incertezza sulle letture effettuate.

Per risolvere questi problemi si aggiungono le resistenze di pull-up e di pull-down.
Si inserisce una piccola resistenza tra il pin di ingresso e la tensione di riferimento, per fare in modo che all'ingresso sia sempre presente un segnale certo. Se la resistenza viene inserita tra il pin e la tensione di alimentazione $\mathbf{V_{dd}}$, si parlerà di resistenza di pull-up [9], se invece viene inserita tra il pin e la massa $\mathbf{V_{ss}}$ (normalmente GND o 0V) si parlerà di resistenza di pull-down.

2.1.3 Sistema operativo Real-Time

I protocolli MQTT e I^2C sono importanti per definire le regole di comunicazione rispettivamente, tra i vari sensori dell'imbarcazione velistica e le componenti di ciascun sensore; tuttavia, senza una supervisione software, che il sistema operativo svolge nei computer, i vari nodi della rete non sarebbero in grado di comunicare tra di loro.
Il sistema operativo utilizzato nel sistema SailTrack non è un normale sistema operativo desktop, per il fatto che è troppo energivoro e abbondante di funzionalità per l'uso in sistemi integrati, ma è un sistema operativo Real-Time, adatto per l'uso in dispostivi con capacità computazionali, dimensione di memoria e numero di porte input-output, ridotte.

L'uso di un sistema operativo Real-Time è necessario per il fatto che il sistema SailTrack stesso è un sistema Real-Time [10]: ovvero utilizza le funzionalità di un sistema operativo Real-Time per svolgere le proprie operazioni correttamente.
Un sistema Real-Time è un calcolatore in cui la correttezza del risultato delle sue elaborazioni dipende non solo dalla correttezza logica ma anche dalla correttezza temporale, espressa in termini di tempo entro il quale un'operazione deve essere eseguita. Un sistema Real-Time può essere estremamente lento, ma deve essere deterministico, garantendo un limite superiore preciso al tempo necessario alla computazione, sapendo sempre entro quanto tempo un processo del sistema verrà eseguito.

Il sistema operativo Real-Time utilizzato, che si occupa di gestire la corretta esecuzione dei processi entro i tempi limite definiti, è **FreeRTOS** [11].
FreeRTOS è un popolare kernel open source, sviluppato per essere adatto all'utilizzo nei sistemi embedded come sistemi di controllo industriali, robot e in molte altre applicazioni.
FreeRTOS si occupa dello scheduling Real-Time dei processi, fornendo primitive di sincronizzazione e temporizzazione per i vari processi presenti nel sistema.
Fornendo così poche funzionalità, sembrerebbe limitante dal punto di vista della produttività; in realtà le funzionalità presenti lo rendono un sistema operativo Real-Time molto leggero e privo di overhead rispetto alla concorrenza.

L'aggiunta di componenti è possibile scaricando l'apposita funzionalità necessaria; si può definire quindi FreeRTOS come microkernel [12], ovvero un kernel modulare, dotato di fabbrica delle sole operazioni basilari come scheduling, gestione processi, controllo dispostivi input-output, con la possibilità dell'aggiunta di componenti extra al bisogno.

I dati raccolti dai sensori vengono salvati su influxDB [13], un popolare database open source, utilizzato per il salvataggio di dati prodotti in serie temporali.
Le serie di dati temporali sono misurazioni di fenomeni, che sono processate e salvate nel tempo.

Per l'interfacciamento al database influxDB si usa il software per l'analisi dei dati Grafana [14], che permette la creazione di grafici, la visualizzazione dei database presenti con relativi dati e la loro interrogazione tramite il linguaggio InfluxQL simile per grammatica a SQL.

2.2 Componentistica

Ogni sensore è dotato della circuiteria minima necessaria per il suo funzionamento: microcontrollore per l'elaborazione dei dati ricevuti dal sensore e batteria di alimentazione esterna. Ogni microcontrollore è provvisto di antenna Wi-Fi per interfacciarsi con gli altri sensori del sistema SailTrack.

La comunicazione wireless della rete locale in barca è gestita dal modulo SailTrack Core che si occupa di smistare i vari pacchetti nella rete di comunicazione della barca stessa.

2.2.1 SailTrack Core

SailTrack Core è il cuore centrale della sensoristica della barca a vela del Mètis; è responsabile della gestione dei topic, decidendo che nodo del sistema SailTrack può trasmettere e a chi.
Gestisce le connessioni Wi-Fi dei vari moduli e smista i pacchetti MQTT dai mittenti ai relativi destinatari, svolgendo il ruolo di broker MQTT.
SailTrack Core è costituito da un single board computer (SBC) [15] modello Raspberry Pi 3 Model B+ [16], che è un vero e proprio computer composto da CPU, memoria RAM e porte di input e output. È in grado di gestire la rete Wi-Fi presente a bordo dell'imbarcazione, grazie alla presenza del modem Wi-Fi contenuto nel Raspberry Pi.
SBC è alimentato da una batteria da 10000 mAh tipologia 1260100 Li-Po Battery; la gestione della batteria è svolta dal Power management system [17] esterno (negli altri sensori è già incorporato nella scheda) modello Geekworm X708 UPS HAT, che si occupa di mantenere una tensione costante, necessaria per il corretto funzionamento dell'SBC, fornendo energia alle componenti che la richiedono.
È dotato di una memoria di massa di 32 GB per il salvataggio dei dati raccolti. Il software che gestisce il microcontrollore di SailTrack Core è DietPi [18], una versione personalizzata di Raspberry Pi OS. I dati raccolti vengono salvati sul database InfluxDB, accessibile via software grafico Grafana.

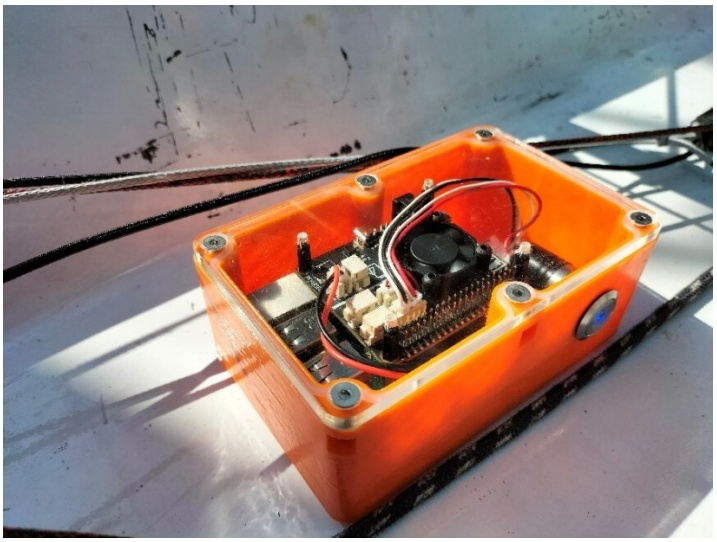

Figure 2.4: SailTrack Core.

2.2.2 SailTrack Imu

L'equipaggio, per la navigazione della barca, necessita di varie informazioni, le più importanti sono: la velocità della barca SOG (Speed Over Ground), la traiettoria della barca COG (Course Over Ground), la velocità del vento, la direzione del vento e il rollio dell'imbarcazione.

SailTrack Imu (Inertial Measurement Unit) [19] è un sensore dotato di multi-assi che si occupa di monitorare la dinamica dell'imbarcazione velistica attraverso l'uso di accelerometri, giroscopi e magnetometri. È dotato del microcontrollore LILYGO modello TTGO T7 Mini32 V1.3 ESP32-WROVER-B [20], che si occupa dell'elaborazione dei dati raccolti dal sensore Imu. Il microcontrollore ESP-32 è alla base dei sensori SailTrack presenti sulla barca (escluso SailTrack Core che usa il microcontrollore Raspberry Pi).

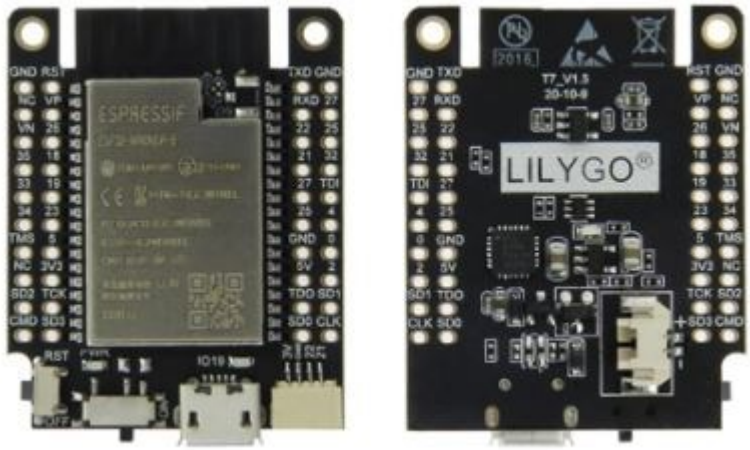

Figura 2.5: Microcontrollore TTGO ESP-32.
Fonte: https://it.aliexpress.com/item/32977375539.html

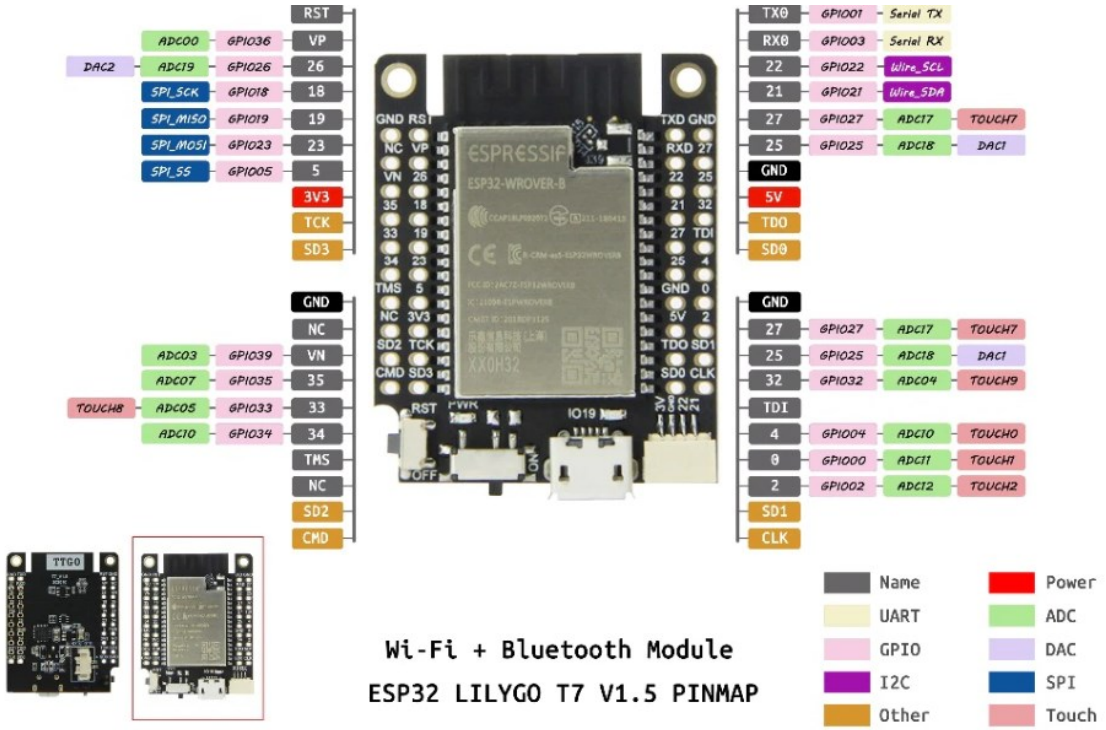

Figura 2.6: diagramma dei pin del microcontrollore ESP-32.
Fonte: https://it.aliexpress.com/item/32977375539.html

Il microcontrollore ESP-32 è dotato di bluetooth e antenna Wi-Fi 802.11 b/g/n per connessioni LAN: la rete di SailTrack Core nel sistema SailTrack. ESP-32 utilizza un processore dual-core a 32 bit con frequenza di lavoro pari a 40MHz.

I pin utilizzati nel progetto sono: 5V per l'alimentazione, GND per la massa, 25 il pin dati e 27 il pin della temporizzazione.
I pin 25 e 27 sono utilizzati per la comunicazione seriale dal protocollo I^2C.
Nota: il diagramma dei pin di **Figura 2.6** è riferito alla versione 1.5 del microcontrollore. La versione 1.3 (quella utilizzata nel progetto) e la 1.5 sono identiche a livello di diagramma dei pin.

SailTrack Imu è responsabile della raccolta dei dati, che permettono di calcolare il rollio dell'imbarcazione e la traiettoria della barca.

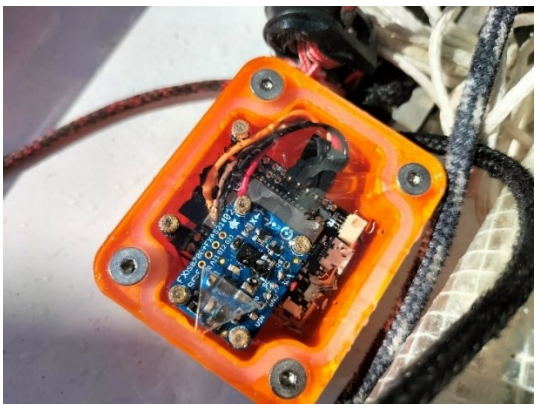

Figura 2.7: SailTrack Imu attaccato alla barca durante una sessione di allenamento.

Figura 2.8: grafico del rollio preso da Grafana. Il rollio dell'imbarcazione è misurato in gradi: zero gradi indica la situazione di stabilità della barca.

2.2.3 SailTrack Monitor

I dati grezzi raccolti dai vari sensori vengono inviati in maniera wireless a SailTrack Core, che attraverso delle operazioni di filtraggio e manipolazione [21], rende i dati processati presentabili per la visualizzazione all'equipaggio.

Figura 2.9: grafico che rappresenta l'andamento della velocità della barca (speed over ground).

I dati grezzi della SOG prodotti dal sensore nella **Figura 2.9**, sono espressi in millimetri al secondo, ma visto che la velocità delle barche si esprime in nodi nautici, la conversione è necessaria.
Prendendo la velocità espressa in millimetri al secondo delle ore 8:39 di circa 2000 mm/s, possiamo dire che la barca, attraverso la divisione per 514.6, viaggiava alla velocità di circa 3.88 nodi nautici.

La componente della rete SailTrack che si occupa di mostrare i dati è SailTrack Monitor, che è dotato di un microcontrollore con batteria esterna, e di un display di tecnologia E-ink [22].
Questa tecnologia è progettata per avere la stessa leggibilità di un foglio di carta.
Tale tecnologia sfrutta il processo dell'elettroforesi: il processo indica il movimento, tramite campo elettrico, di particelle elettricamente cariche immerse in un fluido.
E-ink prevede l'uso di sfere di dimensione molto ridotta all'interno dello schermo; normalmente le sfere utilizzate hanno un diametro di 3 millimetri.
Queste sfere sono caricate elettricamente: una semisfera è nera con carica positiva mentre l'altra semisfera è bianca con carica negativa.

Utilizzando campi elettrici è possibile orientare le sfere, agendo solamente sulla componente positiva o negativa di ciascuna sfera; alterando la posizione delle sfere è possibile modificare il contenuto mostrato dallo schermo.
Questo permette di realizzare supporti sottili che richiedono alimentazione solamente quando si vuole modificare la configurazione delle sfere. Necessitando di alimentazione solo per il cambio di schermata del display, la tecnologia E-ink permette un basso consumo di energia.
Un altro vantaggio molto importante della tecnologia E-ink è quello di affaticare meno gli occhi rispetto ad altre tecnologie di display, mantenendo allo stesso tempo una buona visibilità alla luce del sole.

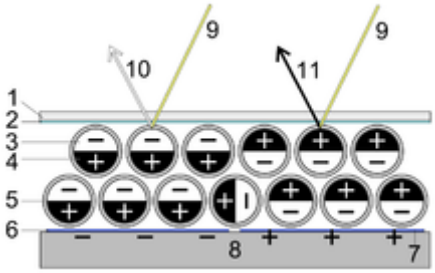

Figura 2.10: tecnologia E-ink, dove ogni sfera si allinea in base al campo elettrico che si produce [22].

2.2.4 SailTrack Radio

La comunicazione wireless tra la sensoristica della barca e quella di terra è gestita dal sensore **SailTrack Radio**. Questo modulo è responsabile della localizzazione della barca a vela tramite la tecnologia GPS. Usa il microcontrollore ESP-32 alimentato da una batteria LiPo. SailTrack Radio è dotato di un'antenna GPS per la localizzazione e di un'antenna LoRa per la comunicazione con il sensore di terra.

La comunicazione wireless a lunga distanza, tra la sensoristica a bordo della barca e il sensore di terra, è resa possibile grazie alla tecnologia LoRa.

LoRa [23] è una tecnologia wireless sviluppata per consentire comunicazioni di trasmissione dati a bassa velocità, su lunghe distanze, tramite sensori e attuatori per applicazioni M2M (Machine-to-Machine) e Internet of Things [24].

Figura 2.11: logo di LoRa.

Utilizzando frequenze di trasmissione molto basse, LoRa permette la trasmissione di informazioni su lunghe distanze (fino a 10 km in assenza di ostacoli), garantendo allo stesso tempo, una ridotta dissipazione di potenza.

La comunicazione via tecnologia LoRa è crittata con crittografia end-to-end, garantendo la privacy delle informazioni trasmesse.

Figura 2.12: SailTrack Radio con antenna LoRa (quella in nero).

2.2.5 SailTrack Ground

L'altro nodo presente nella rete di comunicazione di LoRa, è SailTrack Ground.

SailTrack Ground, costituito dalle stesse componenti hardware di SailTrack Core, si occupa di ricevere i dati raccolti dalla sensoristica a bordo della barca, creando grafici sulla posizione della barca, sulla velocità e direzione del vento, sulla velocità della barca e su altri parametri utili per verificare lo stato in tempo reale dell'imbarcazione.

Oltre ad essere dotato della stessa tecnologia software equipaggiata su SailTrack Core, ha uno script aggiuntivo che permette di trattare i dati ricevuti via LoRa, come dati prodotti da un sensore fittizio presente nella rete locale di terra: questo permette la creazione dei grafici sopracitati e il salvataggio di backup dei dati raccolti dai sensori.

2.2.6 SailTrack Strain

L'ultimo nodo aggiunto alla rete è **SailTrack Strain**, da me sviluppato per il progetto di tesi.
È il componente del sistema SailTrack che si occupa di misurare il carico a cui sono sottoposte le varie sartie presenti nella barca a vela; queste sono delle corde in acciaio inox, responsabili della stabilità dell'albero della barca, che ne evitano una caduta laterale.
Lo strumento che si occupa di effettuare queste misure è la cella di carico.
La cella di carico è un dispositivo costruito in acciaio inox, usato per misurare una forza applicata su un oggetto tramite la misura di un segnale elettrico che varia a causa della deformazione che tale forza produce sul componente [25]: il carico applicato deforma il corpo in esame, che contiene al suo interno gli estensimetri.
L'estensimetro è uno strumento di misura utilizzato per rilevare piccole deformazioni dimensionali di un corpo sottoposto a sollecitazioni meccaniche o termiche [26]; il corpo di cui si vogliono misurare le deformazioni contiene al suo interno gli estensimetri. Applicando una forza sul materiale lo si deforma assieme agli estensimetri; per capire la deformazione del materiale, si controllano le deformazioni subite dagli estensimetri.
Quando sono note le caratteristiche meccanico-fisiche del materiale, risulta facile capire a quali carichi il materiale è sottoposto.

La misurazione non viene effettuata utilizzando un solo estensimetro, ma viene effettuata con quattro estensimetri nella configurazione a Ponte di Wheatstone [27], che garantisce una misurazione più precisa rispetto all'uso di un singolo estensimetro.

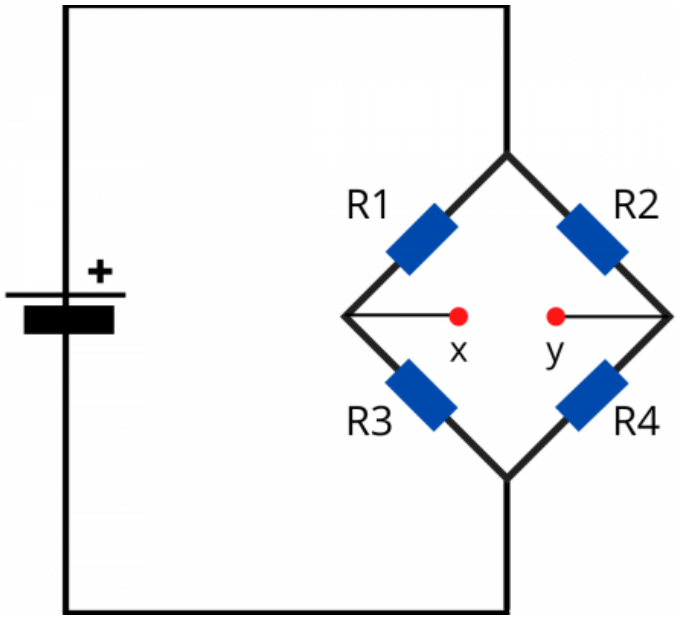

Figura 2.13: Ponte di Wheatstone, dove R1, R2, R3 e R4 rappresentano quattro estensimetri.

SailTrack Strain è composto dalla cella di carico collegata alla circuiteria composta da: microcontrollore ESP-32, batteria tipo103040 lipo battery 1s con capacità pari a 1200 mAh e un amplificatore che riceve le variazioni degli estensimetri in termini di tensione.
La necessità dell'amplificatore è una scelta obbligata per il fatto che i valori di tensione misurati dal ponte di Wheatstone sono nell'ordine dei millivolt rispetto alla tensione di lavoro del microcontrollore ESP-32 che è di 5 Volt.
Per amplificare i valori di tensione della cella di carico, nella necessità di renderli comparabili con la tensione di 5V, è stato scelto l'amplificatore su chip HX711.
Il chip HX711 è un circuito integrato che contiene al suo interno un amplificatore differenziale [28] per amplificare il segnale generato dalla cella di carico, e un convertitore analogico-digitale.

Figura 2.14: chip HX711 contenente l'amplificatore e il convertitore analogico-digitale.

Un convertitore analogico-digitale [29] è un circuito elettronico che si occupa della digitalizzazione del segnale presente al suo ingresso, trasformandolo da segnale continuo (tempo continuo valori continui) a segnale digitale (tempo discreto valori discreti). La discretizzazione dell'asse del tempo viene effettuata tramite il campionamento, seguita dalla discretizzazione dell'asse dei valori con la

procedura di quantizzazione. Dopo aver ottenuto il segnale digitale, lo si trasforma in uno stream di bit.

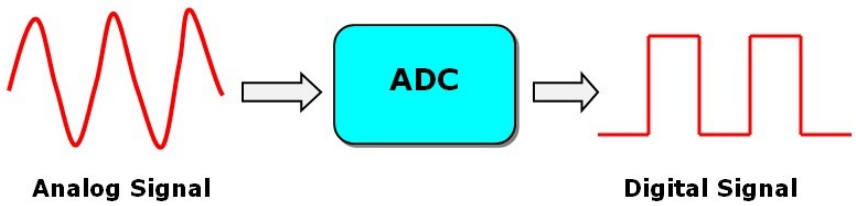

Figura 2.15: procedura di digitalizzazione di un segnale attraverso il convertitore analogico-digitale.

L'unico sensore del sistema SailTrack ad essere dotato di pulsante on-off è SailTrack Core.
Gli altri sensori non sono dotati di tale pulsante perché è stato sviluppato il Deep Sleep, una tecnologia che permette di verificare lo svolgimento di attività della circuiteria dei vari sensori. Se un dispositivo non riesce a trasmettere i dati raccolti o non è in grado di connettersi dopo un numero prefissato di tentativi alla rete Wi-Fi creata da SailTrack Core, entra nello stato di Deep Sleep, che è una modalità di super risparmio batteria paragonabile come consumo allo spegnimento del sensore.

La motivazione dello sviluppo del Deep Sleep al posto dell'inserimento del pulsante di accensione, è nata dal fatto che i sensori, nonostante la loro circuiteria sia contenuta in scatole impermeabili stampate in 3D, sono spesso a contatto con l'acqua, che potrebbe penetrare a causa della pressione che agisce sulle scatole durante le cadute in acqua delle barche.

Il pulsante di accensione esterno alla scatola risulterebbe un alleato dell'acqua per favorirne il passaggio, in caso di rottura del pulsante stesso.

Un altro motivo è che i sensori restano attaccati alla barca; quindi, sarebbe complicato togliere i sensori dalla barca ogni volta solo per accenderli e spegnerli. Usando il Deep Sleep, i sensori restano sempre accesi.

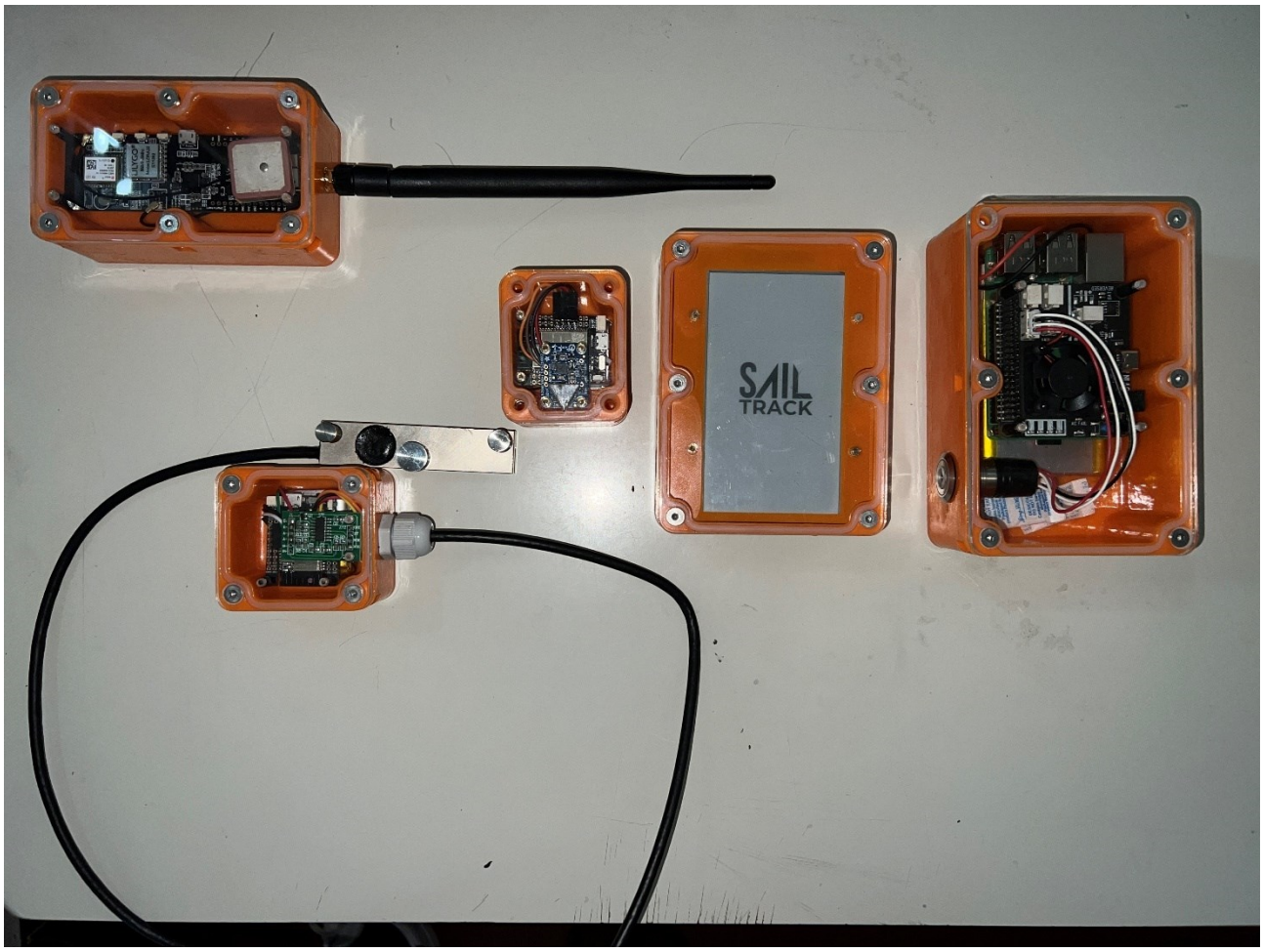

Figura 2.16: componenti del sistema SailTrack presenti sulla barca.

La scatola con l'antenna è SailTrack Radio. Da sinistra a destra troviamo: SailTrack Strain dotato di cella di carico in acciaio inox, SailTrack Imu, SailTrack Monitor e SailTrack Core.

Nota: SailTrack Core è inserito in un vano scavato sullo scafo della barca, chiuso poi da un coperchio: questo permette di proteggerlo dall'acqua e da danneggiamenti, in maniera più marcata rispetto agli altri sensori.

Capitolo 3

Lavoro svolto

In questo capitolo verranno discusse le parti della barca interessate nel lavoro di sviluppo del sensore SailTrack Strain, discutendo successivamente di cosa si è trattato il lavoro di tesi, dettagliando le soluzioni hardware e software utilizzate.

3.1 Parti della barca interessate

Una barca a vela da competizione è formata da varie componenti, che ne permettono la rigidità e la manovrabilità necessarie durante gli allenamenti e le gare.

La cella di carico di SailTrack Strain, durante le sessioni di allenamento e le gare, viene fissata ad una delle sartie presenti nella barca.

Per dare stabilità alle sartie nel sostegno dell'albero, vengono inserite delle **crocette**, ossia dei puntoni in alluminio che aiutano a migliorare l'azione delle sartie sull'albero e ad aumentarne la rigidità. L'albero è un grande palo verticale fissato perpendicolarmente allo scafo della barca, il cui scopo è quello di sopportare il peso della maggior parte delle manovre.

Figura 3.1: crocetta (struttura in nero nella foto) attaccata ad una sartia.

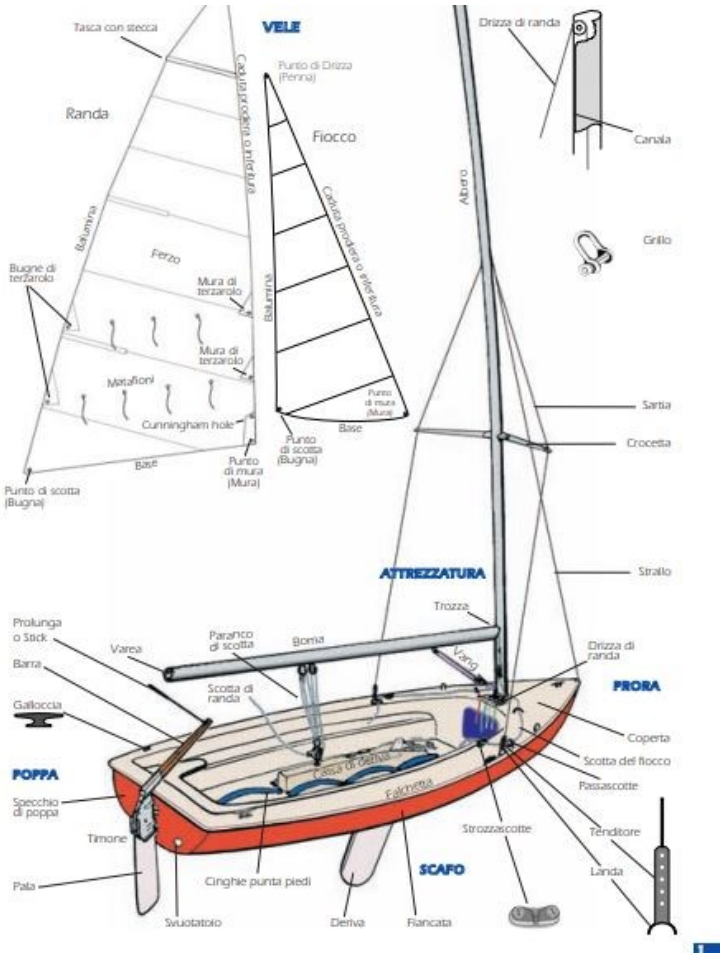

Figura 3.2: schema di una barca a vela, con indicate le principali componenti.

3.2 Impermeabilizzazione scatola della circuiteria

Il primo lavoro di tesi è stato l'impermeabilizzazione della scatola contenente la circuiteria del sensore Strain del sistema SailTrack. L'impermeabilizzazione della circuiteria è resa necessaria dal fatto che i circuiti elettronici non sono in grado di lavorare in presenza di acqua; l'acqua in un circuito non è mai una cosa piacevole perché può causare malfunzionamento o la rottura del circuito stesso.

La costruzione della scatola è stata realizzata tramite stampante 3D da un mio collega del reparto di elettronica; l'impermeabilizzazione è stata effettuata attraverso l'applicazione di una resina marina epossidica [30].

Un lato della scatola è stato lasciato libero, per l'inserimento della circuiteria e per un eventuale modifica delle componenti elettroniche. Questa parte è stata chiusa da un coperchio in plexiglass, impermeabilizzato da un o-ring, una gomma siliconica a tenuta stagna, tenuto in posizione da bulloni e dadi in acciaio inossidabile.

Il passo successivo è stato quello di trovare una soluzione per rendere impermeabile la cella di carico e la connessione cablata tra la cella e la circuiteria.

Figura 3.3: componente SailTrack Strain dotato della cella di carico e relativa circuiteria.

Da come si vede in **Figura 3.3**, il cavo nero impermeabile parte dalla cella di carico e termina con la circuiteria.

Per impermeabilizzare la cella di carico è stata stesa della gomma liquida sulle due parti esposte all'acqua.

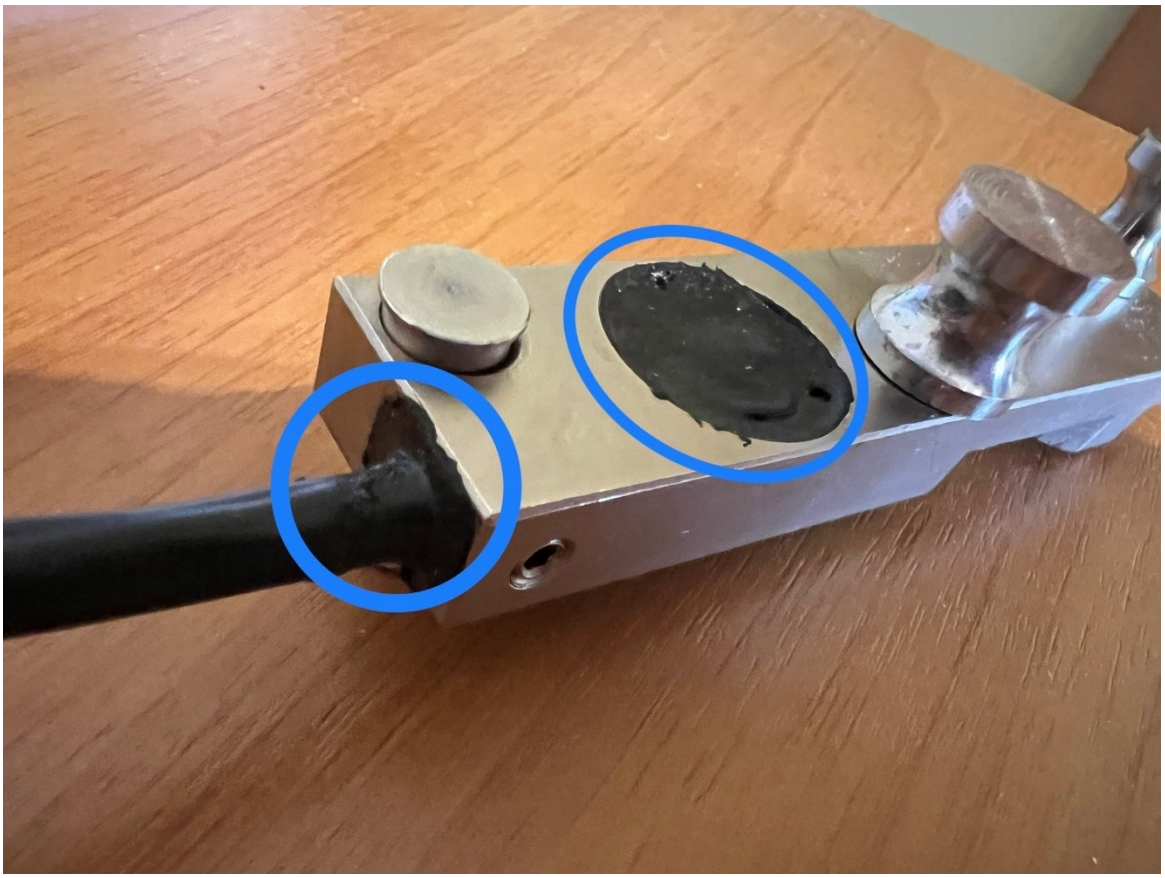

Figura 3.4: punti di applicazione della gomma liquida per l'impermeabilizzazione della cella di carico.

La gomma liquida impermeabilizzante, una volta lasciata indurire, ha reso impermeabile le parti critiche della cella di carico.

Per collegare la circuiteria e la cella di carico era necessario un foro che permettesse il passaggio del cavo. A tale scopo, in fase di stampa della scatola è stata lasciata un'area (oltre allo spazio chiuso dal coperchio in plexiglass). Tuttavia, tale ingresso doveva essere impermeabilizzato per evitare che l'acqua entrasse a contatto con la circuiteria.

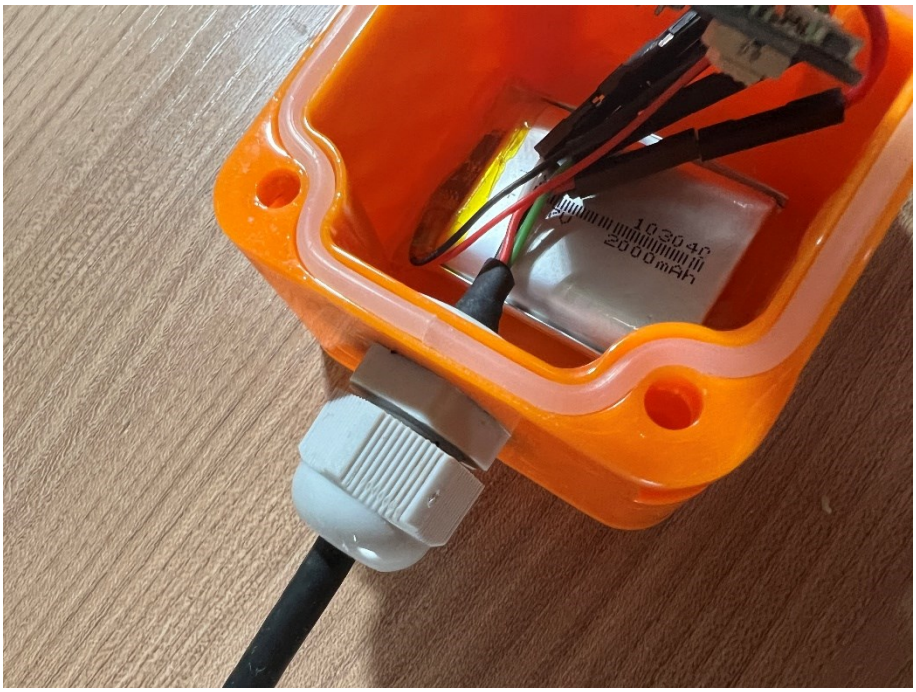

Figura 3.5: passacavo della circuiteria della cella di carico.

Per rendere il cavo e la circuiteria impermeabili è stato utilizzato il passacavo grigio di **Figura 3.5** che tramite l'utilizzo di o-ring al suo interno ha reso possibile l'impermeabilizzazione della scatola.

Le viti che tengono saldi i pioletti alla cella di carico sono state sostituite con viti in acciaio inox, per evitare che si ossidino.

Per il test di tenuta è stato immerso il componente SailTrack Strain in una bacinella d'acqua per 30 minuti, ad una profondità di circa 50 centimetri, verificando l'effettiva bontà dell'impermeabilizzazione svolta.

3.3 Costruzione nuovi pioletti per la cella di carico

Per testare il funzionamento della cella di carico era necessario capire quale fosse il diametro di sartie per cui la cella di carico misurava delle deformazioni.

Svolgendo dei test con vari tipi di corde, si è notato che il diametro minimo di corda necessario affinché si verificasse una deformazione degli estensimetri della cella di carico, dovuta alla forza esercitata dalla corda sulla cella stessa, era di 3.12 millimetri.
Il progetto Mètis per le sue imbarcazioni utilizza sartie di 3 millimetri; perciò era necessario trovare una soluzione che permettesse di misurare variazioni sulla cella di carico.

Le due soluzioni alternative erano l'inspessimento delle sartie per raggiungere il diametro minimo di 3.12 millimetri o la modifica dei pioletti della cella di carico.

La soluzione scelta è stata la modifica del pioletto centrale della cella di carico, in modo da deformare una corda da 3 millimetri con pioletto più grande rispetto a quello di fabbrica (pari a 10 millimetri), nella stessa maniera, utilizzando però il pioletto di vendita e una corda di diametro pari a 6 millimetri.

La scelta di modificare il pioletto e non la cella di carico è maturata dal fatto che se si fosse deciso di inspessire le sartie, sarebbe stato necessario trovare un materiale rigido ma allo stesso tempo

flessibile, che avesse le stesse proprietà meccaniche delle sartie in acciaio inox. Il problema era che bisognava ricoprire tutte le sartie con questo materiale, che in più non era di semplice realizzazione.

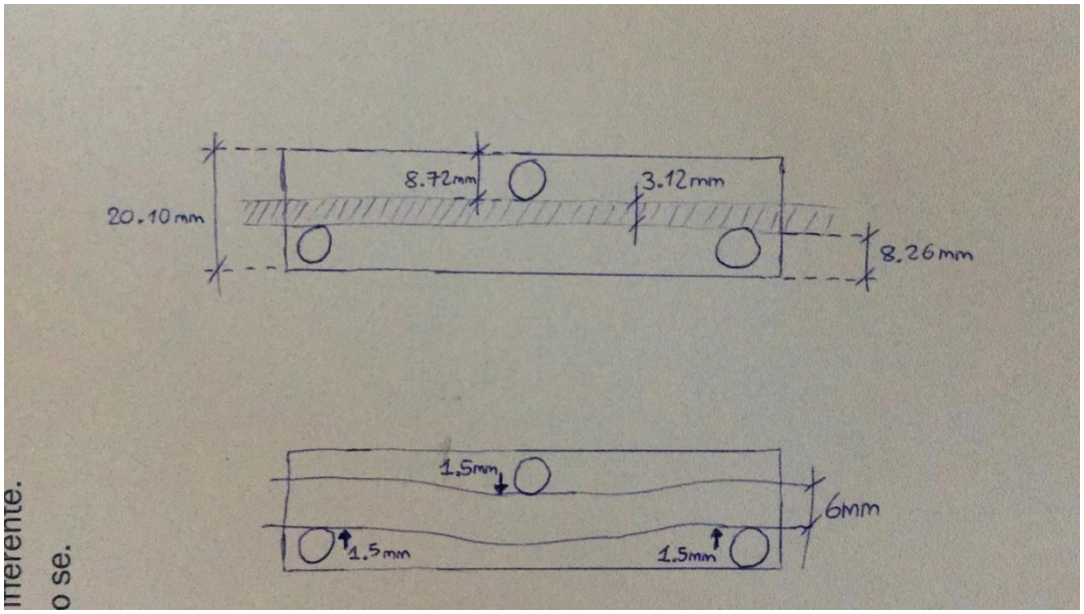

Figura 3.6: quotatura cella di carico e test con corda da 6 millimetri.

Dai test svolti si è giunti alla conclusione che il diametro del nuovo pioletto debba essere di 13 millimetri.

Il nuovo pioletto è stato realizzato dai tecnici del Dipartimento di Ingegneria Industriale, tramite l'utilizzo del tornio.

Figura 3.7: pioletto di fabbrica a sinistra e a destra quello nuovo costruito.

3.4 Sviluppo del codice per il controllo della cella di carico

Lo sviluppo del codice per il controllo della cella di carico necessita di un linguaggio di programmazione che riesca ad interfacciarsi con la memoria senza overhead, in modo da rendere il codice leggero, pulito e il più possibile veloce per l'esecuzione su microcontrollore ESP-32. Per uniformità con il sistema SailTrack esistente, che utilizza il C++, si è deciso di usare lo stesso linguaggio anche per lo sviluppo di SailTrack Strain.
Lo scopo è quello di calibrare la cella di carico e leggere i dati misurati, per capire qual è lo sforzo applicato alle sartie.
Il framework utilizzato è Arduino, usato per la gestione del sistema operativo Real-Time FreeRTOS e i processi asincroni.
Il codice sviluppato si compone dell'inizializzazione dei componenti del sistema SailTrack Strain, della gestione delle chiamate asincrone per avviare la procedura di calibrazione, della funzione di calibrazione con il relativo salvataggio dei dati su memoria EEPROM e della funzione per la lettura dati della cella di carico.

3.4.1 Inizializzazione componenti

L'inizializzazione del sistema viene fatta chiamando la funzione *setup()* del framework Arduino:

```
SailtrackModule stm;
HX711 hx;

void setup() {

        stm.begin("strain", IPAddress(192, 168, 42, 105), new ModuleCallbacks());
        hx.begin(HX711_DOUT_PIN, HX711_SCK_PIN);
        EEPROM.begin(EEPROM_ALLOC_SIZE_BYTES);
        loadCalibration();
        stm.subscribe("sensor/strain0/calibration");
}
```

Per controllare l'amplificatore tramite il protocollo seriale I^2C, è necessario includere la libreria HX711.
Il sistema operativo Real-Time FreeRTOS e la gestione dei processi asincroni sono contenuti nella libreria Arduino.
Per il salvataggio dei dati della calibrazione si usa la EEPROM presente sul microcontrollore, gestita dalla libreria EEPROM. I dati salvati nella memoria EEPROM sono l'offset della cella di carico e il divider che si occupa della conversione dei dati grezzi della cella in dati espressi in chilogrammi.
Essendo il componente SailTrack Strain parte integrante del sistema SailTrack, è necessario creare l'istanza della classe SailTrackModule, che ne definisce nome, indirizzo IP e crea un'istanza della classe ModuleCallbacks per la gestione delle chiamate asincrone.
La libreria SailTrackModule si occupa della gestione delle connessioni Wi-Fi dei vari moduli, gestisce la connessione con il broker MQTT e verifica la pubblicazione e ricezione dei messaggi MQTT.
È necessaria l'iscrizione al topic "sensor/strain0/calibration", che permette di ricevere i dati per l'esecuzione della calibrazione.

3.4.2 Gestione chiamate asincrone

Per la gestione delle chiamate a funzioni asincrone si usa un'estensione della classe SailtrackModuleCallbacks (sviluppata da un mio collega), che tramite l'uso di processi asincroni presenti nella libreria di Arduino, permette di ricevere i dati necessari per l'avvio della procedura di calibrazione e conoscere lo stato della batteria del modulo SailTrack Strain.

```
class ModuleCallbacks: public SailtrackModuleCallbacks {
   void onStatusPublish(JsonObject status) {
        JsonObject battery = status.createNestedObject("battery");
        float avg = 0;
        for (int i = 0; i < BATTERY_NUM_READINGS; i++) {

          avg += analogRead(BATTERY_ADC_PIN) / BATTERY_NUM_READINGS;

          delay(BATTERY_READING_DELAY_MS);
     }
   battery["voltage"] = 2 * avg / BATTERY_ADC_RESOLUTION *
   BATTERY_ESP32_REF_VOLTAGE * BATTERY_ADC_REF_VOLTAGE;
     }
   void onMqttMessage(const char * topic, JsonObjectConst message) {
         calibrationLoad=message["calibrationLoad"];
         calibrationScaleDelay=message["calibrationScaleDelay"]?
         message["calibrationScaleDelay"]:CAL_SCALE_DELAY_S;
         calibration=true;
     }
};
```

La funzione onStatusPublish viene chiamata asincronamente dalla libreria SailTrackModule per conoscere lo stato della batteria di SailTrack Strain; una volta calcolato, il valore della batteria viene aggiunto al Json contenente tutte le informazioni di controllo dell'hardware del modulo SailTrack Strain. La funzione è necessaria perché i sensori utilizzano batterie diverse tra loro; in questo modo la libreria SailTrackModule riesce autonomamente a reperire le informazioni necessarie per ogni tipologia di batteria presente nel sistema SailTrack.

La procedura di calibrazione viene avviata attraverso l'invio di un messaggio MQTT usando il software MQTT Explorer. Questo software è utilizzato per l'invio dei messaggi di calibrazione attraverso protocollo MQTT, si occupa della visualizzazione dei topic presenti nella rete di SailTrack Core con un'intuitiva interfaccia grafica e permette la creazione di grafici che aiutano l'utilizzatore nella comprensione dei dati raccolti.

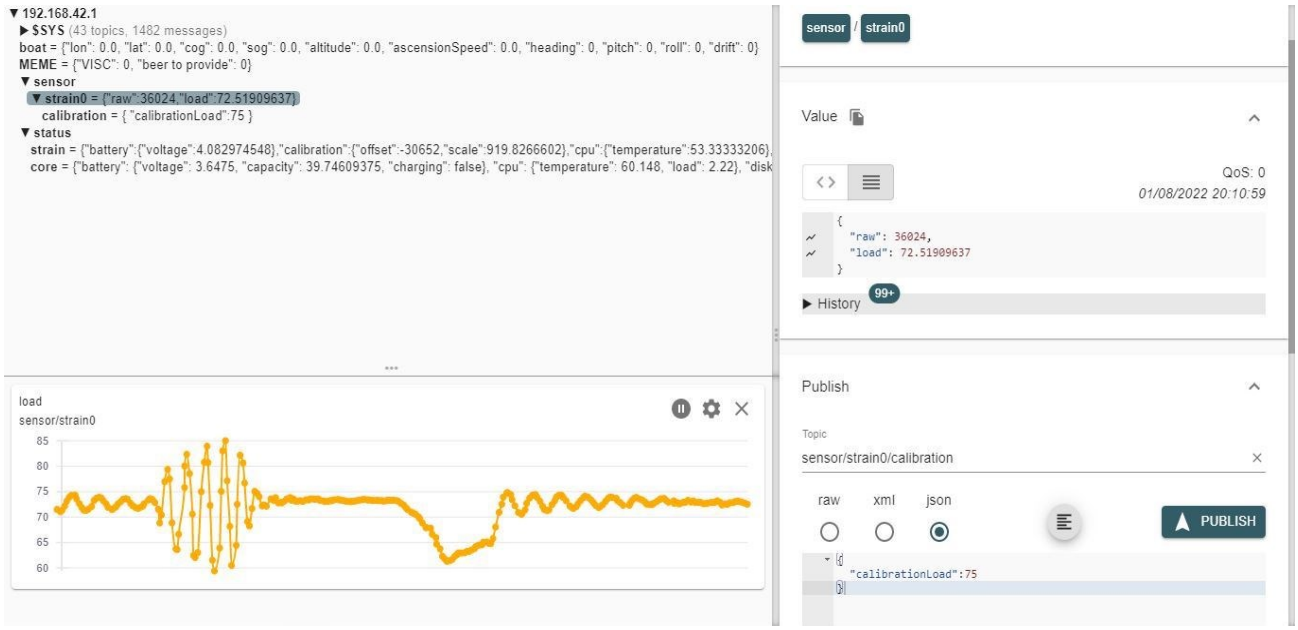

Figura 3.8: interfaccia grafica del software MQTT Explorer.

L'invio del messaggio MQTT causa la chiamata asincrona alla funzione onMqttMessage, che salva i parametri per la chiamata alla funzione di calibrazione, impostando a "vero" la variabile calibration, che servirà per iniziare la procedura di calibrazione. Nel messaggio Json inviato, l'utilizzatore della cella di carico deve inserire obbligatoriamente il carico e, opzionalmente, il tempo necessario per attaccare la cella alla sartia. Se nessun valore di tempo è specificato, si utilizzerà il valore di default.

Possiamo riferirci ai processi asincroni con il nome di callback.

La **callback** [31] è una funzione o un "blocco di codice" che viene passata come parametro ad un'altra funzione. Questo permette alla funzione chiamante di realizzare un compito specifico (quello svolto dalla callback) che è noto solo a tempo di esecuzione del codice.

Le funzioni asincrone onStatusPublish e onMqttMessage sono delle callback perché vengono incapsulate dalla classe ModuleCallbacks nel metodo stm.begin() della funzione setup() del codice sviluppato; le due funzioni vengono chiamate asincronamente dalla libreria SailTrackModule per quanto riguarda la funzione onStatusPublish, per conoscere lo stato della batteria del modulo, mentre la funzione onMqttMessage viene chiamata quando un nuovo messaggio MQTT viene pubblicato nel topic definito dal parametro "const char * topic".

Le chiamate asincrone eseguite rappresentano un esempio di **Command Pattern**.
Definito nella Gang of Four [32] il Command Pattern [33] rappresenta un paradigma per il disaccoppiamento dell'esecuzione di una determinata porzione di codice, dal relativo codice che ne ha richiesto l'esecuzione. Questo permette di rendere variabile l'azione del client senza però conoscere i dettagli dell'operazione stessa. Un altro aspetto rivelante è che colui che richiede l'esecuzione di una determinata azione non è noto all'atto dell'istanziazione del Command, ma solo a tempo di esecuzione.

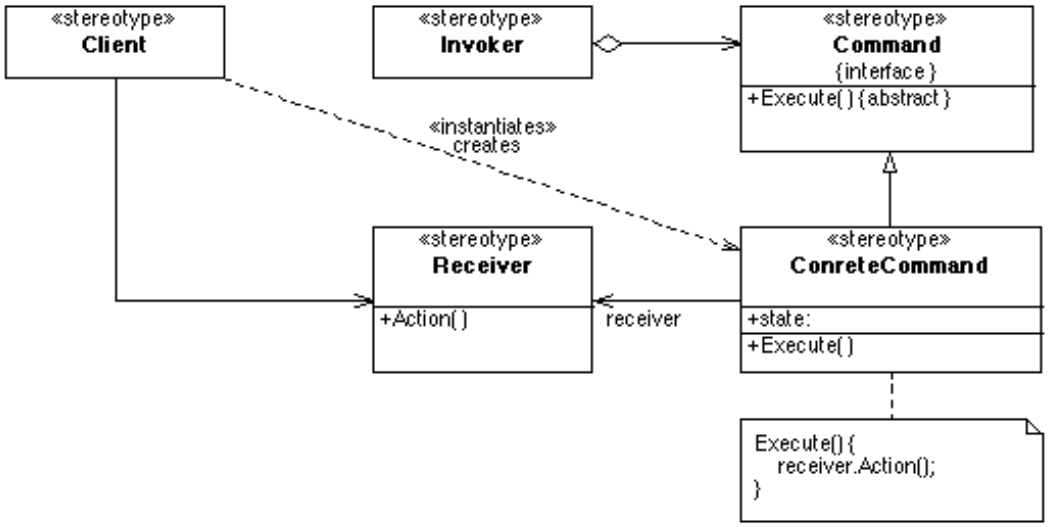

Figura 3.9: schema del Command Pattern. Fonte: https://it.wikipedia.org/wiki/Command_pattern

3.4.3 Implementazione funzione di calibrazione della cella di carico

La calibrazione della cella di carico è necessaria per definire lo stato in cui lavorerà; si effettua calcolando la tara (o offset) e il divider (o scale).

Il comportamento della cella di carico è dettato dalla legge lineare $y = mx + q$: q rappresenta l'offset, x è il carico espresso in chilogrammi misurato dalla cella di carico, m è il divider che si occupa di convertire il valore grezzo in valore espresso in chilogrammi e y rappresenta il valore grezzo associato al carico x.

```
bool calibrate(float calLoad, int calScaleDelay){
    //if zero return false
    if(!calLoad) return false;
    //set tare
    for(int i=0;i<(1000*CAL_TARE_DELAY_S)/(2*CAL_TARE_LED_DELAY_MS);i++){

        digitalWrite(STM_NOTIFICATION_LED_PIN, LOW);
        delay(CAL_TARE_LED_DELAY_MS);
        digitalWrite(STM_NOTIFICATION_LED_PIN, HIGH);
        delay(CAL_TARE_LED_DELAY_MS);
    }
    hx.tare(CAL_NUM_READINGS);
    //set scale
    for(int i=0;i<(1000*calScaleDelay)/(2*CAL_SCALE_LED_DELAY_MS);i++){
        digitalWrite(STM_NOTIFICATION_LED_PIN, LOW);
        delay(CAL_SCALE_LED_DELAY_MS);
        digitalWrite(STM_NOTIFICATION_LED_PIN, HIGH);
        delay(CAL_SCALE_LED_DELAY_MS);
    }
    double reading=hx.get_value(CAL_NUM_READINGS);
    //set new tare
```

```
        hx.set_scale(reading/calLoad);
        return saveCalibration();
}
```

Una volta ricevuto il messaggio MQTT contenente il carico della cella di carico e il tempo per attaccarla alla sartia, viene chiamata la funzione che svolge la calibrazione.

La calibrazione si compone della fase di taratura in cui bisogna lasciare la cella di carico a riposo, con carico pari a zero. Funziona come una normale bilancia: prima di appoggiare l'oggetto da posare, si effettua l'operazione di taratura. Una volta calcolata la tara, si imposta tale valore nell'amplificatore tramite la funzione hx.tare. Quanto calcolato è l'offset di misura dal valore che utilizzeremo per il calcolo del divider.

Il divider viene calcolato, una volta attaccata la cella di carico alla sartia nel tempo stabilito, dalla funzione hx.set_scale, effettuando la divisione tra la lettura puntuale dei dati della cella di carico a cui viene sottratto l'offset e il carico definito dal software MQTT Explorer.
La tara e il divider vengono salvati nella memoria EEPROM, per un futuro utilizzo.

3.4.4 Salvataggio e lettura dei dati di calibrazione

Una volta terminata l'esecuzione del programma di gestione della cella di carico, i dati di offset e divider verrebbero persi se non venissero salvati permanentemente su una memoria di archiviazione di massa. Grazie alla memoria EEPROM [34] presente sul microcontrollore ESP-32 è possibile evitare questo problema.
La procedura di salvataggio in memoria si compone di due fasi: la prima è il salvataggio di offset e divider su due indirizzi noti della memoria, la seconda si occupa del calcolo e del successivo accodamento del checksum ai dati inseriti in memoria.

Il calcolo del checksum avviene attraverso l'algoritmo crc16.

```
uint16_t crc16Update(uint16_t crc, uint8_t a) {
  int i;
        //logic xor
  crc ^= a;
  for (i = 0; i < 8; i++) {
    if (crc & 1) crc = (crc >> 1) ^ 0xA001;
    else crc = (crc >> 1);
  }
  return crc;
}
```

Il **cyclic redundancy check** [35] **a 16 bit** (CRC) è un algoritmo per il calcolo del checksum, che sfrutta un registro a scorrimento di 16 bit.
Il checksum è una procedura che permette di verificare l'integrità dei dati trasmessi in un mezzo di comunicazione. Il checksum viene usato per il controllo di alterazioni di messaggi causate dal rumore di trasmissione o per altri fattori.

Il calcolo del checksum è svolto attraverso la divisione incrementale tra i dati, accodati da W bit uguali a zero e un polinomio di W bit; W è il grado del polinomio divisore.

Le operazioni sono senza riporto, 1+1=0: questo permette il calcolo di operazioni di somme e sottrazioni attraverso la funziona logica XOR, dove l'uscita di due ingressi è uno solo se i due ingressi hanno valori diversi.

I bit risultanti dall'operazione incrementale di divisione compongono la CRC.
La CRC viene accodata al messaggio originale che verrà trasmesso al destinatario.
Il ricevitore può sia ricalcolare la CRC e confrontarla con quello ricevuta, o più semplicemente calcolare la CRC su tutto il messaggio ricevuto. Se non si sono verificati errori, il risultato sarà uguale a zero.

```
bool saveCalibration() {

        uint8_t buf[EEPROM_CAL_SIZE_BYTES];
        //set to zero EEPROM_CAL_SIZE_BYTES pointed by buf
        memset(buf, 0, EEPROM_CAL_SIZE_BYTES);
        buf[0] = EEPROM_CAL_ID_0;
        buf[1] = EEPROM_CAL_ID_1;

        long offset=hx.get_offset();
        float scale=hx.get_scale();
        memcpy(buf+2,&offset,sizeof(offset));
        memcpy(buf+2+sizeof(offset),&scale,sizeof(scale));

        uint16_t crc = 0xFFFF;
        for (uint16_t i = 0; i < EEPROM_CAL_SIZE_BYTES - 2; i++)
                crc = crc16Update(crc, buf[i]);
        buf[EEPROM_CAL_SIZE_BYTES - 2] = crc & 0xFF;
        buf[EEPROM_CAL_SIZE_BYTES - 1] = crc >> 8;
        for (uint16_t a = 0; a < EEPROM_CAL_SIZE_BYTES; a++)
                EEPROM.write(a + EEPROM_CAL_ADDR, buf[a]);

        EEPROM.commit();
        return true;
}
```

Una volta definiti gli indirizzi di salvataggio di offset e divider, si procede con il loro salvataggio, prendendo i valori dell'amplificatore, attraverso le funzioni di sistema hx.get_offset e hx.get_scale.

Si procede con il calcolo e accodamento della CRC nella memoria EEPROM.
La CRC non viene usata direttamente dalla funzione saveCalibration, ma sarà utile per verificare l'integrità dei dati quando verranno letti dalla memoria di massa.

Per capire quanti chilogrammi sta misurando la cella di carico è necessario partire dai dati grezzi prodotti, e tramite la lettura di offset e divider presenti in memora EEPROM, calcolare il carico a cui è sottoposta.
Quando viene avviato il software che controlla la cella di carico, nella funzione *setup()* viene svolta la lettura dei dati di offset e divider dalla memoria EEPROM.

Anche la procedura di lettura dati dalla memoria fa uso dell'algoritmo di CRC.

```
bool loadCalibration() {
	uint8_t buf[EEPROM_CAL_SIZE_BYTES];

	uint16_t crc = 0xFFFF;
	for (uint16_t a = 0; a < EEPROM_CAL_SIZE_BYTES; a++) {
		buf[a] = EEPROM.read(a + EEPROM_CAL_ADDR);
		crc = crc16Update(crc, buf[a]);
	}
	//if the are data corrupted crc is not zero.
	if (crc != 0 || buf[0] != EEPROM_CAL_ID_0 || buf[1] != EEPROM_CAL_ID_1)
		return false;

	long offset;
	//divider=scale
	float scale;
	memcpy(&offset, buf + 2, sizeof(offset));
	memcpy(&scale,buf+2+sizeof(offset),sizeof(scale));
	hx.set_offset(offset);
	hx.set_scale(scale);

	return true;
}
```

Prima di caricare la calibrazione sull'esecuzione corrente del programma, si effettua il controllo tramite CRC per verificare se i dati presenti in memoria siano effettivamente validi; si controlla che i dati presenti in memoria sono quelli caricati in precedenza da noi.

Questo viene svolto dall'algoritmo di CRC che calcola il checksum con offset-divider letti dalla memoria e checksum(offset-divider in memoria): se il risultato dell'operazione di CRC è zero significa che i dati in memoria non sono stati modificati.

Una volta controllato che i dati in memoria non sono stati corrotti, si procede con il salvataggio dei valori di offset e divider in memoria RAM del microcontrollore.

3.4.5 Esecuzione del programma

Il flusso del programma alterna due fasi: la fase di lettura dei dati misurati dalla cella di carico e la fase di calibrazione. Queste due fasi sono eseguite nella funzione loop:

```
void loop() {

	if(calibration){
		if(calibrate(calibrationLoad,calibrationScaleDelay)){
			calibration=false;
			calibrationTries=CAL_NUM_TRIES;
		}
		else
			{
				for(int i=0;i<CAL_ERROR_BLINKING_TIMES;i++){
```

```
                        digitalWrite(STM_NOTIFICATION_LED_PIN, LOW);
                        delay(CAL_ERROR_LED_DELAY_MS);
                        digitalWrite(STM_NOTIFICATION_LED_PIN, HIGH);
                        delay(CAL_ERROR_LED_DELAY_MS);
                    }
                    if(!calibrationTries--){
                        calibration=false;
                        calibrationTries=CAL_NUM_TRIES;
                    }
                    delay(CAL_TRIES_DELAY_MS);
                }
        }
        else
            readData();

}
```

Come già discusso nella descrizione della funzione *calibrate*, la procedura di calibrazione viene avviata dall'invio del messaggio MQTT sullo specifico topic "sensor/strain0/calibration", che causa la chiamata asincrona alla funzione onMqttMessage.
La funzione onMqttMessage imposta a "vero" la variabile calibration, che permette di avviare la calibrazione della cella di carico. Se ci sono problemi nella routine di calibrazione, che causano un'interruzione anomala nella procedura di calibrazione, si riprova l'esecuzione della calibrazione per un numero fissato di volte; se non va a buon fine, l'utilizzatore deve inviare di nuovo il comando di calibrazione via MQTT.
Quando la procedura di calibrazione è terminata, si procede con la lettura dei dati dalla cella di carico.

```
void readData(){
        TickType_t lastWakeTime = xTaskGetTickCount();
        StaticJsonDocument<STM_JSON_DOCUMENT_MEDIUM_SIZE> doc;
        doc["raw"]=hx.read_average(HX711_NUM_READING);
        doc["load"]=hx.get_units(HX711_NUM_READING);
        stm.publish("sensor/strain0", doc.as<JsonObjectConst>());
        vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(LOOP_TASK_INTERVAL_MS));
}
```

La funzione *readData* si occupa di prendere i dati grezzi prodotti dalla cella di carico, per trasformarli, usando offset e divider, in dati espressi in chilogrammi, che permettono di capire quanto è il carico a cui è sottoposta la sartia.
I dati grezzi e quelli in chilogrammi vanno a formare il Json, che andrà a comporre il messaggio MQTT che verrà pubblicato nel topic "sensor/strain0": i dati presenti nel Json verranno salvati nel database di SailTrack Core.

Capitolo 4

Utilizzo

Prima di illustrare il corretto utilizzo della cella di carico si introduce l'attrezzatura necessaria per effettuare le operazioni di misura.
Nella seconda parte verrà dettagliata la procedura di uso della cella di carico.

Tensiometro

Per l'utilizzo della cella di carico è necessario introdurre il tensiometro.
Il tensiometro è uno strumento di misura, che permette di stabilire a che carico sono sottoposte le sartie dell'imbarcazione.

Poiché lo scopo delle sartie è quello di dare stabilità all'albero, è necessario che siano il più rigide possibili, per sostenere nel miglior modo il suo peso.

Il tensiometro, conoscendo il diametro di sartie utilizzato, definisce il valore di carico a cui le sartie sono sottoposte.
Dato che il tensiometro, in base ai test effettuati nel tempo dai membri dell'equipaggio che si occupano di rendere rigide le sartie, presenta una variabilità di errore nella misura del carico applicato, la squadra di Elettronica ha deciso di sviluppare SailTrack Strain, per verificare se quanto misurato dal tensiometro sia veritiero o meno. Il sensore SailTrack Strain non va a sostituire il lavoro del tensiometro, ma va a completarlo, definendo la bontà delle letture effettuate.

Uso della cella di carico

La procedura di utilizzo della cella di carico inizia attaccando il tensiometro ad una delle sartie dell'imbarcazione.

Figura 4.1: tensiometro attaccato alla sartia dell'imbarcazione per verificarne il carico.

Il tensiometro viene attaccato a circa un metro dall'inizio della sartia dal punto di ancoraggio della barca.
Una volta applicato il tensiometro, la scala graduata di **Figura 4.2** ci dice in che valore ci troviamo.

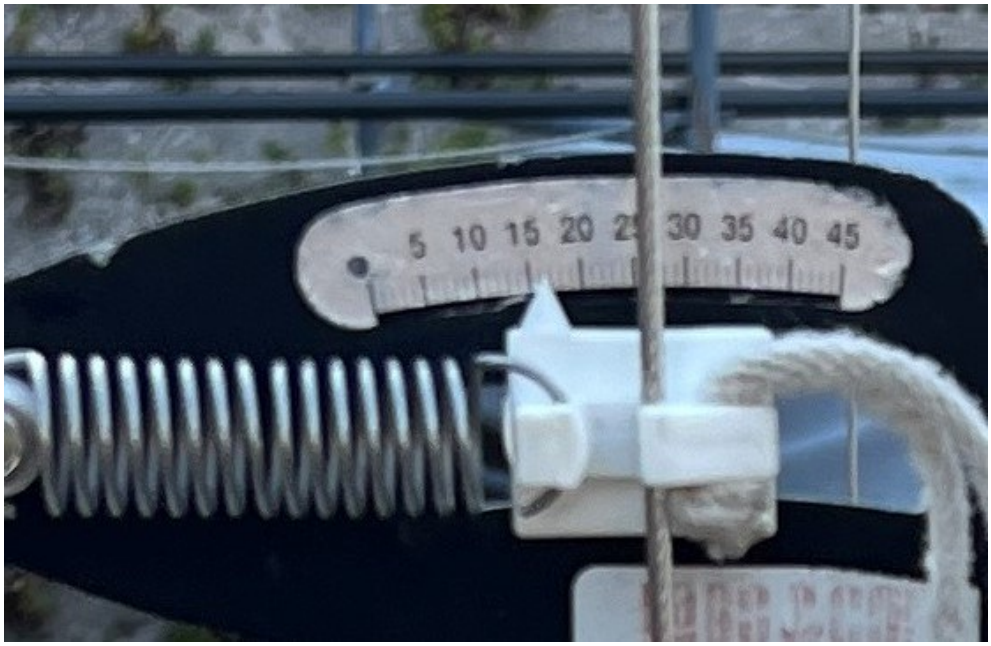

Figura 4.2: scala graduata del tensiometro.

Il valore misurato (in questo esempio) è pari a 16.

Conoscendo il diametro della sartia è possibile risalire al peso ad essa applicato.
Ora dobbiamo controllare la tabella nella **Figura 4.3**: in verticale sono espressi i valori di scala da 0 a 45, mentre in orizzontale il diametro delle sartie.
Ricordando che stiamo lavorando con sartie di diametro pari a 3 millimetri, leggendo la seconda colonna che contiene il carico delle sartie con tale diametro, si trova che alla sartia in questione è applicato un carico pari a 75 chili.

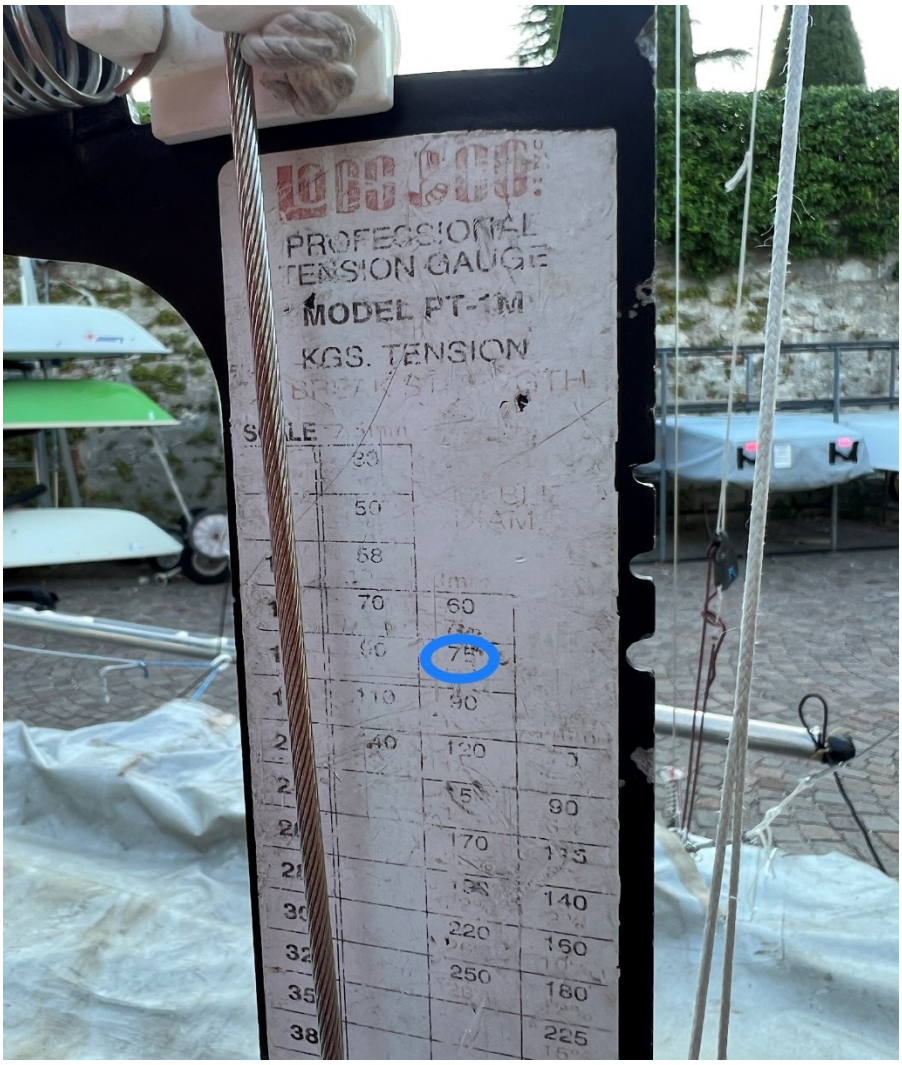

Figura 4.3: 75 chili di carico applicato alla sartia in esame.

Per avere un confronto con la misurazione del tensiometro, iniziamo la procedura di calibrazione della cella di carico, inviando il messaggio MQTT attraverso il software MQTT Explorer, specificando il carico di 75 chili misurato dal tensiometro.

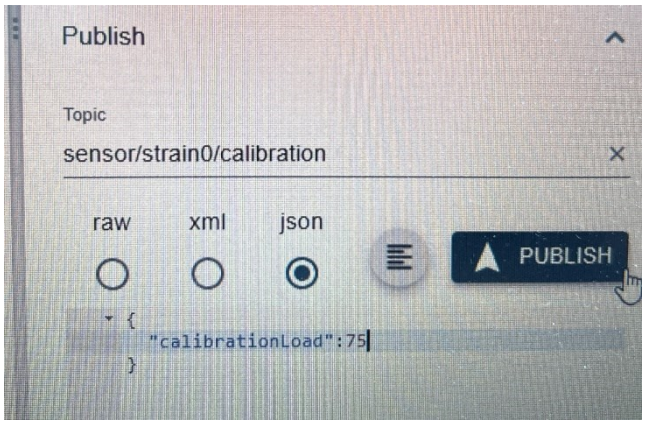

Figura 4.4: invio del messaggio MQTT attraverso il software MQTT Explorer, sul topic "sensor/strain0/calibration", per iniziare la calibrazione.

Avviata la calibrazione, il led del microcontrollore ESP-32 lampeggerà per un tempo predeterminato ad una velocità predefinita, segnalando che è necessario lasciare la cella di carico a riposo, ovvero scollegata da qualsiasi sartia, per il calcolo dell'offset.

Una volta terminata la fase di taratura, una seconda tipologia di lampeggio con velocità diversa dal precedente segnala che è necessario attaccare la cella di carico alla sartia (nella stessa posizione in cui era stato attaccato il tensiometro), entro il tempo massimo stabilito.

Terminata la fase di calibrazione, si entra nella fase di lettura dati.

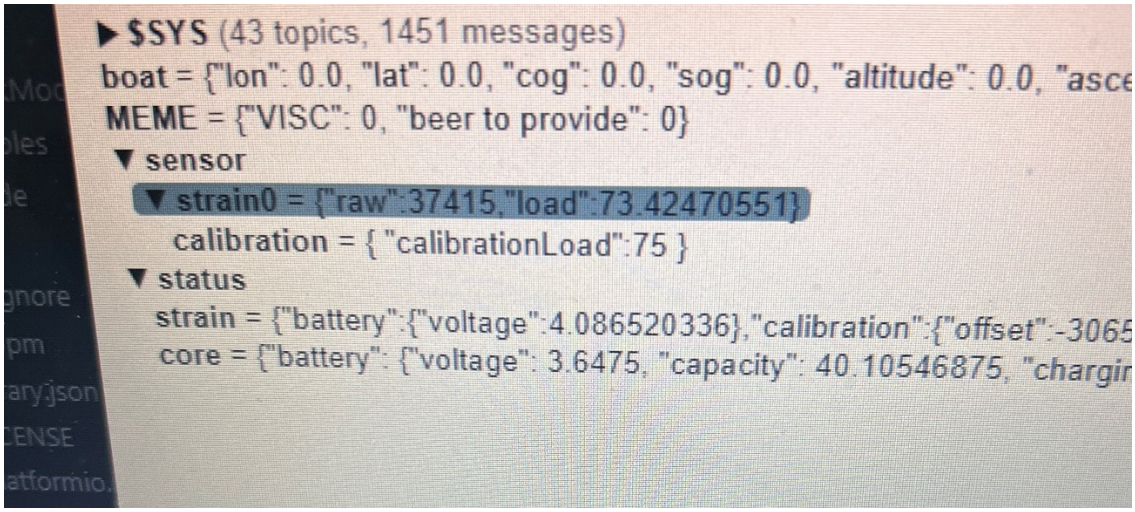

Figura 4.5: lettura in tempo reale, dei valori grezzi della cella di carico e del carico espresso in chilogrammi.

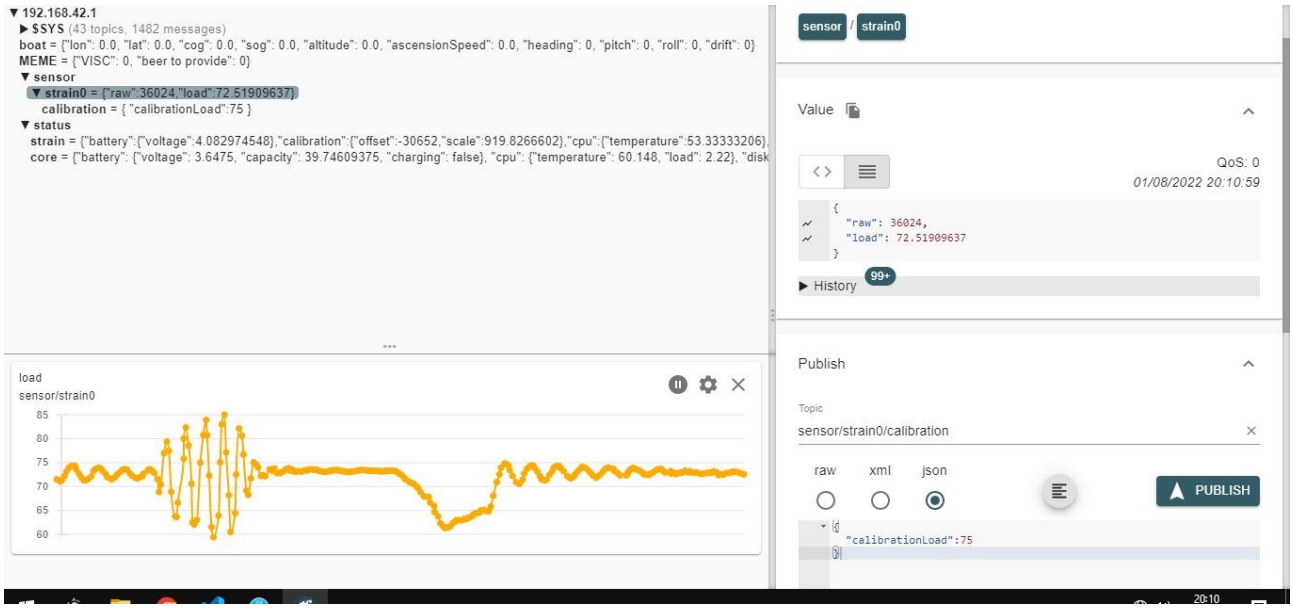

Figura 4.6: software MQTT Explorer che mostra la variazione di carico applicato alla sartia, tramite l'ausilio del grafico in arancione.

In modalità lettura dati il team, leggendo i dati inviati in tempo reale a terra, vede se è presente vento molto forte (le sartie colpite direttamente dal vento saranno sottoposte ad un carico maggiore), se la barca ha "scuffiato" ovvero se si è ribaltata o altre andature della barca.

Per visualizzare i dati raccolti dai vari sensori attraverso il software MQTT Explorer, è necessario che quest'ultimo sia un client della rete di SailTrack Core; bisogna perciò assegnare al client MQTT Explorer, in fase di configurazione dello stesso, un indirizzo IP della rete di SailTrack Core.

Capitolo 5

Conclusioni

Dalla **figura 4.6**, le variazioni repentine di carico rappresentano la simulazione di movimento dell'imbarcazione, dovuta ad un vento fittizio, effettuata muovendo lateralmente il carello su cui era posta la barca.
Le fine del grafico non presenta variazioni significative del carico, indicando che la sartia a cui è applicata la cella di carico è sottoposta ad un carico quasi stazionario.

Rispetto ai 75 chilogrammi di carico misurati dal tensiometro, la cella rileva che il carico a cui è sottoposta nelle stesse condizioni del tensiometro è di 73.42 chilogrammi. La differenza misurata è molto ridotta rispetto al carico totale a cui è sottoposta la sartia: 1.58 su 75 chilogrammi genera un errore di circa 2.10% sulla misura, risultando ininfluente l'errore prodotto dal tensiometro.

Considerando ininfluente l'errore singolo di misura, è importante notare che un errore di 1.58 chilogrammi per ogni sartia, causa un errore totale di misura, utilizzando 6 sartie, pari a 9.48 chilogrammi.

L'equipaggio è rimasto positivamente soddisfatto dell'utilità del sensore SailTrack Strain perché permette di verificare se il carico misurato dal tensiometro è veritiero, e nel caso non lo fosse, di apportare delle modifiche nel tiraggio delle sartie, per avere una maggior stabilità dell'albero.

Figura 5.1: il sensore SailTrack Strain completo.

BIBLIOGRAFIA

[1] Maspannelli, “IL MULTISTRATO MARINO”. URL: https://www.maspannelli.com/materiali/multistrato-marino.html

[2] Marta Abbà, “Resine termoindurenti e termoplastiche”. URL: https://www.ideegreen.it/resine-termoindurenti-110328.html

[3] Sito web Mètis, http://metisvela.dii.unipd.it/

[4] Pagina Instagram Mètis, URL: https://www.instagram.com/metis_vela_unipd/

[5] Pagina Facebook Mètis, URL: https://www.facebook.com/metisvelaunipd/

[6] Duckma, “MQTT: ecco cos'è e come funziona il protocollo di comunicazione IoT”. URL: https://blog.duckma.com/mqtt-ecco-come-funziona-il-protocollo-di-comunicazione-iot/

[7] Guido Tassinari, “ISO/OSI Vs. TCP/IP”. URL: https://www.isistassinari.edu.it/progettodicembre/reti/TCP-IP.html

[8] Università degli studi di Trieste, “Simulazione di una comunicazione fra

dispositivi che utilizzano il protocollo i2c”. URL: http://www2.units.it/marsi/elettronica2/tesine_elettro/i2c/i2c.pdf

[9] Emanuele Genovese, “Resistenza di pull-up / pull-down”. URL: https://www.emanuelegenovese.it/resistenza-pull-up-pull-down/

[10] Wikipedia, l'enciclopedia libera, “Sistema real-time”. URL: https://it.wikipedia.org/wiki/Sistema_real-time

[11] freeRTOS, “What is FreeRTOS?”. URL: https://www.freertos.org/about-RTOS.html

[12] geeksforgeeks, “Microkernel in Operating Systems” URL: https://www.geeksforgeeks.org/microkernel-in-operating-systems/

[13] “influxDB”, URL: https://www.influxdata.com/

[14] “Grafana”, URL: https://grafana.com/

[15] Wikipedia, l'enciclopedia libera, “Single board computer”. URL: https://it.wikipedia.org/wiki/Single-board_computer

[16] Raspberry Pi, “Raspberry Pi 3 Model B+”. URL: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

[17] Henryk Pepliński, “Ship and Mobile Offshore Unit Automation”. URL: https://www.sciencedirect.com/topics/engineering/power-management-system

[18] DietPi. URL: https://dietpi.com/

[19] Wikipedia, l'enciclopedia libera, “Inertial measurement unit”. URL: https://en.wikipedia.org/wiki/Inertial_measurement_unit

[20] “LILYGO® TTGO T7 Mini32 V1.5 ESP32-WROVER-B Circuito senza fili CH9102 di sviluppo del modulo di Bluetooth di Wi-Fi del centro PSRAM di TTGO T7 Mini32 V1.5 ESP32-WROVER-B”. URL: https://it.aliexpress.com/item/32977375539.html

[21] SailTrack filter, URL: https://github.com/metis-vela-unipd/sailtrack-core/blob/main/sailtrack/sailtrack-filter

[22] Wikipedia, l'enciclopedia libera, “Carta elettronica” . URL: https://it.wikipedia.org/wiki/Carta_elettronica

[23] Farnell, “LoRa”, URL: https://it.farnell.com/wireless-lora-technology

[24] Redhat, “Che cos'è l'Internet of Things (IoT)?”. URL: https://www.redhat.com/it/topics/internet-of-things/what-is-iot

[25] Wikipedia, l'enciclopedia libera, “Cella di carico”. URL: https://it.wikipedia.org/wiki/Cella_di_carico

[26] Wikipedia, l'enciclopedia libera, “Estensimetro. URL: https://it.wikipedia.org/wiki/Estensimetro

[27] Wikipedia, l'enciclopedia libera, “Ponte di Wheatstone”. URL: https://it.wikipedia.org/wiki/Ponte_di_Wheatstone

[28] Università degli studi di Napoli Federico secondo, “Amplificatore differenziale” URL: https://www.docenti.unina.it/webdocenti-be/allegati/materiale-didattico/34215699

[29] Wikipedia, l'enciclopedia libera, “Convertitore analogico-digitale, URL: https://it.wikipedia.org/wiki/Convertitore_analogico-digitale

[30] Cecchi, “Le nostre Resine Epossidiche”. URL: https://www.cecchi.it/resine-epossidiche/

[31] Wikipedia, l'enciclopedia libera, “Callback”. URL: https://it.wikipedia.org/wiki/Callback

[32] Erich Gamma, John Vlissides, Richard Helm e Ralph Johnson “Gang of Four”. URL: https://it.wikipedia.org/wiki/Design_Patterns

[33] “Command pattern”. URL: https://it.wikipedia.org/wiki/Command_pattern

[34] Wikipedia, l'enciclopedia libera, “EEPROM”. URL: https://it.wikipedia.org/wiki/EEPROM

[35] Stefano Lovati, “Cyclic Redundancy Check (CRC) – Concetti basilari”. URL: https://it.emcelettronica.com/cyclic-redundancy-check-crc-concetti-basilari

GitHub Mètis Vela. URL: https://github.com/metis-vela-unipd