INTRODUZIONE 3

1. Cenni teorici 7
1.1 Imagenex852 ultra miniature echosounder 7
1.2 Generazione di immagini SONAR 10
1.3 Il filtro di Kalman 12
1.4 Sistema di riferimento 14
1.5 La trasformata di Hough 16

2. I veicoli 18
2.1 FeelHippo 18
2.1.1 La competizione Sauc-e 2013 18
2.1.2 Breve descrizione del veicolo 21
2.1.3 Obiettivi e specifiche 23
2.2 Nemo Rov 25
2.2.1 Il progetto Nemo 25
2.2.2 Breve descrizione del veicolo 26
2.2.3 Obiettivi e specifiche 31
2.3 Turtle 32
2.3.1 Breve descrizione del veicolo 32
2.3.2 Eredità tecnologica del progetto 35

3. Ambiente di sviluppo del software 45
3.1 Ambiente di sviluppo 37
3.2 Middleware 37
3.2.1 Ros (Robot Operating System) 38
3.2.2 MOOS ivp 40
3.3 Framework e librerie 43

3.3.1 Il framework Qt 43
3.3.2 Le librerie OpenCv 44
3.4 Sistema operativo 44

4. Algoritmo di stima della posizione 45
4.1 Elenco ed analisi delle ipotesi 45
4.2 Algoritmo di localizzazione 46
4.2.1 Ricezione dell'eco e stima della direzione del beam .. 47
4.2.2 Misura della distanza parete-veicolo 47
4.2.3 Filtro euristico 49
4.2.4 Aggiornamento buffer dati 50
4.2.5 Misura della posizione 51
4.2.5.1 Metodo geometrico 51
4.2.5.2 Metodo basato sulla trasformata di Hough 52
4.2.6 Filtraggio ricorsivo 53

5. Realizzazione del software 56
5.1 Analisi degli obiettivi 56
5.2 Implementazione delle classi 58
5.2.1 Classe SerialHandler 58
5.2.2 ClasseTcpHandler 61
5.2.3 Classe MoosHandler 64
5.2.4 Classe RosHandler 65
5.2.5 Classe InterfaceHandler 67
5.3 Caratteristiche di alto livello del software 69
5.3.1 Schema UML Nemo 69
5.3.2 Schema ROS di FeelHippo 72
5.4 Graphical user interface (GUI) 73
5.4.1 Aspetti software 73
5.4.2 Dati di interesse 74
5.4.3 Descrizione della GUI 74
5.4.4 Echosounder virtuale 75

6. Risultati sperimentali 78
6.1 Generazione di immagini acustiche (caso ROV)... 80
6.1.2 Stima della posizione del veicolo..... 81

CONCLUSIONI 83

SVILUPPI FUTURI 85

DESCRIZIONE DEI DISPOSITIVI IMPIEGATI 86

Introduzione

Questo lavoro di tesi è stato condotto presso il Laboratorio di Modellazione Dinamica e Meccatronica del Dipartimento di Ingegneria Industriale dell'Università degli Studi di Firenze nell'ambito di una collaborazione con il Dipartimento di Ingegneria dell'Informazione della stessa Università. In particolare l'attività è stata svolta nell'ambito della competizione SAUC-e a cui ho partecipato con l'UNIFI team nella edizione 2012 ed a cui parteciperò nel 2013. La competizione si svolge a La Spezia presso l'ex NATO Undersea Research Centre (NURC), ora Centre for Maritime Research and Experimentation (CMRE) ed il suo scopo è lo sviluppo di tecnologie nell'ambito dei veicoli sottomarini autonomi AUV (Authonomous Underwater Vehicle). Ogni team partecipante deve realizzare un veicolo che sia in grado di eseguire autonomamente i task indicati dal regolamento; inoltre deve presentarlo di fronte ad una giuria di esperti spiegando le scelte progettuali effettuate. A questo scopo un team di circa 15 persone lavora ai vari aspetti del veicolo che vanno dalla meccanica all'elettronica al software; in questo lavoro di tesi verrà trattato il sistema di localizzazione del veicolo che verrà impiegato durante la competizione 2013.

Il sistema di localizzazione è basato sull'impiego dell'Imagenex 852 ultra miniature echosounder: si tratta di un sensore SONAR (SOund Navigation And Ranging) unidirezionale che generando impulsi sonori e ricevendone l'eco è in grado di rilevare gli ostacoli che si trovano nel suo raggio d'azione (un cono dell'ampiezza di 10°). Trattandosi di un sensore unidirezionale è stato realizzato un sistema meccanico basato su un servomotore come azionamento che permette al sensore di ruotare parallelamente all'asse di imbardata del veicolo (figura 1);di fatto il grado di libertà aggiuntivo trasforma l'echosounder in un mechanically scanned imaging SONAR.

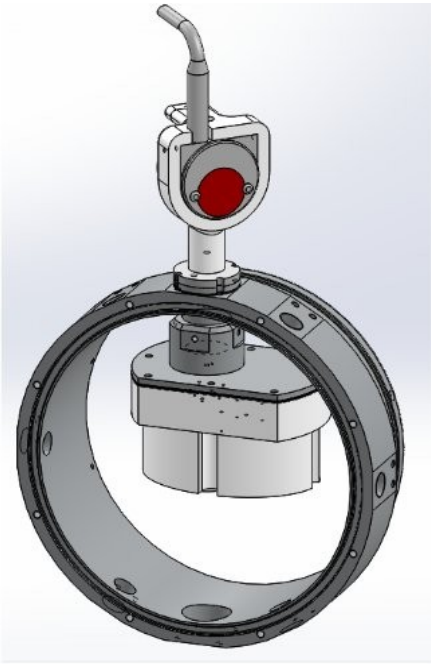

Fig 1: Assieme echosounder-azionamento

In questo lavoro di tesi,inoltre, si riporta lo sviluppo di un software che, tramite l'impiego

del sensore e dell'azionamento, è in grado di generare immagini acustiche che rappresentano l'ambiente circostante. I dati così prodotti possono essere esaminati dal pilota di un veicolo controllato in remoto (ROV : Remoted Operated Vehicle) oppure dal sistema di controllo di un veicolo sottomarino autonomo (AUV : Autonomous Underwater Vehicle) .

Il software realizzato è stato impiegato sui seguenti veicoli:

- Turtle (AUV);
  - Veicolo realizzato per il SAUC-e 2012.
- FeelHippo (AUV);
  - Veicolo realizzato per il SAUC-e 2013.
- Nemo Rov (ROV).
  - Veicolo realizzato per l'esplorazione del relitto della nave Costa Concordia.

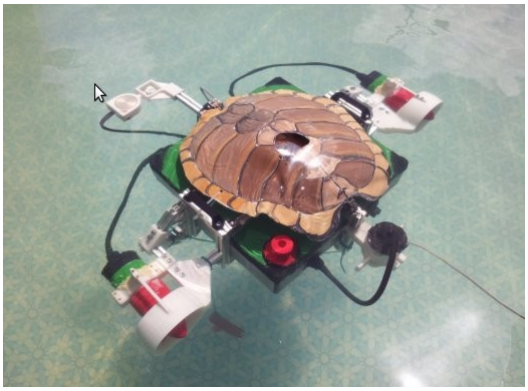

Fig 2: Il veicolo Turtle (AUV)

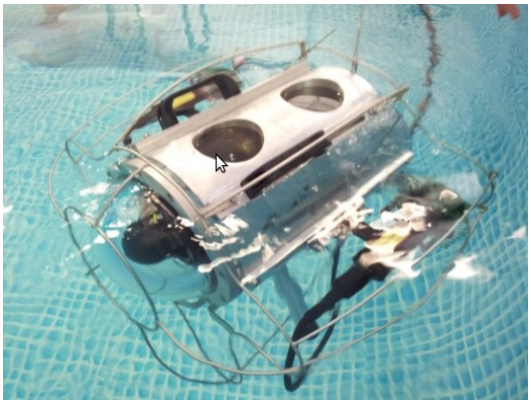

Fig 3: Il veicolo Nemo (ROV)

Fig 4: Il veicolo FeelHippo (AUV) durante la fase di montaggio.

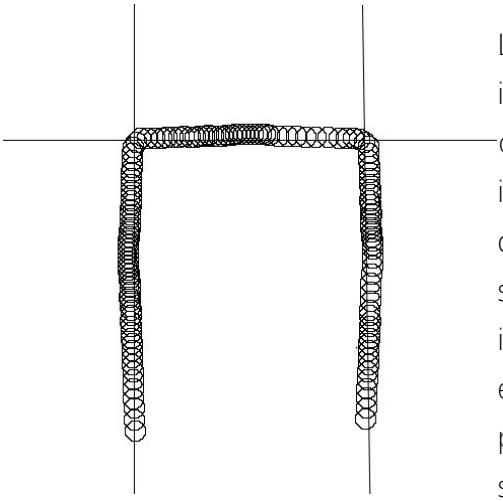

Fig 5: Riconoscimento delle pareti basato sulla trasformata di Hough (sistema impiegato su veicoli AUV)

Le immagini generate, come appena indicato, possono essere impiegate in modi diversi: nel caso di un ROV vengono interpretate dal pilota del veicolo per osservare l'ambiente circostante in situazioni di scarsa visibilità; nel caso invece di un AUV i dati devono essere elaborati da un software al fine di estrarne parametri che siano fruibili da parte di un software.

Il caso dell'AUV è il più complesso: lo scopo è analizzare l'immagine per rilevare la posizione del veicolo all'interno del campo gara. È stato a tale scopo sviluppato un algoritmo che tramite l'impiego della trasformata di Hough e sfruttando la conoscenza a priori del campo gara permette di stimare la posizione del veicolo in modo sufficientemente robusto.

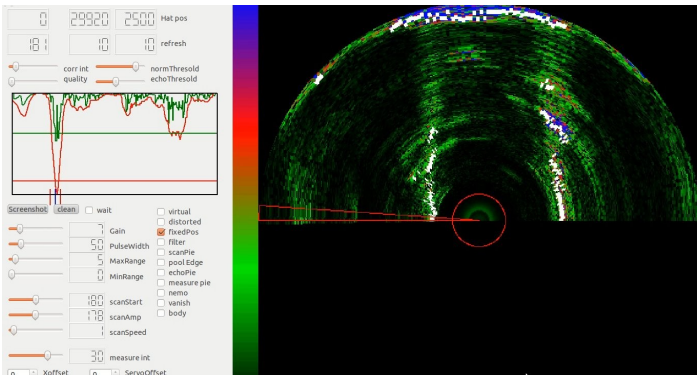

Fig 6: Interfaccia grafica per ROV

La tesi è organizzata in 5 capitoli:

- Primo capitolo (cenni teorici)
  - Vengono affrontati gli aspetti teorici più importanti.
- Secondo capitolo (veicoli)
  - Vengono descritti i veicoli FeelHippo, Nemo e Turtle; la descrizione

è incentrata sul software e sulle differenze tra i vari approcci.

- Terzo capitolo (algoritmo di stima della posizione)
  - In questo capitolo viene descritto l'algoritmo di localizzazione, esaminando nel dettaglio le varie fasi di parsing dei dati.
- Quarto capitolo (ambiente di sviluppo software)
  - In questo capitolo viene descritto l'ambiente di sviluppo che permetterà al software prodotto di interfacciarsi con i processi scritti dagli altri membri del team.
- Quinto capitolo (realizzazione del software)
  - In questo capitolo viene descritto nel dettaglio il software che è stato realizzato per il sistema di localizzazione; in particolare sono descritte le varie classi impiegate: di ognuna si riporta l'header, il relativo UML.
- Appendice (Descrizione dei dispositivi impiegati)
  - In questo capitolo vengono descritti i principali dispositivi impiegati, esaminandone in particolare vantaggi e svantaggi.

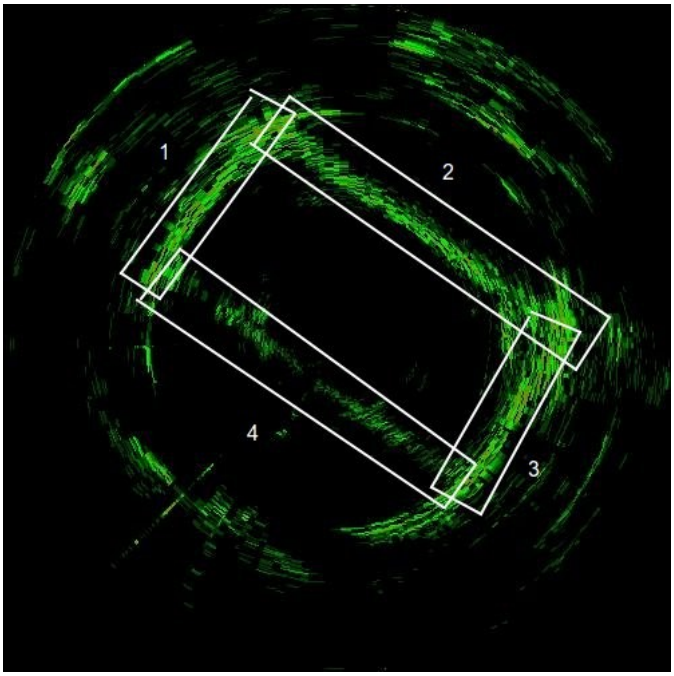

Fig 7: Un esempio di immagine acustica di una piscina rettangolare: sono chiaramente riconoscibili le 4 pareti.

1 Cenni teorici

In questo primo capitolo vengono riportati i fondamentali teorici indispensabili per la comprensione del resto del testo.

1.1 Imagenex 852 Ultra miniature Echosounder

Il termine SONAR (SOund Navigation And Ranging) indica una tecnica di misura che sfrutta le proprietà di propagazione del suono (solitamente in ambiente sottomarino, ma non solo); può essere attivo se sfrutta l'eco degli impulsi emessi dal sonar stesso o passivo nel caso in cui non emetta alcun suono e semplicemente "ascolti" i suoni emessi dall'ambiente circostante.

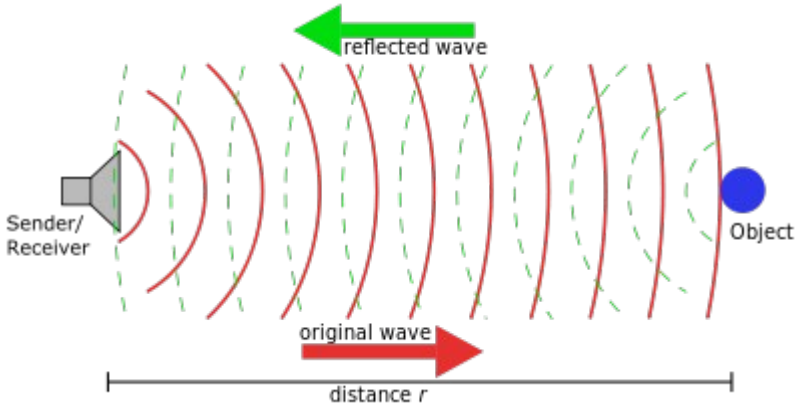

Fig 8: Meccanismo di funzionamento dei sensori SONAR

L'Imagenex 852 è un sensore SONAR attivo dotato di un trasduttore piezoelettrico che viene impiegato sia per generare l'impulso sonoro sia per riceverne l'eco (figura 8). L'impulso generato dal trasduttore piezoelettrico è una sinusoide a 675 kHz di una durata compresa tra 1 e 150 μs , la forma del beam è conica con un'ampiezza di 10°.

L'Imagenex 852 nasce come profondimetro, solitamente impiegato nelle barche da pesca per misurare la distanza dal fondo e rilevare eventuale presenza di fauna marina; tuttavia è sempre più utilizzato a bordo di ROV e AUV in virtù del suo basso costo e ingombro. In questo lavoro viene impiegato per la localizzazione del sottomarino, in particolare per il rilevamento delle pareti di un bacino artificiale e la misura della distanza da

esse.

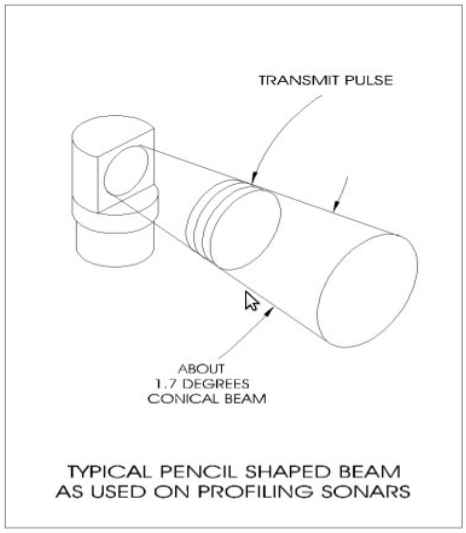

Fig 9: Ricostruzione della forma conica del beam

Fig 10: Foto dell'Imagenex 852 ultra miniature echosounder

.

Considerazioni sul rumore

Il sensore campiona l'eco ricevuto in 250 istanti relativi a 250 distanze, il rumore di misura è dovuto a molteplici cause :

- errore di quantizzazione;
- rumore di misura del trasduttore piezoelettrico;
- rumore intrinseco dell'ambiente.

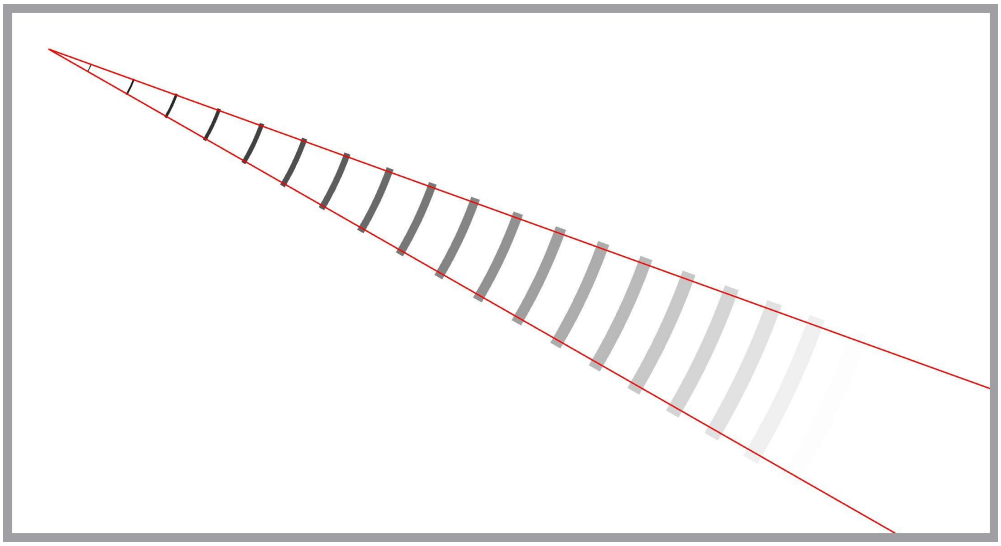

Fig 11: Propagazione del beam in assenza di ostacoli

L'intensità del rumore deve essere considerata in relazione all'intensità dell'eco di ritorno, ovvero deve essere considerato il rapporto segnale rumore SNR (Signal Noise Ratio). L'attenuazione dell'eco è solo in minima parte dovuta all'attenuazione del suono ($2.17 \ 10^{-3}$ dB/m), nel caso del range massimo di 50m si avrà un'attenuazione di -0.217 dB (0.977): da questo punto in poi si assumerà che l'attenuazione dell'onda è trascurabile e quindi che l'energia è costante.
Dato che l'onda emessa dall'echosounder ha forma conica (figura 9), anche se l'energia si conserva, al crescere della distanza percorsa la base del cono si allarga e quindi la densità di potenza sul fronte d'onda diminuisce.

$$E = \int_A \epsilon \, dA = \pi r^2 \epsilon \qquad \epsilon = \frac{E}{\pi r^2} \simeq \frac{E}{\pi (\theta d)^2}$$

$E = energia\ dell'\ onda\ (\mathrm{J})$
$A = area\ del\ beam\ (\mathrm{m}^2)$
$\epsilon = densità\ di\ energia\ dell'\ onda\ (\mathrm{J/m}^2)$
$\theta = ampiezza\ del\ beam\ espressa\ \mathrm{in}\ radianti$
$d = distanza\ percorsa\ dall'\ onda\ (\mathrm{m})$
$r = raggio\ della\ circonferenza\ (\mathrm{m})$

Il fronte d'onda può essere approssimato con una circonferenza la cui area aumenta con il quadrato della distanza e quindi l' SNR diminuisce quadraticamente. L'echosounder effettua automaticamente una normalizzazione dell'eco amplificandolo in base al tempo di volo (che ha una relazione lineare con la distanza percorsa), dunque per uno stesso oggetto posto a distanze diverse si avranno echi uguali ma con SNR diversi.

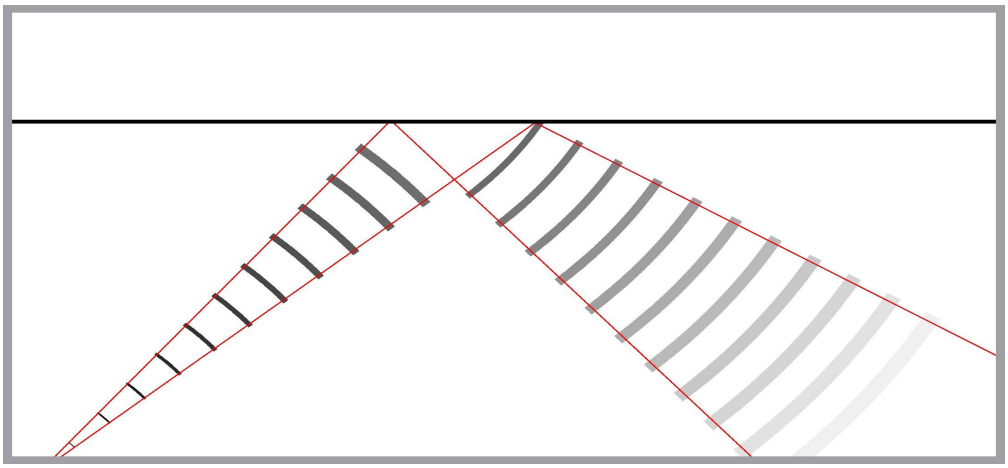

Fig 12: Riflessione del beam da parte di una parete piana

Solitamente gran parte dell'energia dell'onda incidente viene riflessa ma la potenza dell'onda si distribuisce in modi diversi nello spazio: una parete liscia tende ad avere un'eco molto intensa e concentrata in una direzione mentre superfici scabrose hanno un eco più diffusa. Se il beam ha un angolo di incidenza molto stretto con la superficie, solo una piccola parte dell'onda emessa torna in direzione dell'echosounder, mentre gran parte dell'energia segue le leggi tipiche delle onde elettromagnetiche: in particolare angolo di riflessione uguale all'angolo di incidenza (figura 12).

1.2 Generazione di immagini SONAR

La generazione di immagini SONAR è possibile sono nel caso in cui siano disponibil:

- La direzione dell'eco;
- I campionamenti dell'eco registrati da un sensore SONAR.

La direzione dell'eco solitamente viene misurata tramite l'impiego di una piattaforma inerziale (in questo caso l'Xsens MTi) mentre l'eco viene campionato dall'Imagenex ultra miniature echosounder.

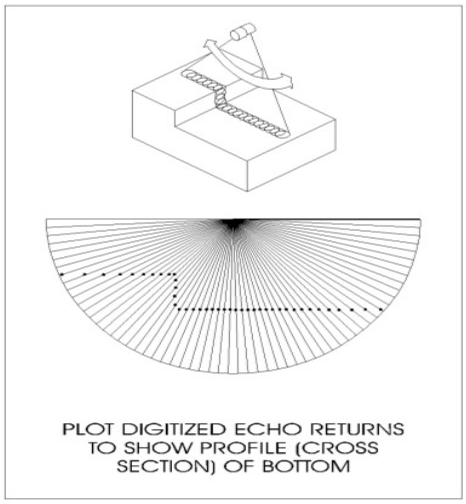

Fig 13: Meccanismo di generazione di immagini SONAR

In molti casi l'immagine SONAR di un oggetto può essere molto simile ad un'immagine ottica mentre in altri è di più difficile interpretazione. Ci sono due strade per l'interpretazione dei dati: nel caso di un ROV è importante generare un'immagine nitida la cui interpretazione è totalmente lasciata all'operatore del veicolo; nel caso invece di uno AUV è importante effettuare una fase di parsing dei dati che permetta poi ad un processo automatico di interpretare l'immagine al fine di estrarne parametri.

Le Fig 14,15 e 17 mostrano una scansione simulata del campo di gara del SAUC-e descritto nel dettaglio nel paragrafo 2.1.1; le tre immagini mostrano i tre formati di memorizzazione dei dati:

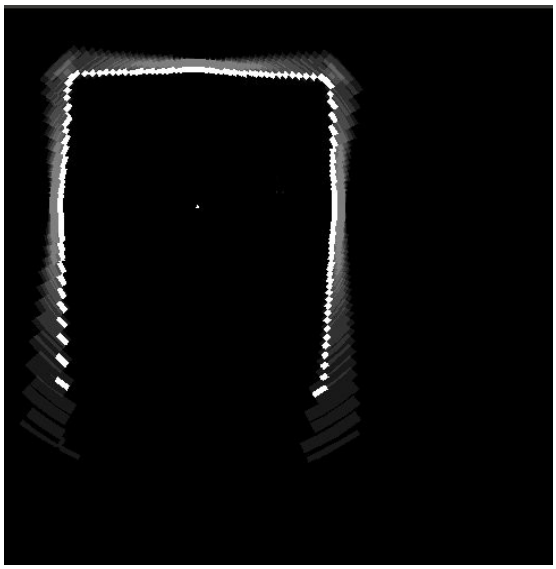

Fig 14: Immagine monocanale, impiegata per la memorizzazione dei dati

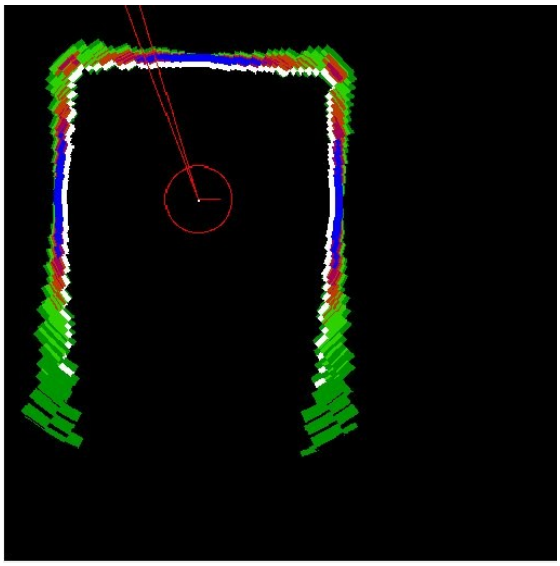

Fig 15: Equivalente della figura 11 ma con la scala dei colori riportata in figura 13

Fig 16: Scale dei colori impiegate rispettivamente per le immagini 14 e 15

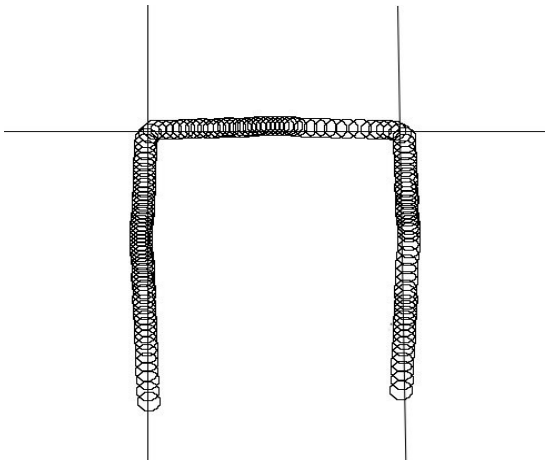

Fig 17: Immagine basata sulla prototipazione: riporta solo i punti in cui è stata rilevata una parete (le linee nere indicano le pareti identificate)

1. Immagine monocromatica: si tratta del formato ottimizzato per la memorizzazione dei dati;
2. immagine a tre canali: variando la scala cromatica dell'immagine precedente si ottiene un'immagine più semplice da leggere per un utente umano;
3. immagine booleana: si tratta di un'immagine prototype-based, in questo caso dopo una prima fase di parsing dei dati si realizza un'immagine che riporta solo i punti in cui le pareti della piscina sono state rilevate. Si tratta del formato che meglio si presta all'elaborazione da parte di un algoritmo.

1.3 Il filtro di Kalman

Nel 1960, R. E. Kalman pubblicò un articolo che descriveva una soluzione ottima ricorsiva al problema del filtraggio lineare di dati tempo discreto; da quel momento il filtro di Kalman è stato oggetto di numerose ricerche ed applicazioni. In questa sede verrà affrontato, in modo essenziale, solo il filtro di Kalman esteso tempo discreto.

Principi di base

Il filtro affronta il problema della stima dello stato $\boldsymbol{X}_k \in \mathbb{R}^n$ di un processo dinamico. Si assume che tale sistema possa essere descritto nel seguente modo:

$$\boldsymbol{X}_k = F\,\boldsymbol{X}_{k-1} + G\,\boldsymbol{u}_k + \boldsymbol{w}_{k-1} \quad (1)$$

Le misure $z_k \in R^m$ del processo sono effettuate a tempo discreto e si assume che abbiano una relazione con lo stato $\boldsymbol{x}$, del tipo:

$$\boldsymbol{z}_k = H\,\boldsymbol{X}_k + \boldsymbol{v}_k \quad (2)$$

Avremo una matrice di transizione $F^{n\times n}$ che descrive l'evoluzione libera dello stato, una matrice $H^{n\times m}$ che fornisce il collegamento tra lo stato $\boldsymbol{X}_k$ del sistema e le uscite $\boldsymbol{z}_k$ ed infine una matrice $G^{n\times l}$ che mette in relazione gli ingressi con le variazioni dello stato. Le variabili aleatorie $\boldsymbol{w}_{k-1}$ e $\boldsymbol{v}_k$ rappresentano rispettivamente il rumore del processo ed il rumore di misura :

$$E\left[w_k w_i^T\right] = \begin{matrix} \sigma_{w,i} & i=k \\ 0 & i\neq k \end{matrix}$$

$$E\left[v_k v_i^T\right] = \begin{matrix} \sigma_{v,i} & i=k \\ 0 & i\neq k \end{matrix}$$

$$E\left[w_k v_i^T\right] = 0 \qquad \forall k, i$$

inoltre:

$$Q=\begin{bmatrix} \sigma_{w,1} & \cdots & 0 & 0 & 0 \\ \vdots & \sigma_{w,2} & 0 & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & 0 & \sigma_{w,n} \end{bmatrix} \quad R=\begin{bmatrix} \sigma_{r,1} & \cdots & 0 & 0 & 0 \\ \vdots & \sigma_{r,2} & 0 & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & 0 & \sigma_{r,n} \end{bmatrix}$$

Le matrici di covarianza sono $Q \in R^n$ (matrice di peso degli stati) ed $R \in R^m$ (del rumore di misura); solitamente vengono considerate costanti. La matrice $F^{n \times n}$ nella (1) lega lo stato al passo temporale precedente $|k-1|$ allo stato attuale $|k|$. In assenza del rumore di processo, la matrice $G^{n \times l}$ mette in relazione l'ingresso opzionale di controllo $\boldsymbol{u} \in R^l$ con lo stato, mentre nella (2) la matrice H rapporta lo stato con le misure z. Il filtro di Kalman è uno stimatore ricorsivo: per ricavare lo stato attuale di un sistema sono necessari lo stato stimato al passo temporale precedente e la misurazione allo stato attuale. Possiamo suddividere le equazioni per l'implementazione del filtro in due gruppi: *equazioni di aggiornamento temporale* ed *equazioni di aggiornamento di misura*.

Aggiornamento temporale

Le equazioni di aggiornamento temporale hanno il compito di ricavare una stima *a priori* dello stato attuale $\boldsymbol{X_k}$ e della matrice di covarianza dell'errore stimato a priori P_k. Avremo:

$$\hat{\boldsymbol{X}}_{k|k-1} = F\,\hat{\boldsymbol{X}}_{k-1|k-1} + G\,\boldsymbol{u}_{k-1} \quad (3)$$

mentre $\hat{P}_{k|k-1}$ si ricava dall'equazione:

$$P_{k|k-1} = F\,P_{k-1|k-1}\,F^T + Q \quad (4)$$

I valori ricavati nelle stime a priori delle (3) e (4) saranno necessari per ricavare le stime a *posteriori* dello stato e della matrice di covarianza dell'errore, prodotte dalle equazioni di aggiornamento di misura.

Aggiornamento di misura

Il passo successivo all'aggiornamento temporale è l'aggiornamento di misura: il filtro produce una stima a *posteriori* incorporando la stima a priori con la misura proveniente dai sensori, sia per quando riguarda lo stato $\hat{X}_k$ che la matrice di covarianza dell'errore $\hat{P}_k$. Il primo passo

nell'aggiornamento di misura è quello di ricavare il guadagno di Kalman (o *blending factor*) che minimizza la covarianza dell'errore ricavato a posteriori, dall'equazione:

$$K_k = \hat{P}_{k|k-1} H^T (H \hat{P}_{k|k-1} H^T + R)^{-1}$$

per poi produrre la stima a posteriori dello stato dall'equazione:

$$\hat{\boldsymbol{X}}_{k|k} = \hat{\boldsymbol{X}}_{k|k-1} + K_k (\boldsymbol{z}_k - HX_k)$$

dove la differenza è chiamata *innovazione* o *misura residua*. L'ultima operazione consiste nell'ottenere una stima a posteriori della covarianza dell'errore di stima:

$$\hat{P}_{k|k} = P_{k|k-1} \boldsymbol{K}_k (H P_{k|k-1} H' + R) \boldsymbol{K}'_k$$

Al termine dell'aggiornamento di misura, i risultati ottenuti verranno utilizzati di nuovo allo step successivo come stima a priori dando luogo ad un procedimento ricorsivo.

1.4 Sistemi di riferimento

In questo paragrafo vengono introdotte le convenzioni relative ai sistemi di riferimento che variano a seconda dei veicoli trattati.

Caso SAUC-e (veicoli Turtle e FeelHippo)

- Terna di navigazione (fissa)
  Ha la sua origine nello spigolo nord est del bacino del CMRE di La Spezia (Coordinate 44.095842 Nord, 9. 864575 Est):
  - il versore x_n segue la direzione della parete est della piscina;
  - Il versore y_n segue la direzione della parete nord ella piscina;
  - il versore z_n punta nella direzione determinata dal prodotto vettoriale degli altri 2 vettori (verso il basso).

Fig 18: Immagine satellitare del bacino del CMRE

- Terna body:
  Ha la sua origine nel centro del veicolo ed è solidale con esso, tale terna non è inerziale ed i suoi versori sono orientati(figura 18):
  - il versore x_b punta verso la prua del veicolo;
  - il versore y_b laterale verso la sinistra del veicolo;
  - il versore z_b è tale da formare una terna destrorsa con x_b e y_b . ed è quindi laterale verso il basso.

L'orientazione del veicolo viene espressa, rispetto alla terna fissa, tramite angoli di eulero di tipo RPY (Roll Pitch Yaw).

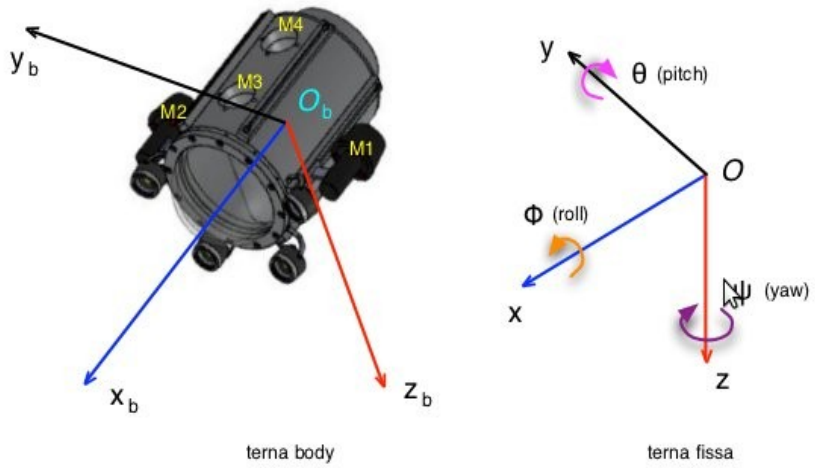

Fig19: Terna di riferimento RPY

Caso Progetto Nemo (veicolo Nemo Rov)
Per la terna body valgono le considerazioni appena fatte, per la terna fissa invece si impiega un sistema di riferimento NED (North East Down), l'origine della terna fissa non è importante in quanto la posizione del veicolo non viene stimata.

1.5 La trasformata di Hough

La trasformata di Hough è una tecnica di estrazione dei parametri impiegata nella computer vision per l'elaborazione di immagini; il suo scopo è la ricerca di forme geometriche imperfette all'interno di un'immagine tramite una procedura di voting. Tale procedura viene svolta in uno spazio parametrico che viene esplicitamente calcolato ed i prototipi vengono rilevati come massimi locali in tale spazio. La trasformata di Hough, così come viene universalmente impiegata oggi, è stata inventata da Richard Duda e Peter Hart nel 1972 che la chiamarono "trasformata di Hough generalizzata". La popolarità però cominciò nel 1981 nella "computer vision community" di Dana H. Ballard; oggi tale trasformata viene impiegata per la ricerca di linee, ma anche di altre primitive come ellissi e poligoni. Le linee rette nell'immagine vengono parametrizzate in modo tale da riportarsi ad una rappresentazione minima data da due parametri ρ e θ :

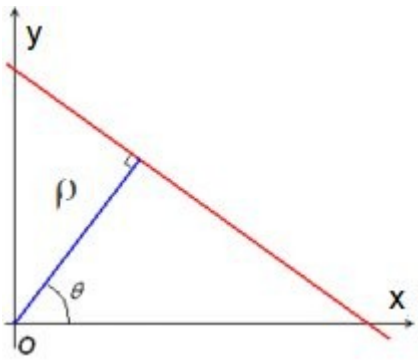

Fig 20: Sistema di parametrizzazione delle rette nell'immagine

Il calcolo della trasformata viene effettuato nel seguente modo:

1. l'immagine di partenza deve essere una matrice binaria di dimensioni $n \times m$, in questo caso avremo $A^{n \times m}$;
2. lo spazio parametrico è rappresentato da una matrice $H^{360 \times 2n}$ ma le dimensioni possono variare a seconda di quanto si vuole campionare finemente lo spazio parametrico;
3. Viene effettuata la procedura di voting:
   $k=0$
   $\forall i \in N, j \in M$
   Se $A_{i,j}=1$ allora

$$k = k + 1$$

$$H_{[\rho,\theta,k]} = H_{[\rho,\theta,k-1]} + i \cdot \cos(\theta) + j \cdot \sin(\theta) \quad \theta \in [0,360]$$

4. Si effettua una ricerca dei massimi all'interno dello spazio parametrico.

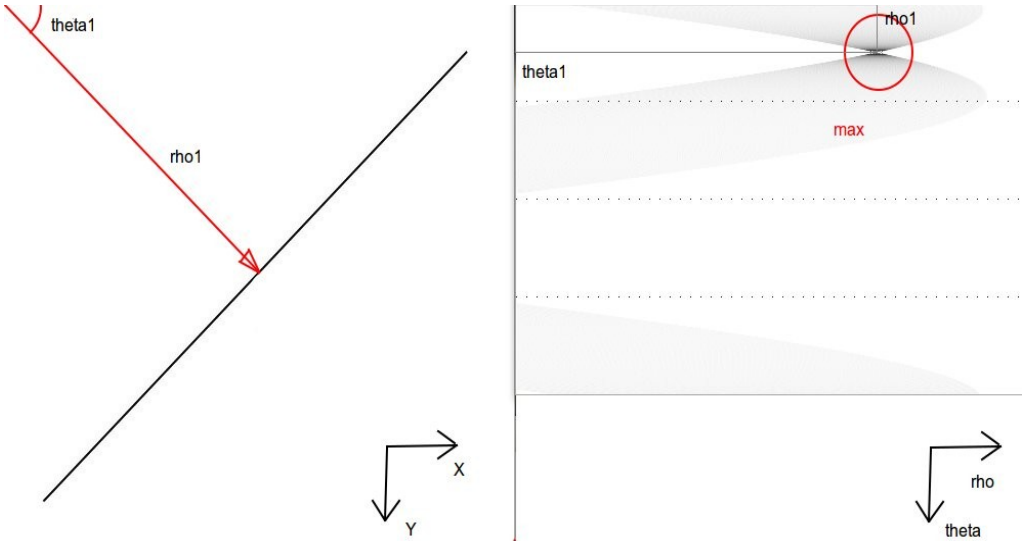

Fig 21: L'immagine riporta la trasformata di Hough relativa alla retta mostrata nel riquadro di sinistra; come si vede tale trasformata riporta un massimo (zona cerchiata in rosso) nel punto relativo ai parametri di tale retta.

2 I veicoli

In questo capitolo si riporta una descrizione dei veicoli impiegati nello sviluppo di questo lavoro di tesi; un'analisi accurata non è tra gli scopi di questo lavoro, tuttavia saranno approfonditi gli aspetti principali dei 3 veicoli Hippo, Nemo Rov e Turtle.

2.1 FeelHippo

FeelHippo è un veicolo sviluppato nel Laboratorio di Modellazione Dinamica e Meccatronica (MDM Lab) del Dipartimento di Ingegneria Industriale dell'Università di Firenze nell'ambito della competizione SAUC-e 2013. Il progetto, all'interno del laboratorio, è attivo da 2 anni e molte delle tecnologie sviluppate in questo ambito sono state reeimpiegate in altri veicoli.

2.1.1 La competizione SAUC-e 2013

La competizione nasce nel 2006 e vi partecipano numerose Università, in maggioranza europee. L'edizione 2013 si terrà dal 28 Giugno al 5 Luglio a La Spezia all'ex NATO Undersea Research Centre (NURC), ora Centre for Maritime Research and Experimentation (CMRE). Lo scopo di questa competizione è sviluppare tecnologie nell'ambito dei veicoli sottomarini autonomi (AUV). I team impegnati nella gara dovranno affrontare le problematiche caratteristiche della robotica autonoma e curare lo sviluppo del robot in modo interdisciplinare, dalla meccanica, all'elettronica al software. Esiste una competizione analoga negli Stati Uniti: la ROBOSUB CHALLENGE che si tiene a San Diego e che presenta caratteristiche molto simili al SAUC-e ma è più incentrata sulla manipolazione e sui sistemi di elaborazione delle immagini. Il regolamento di gara e soprattutto le caratteristiche dei "task" che il sottomarino deve affrontare variano di anno in anno: in queso documento la trattazione fa riferimento al regolamento 2013. Il sottomarino non deve solo effettuare le prove richieste, ma deve

anche generare un file di log con le caratteristiche indicate nel regolamento che dimostri un sufficiente grado di consapevolezza da parte del veicolo nei confronti dell'azione svolta.

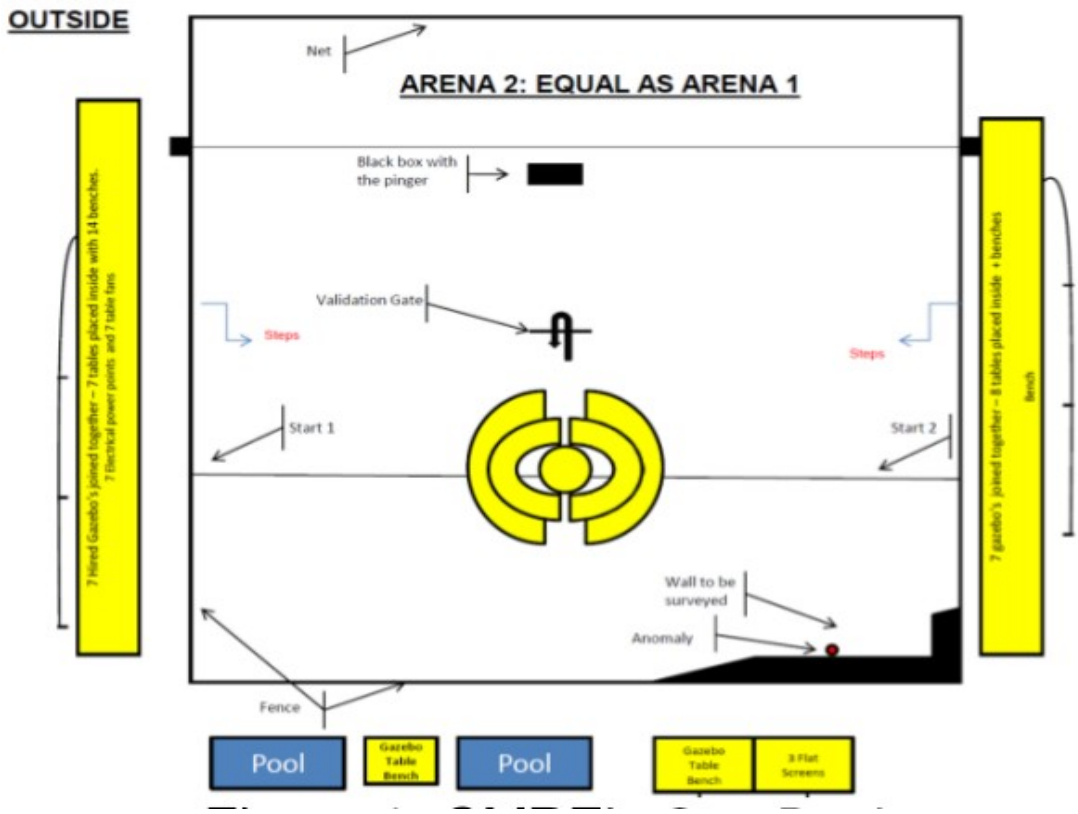

Fig 22: Schema del campo di gara

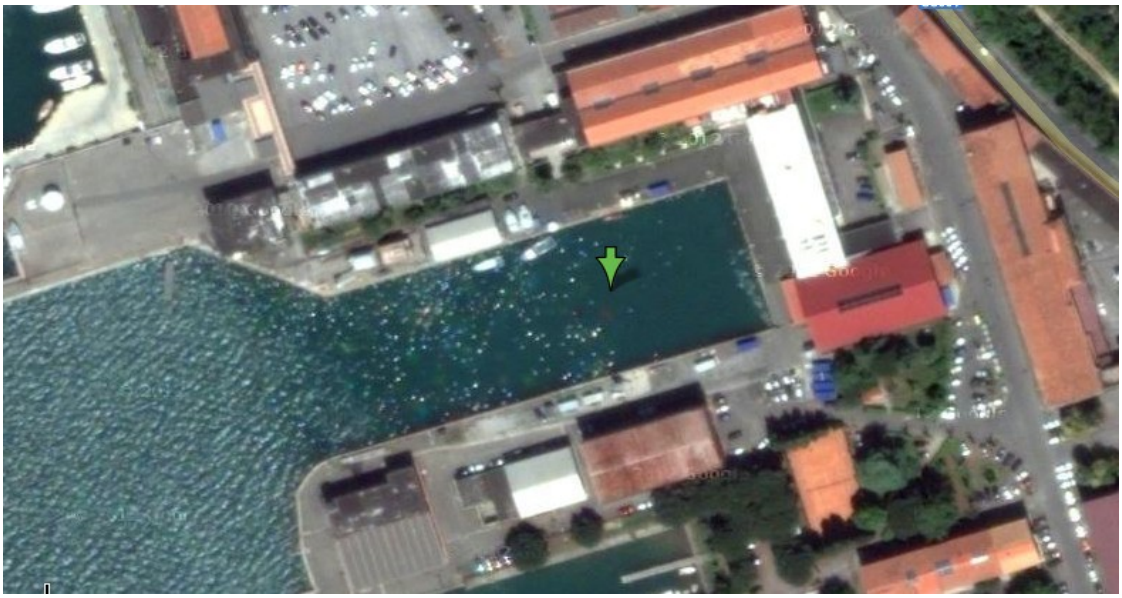

Fig 23: Un immagine satellitare del campo di gara.

La competizione si svolgerà in un bacino artificiale di dimensioni 50m per 120

m con una profondità di circa 5.5 m che è in diretta comunicazione con il mare; l'acqua presenta caratteristiche che variano fortemente in base alle condizioni meterologiche, all'intensita della luce solare e al livello di inquinamento: solitamente la visibilità è compresa tra 1 e 2 m.

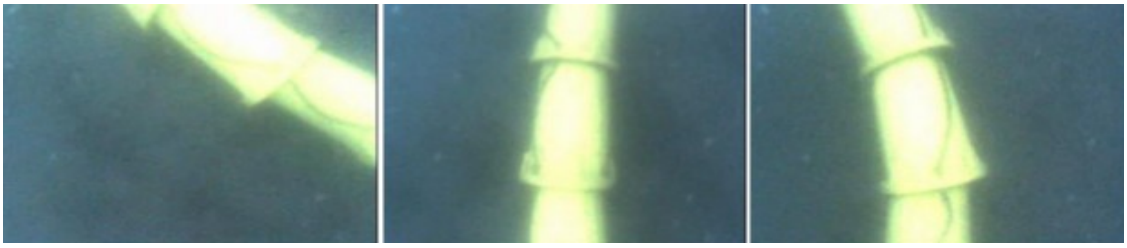

Fig 24: Fotografia di tubi da cantiere posti sul fondale del campo di gara

La competizione si sviluppa in 4 task. Il punteggio relativo ad ogni prova varia in base alla modalità di esecuzione: se la prova viene eseguita singolarmente oppure se viene eseguita in successione con altre prove. Inoltre i giudici di gara possono discrezionalmente aumentare il punteggio in base a valutazioni particolari. Nel seguito indicheremo il punteggio relativo ad ogni task con (SSS,MMM,PPP) dove SSS indica il punteggio relativo alla prova eseguita singolarmente, MMM indica il punteggio della prova eseguita in sequenza con altre prove e PPP indica il punteggio aggiuntivo decretato dai giudici di gara.

Di seguito si descrivono brevemente le prove proposte per il 2013:

1. "passing the validation gate": il veicolo deve immergersi nel punto 1 o 2 della figura 22, deve raggiungere in modo approssimativo il centro della piscina, deve ruotare di 90° e attraversare una struttura composta da due pali verticali detta "validation gate" senza entrare in contatto con alcuna parte di essa. Tale struttura sarà indicata da due boe rosse che saranno visibili dalla distanza di 1-2 metri circa. (150/150/150);
2. "underwater structure inspection": il veicolo deve identificare la struttura posta al centro della piscina composta da tubi da cantiere di diametro 0.5 m e lunghezza 1.5 m disposti in cerchi concentrici su più file sovrapposte; possono essere impiegati sensori SONAR o video camere. Il punteggio sarà determinato in base alla qualità dei dati forniti. (600/300/300);
3. "inspect the wall, find the anomaly call a friend": il veicolo deve inseguire la parete est della piscina mantenendosi ad una distanza

compresa tra i 2 ed i 4 metri e deve rilevare la presenza di una boa rossa; in seguito deve trasmetterne ad un altro AUV la posizione.
Sono previsti punti aggiuntivi nel caso in cui l'altro veicolo raggiunga l'anomalia e la rilevi correttamente. (2000,1000,1000) ;

4. "map the environment, find the pinger": il veicolo deve realizzare una mappa dell'ambiente basata sui dati provenienti da una videocamera, inoltre deve rilevare la presenza di un dispositivo detto "black box". Il rilevamento può essere effettuato tramite un' esplorazione su tutta l'area di gara effettuata con la videocamera oppure sfruttando una sequenza di impulsi sonori emessi dall'oggetto; gli impulsi hanno una durata di 10 μs alla frequenza di 15 kHz. (1500,750,750).

2.1.2 Breve descrizione del veicolo

Il corpo centrale di FeelHippo è costituito da un tubo di plexiglass che contiene l'hardware del veicolo con l'eccezione delle batterie che sono inserite nei due tubi in alluminio posti sotto il corpo. Alle estremità del tubo sono poste due cupole trasparenti, anch'esse in plexiglass e connesse al corpo centrale tramite due anelli strutturali in alluminio; la rigidezza dell'insieme è garantita da 6 barre che che fungono anche da supporto per alcuni dispositivi esterni allo scafo.

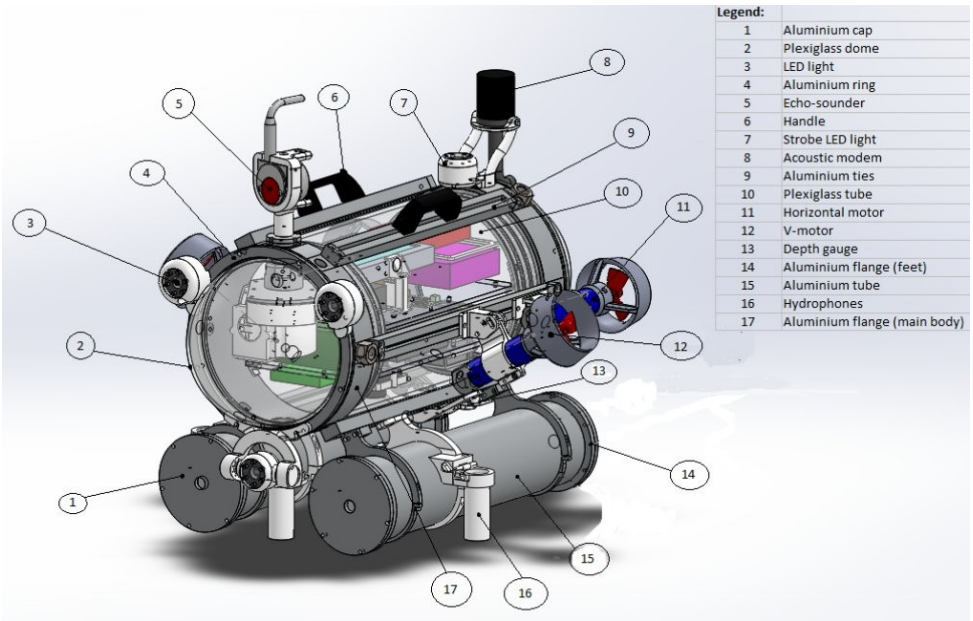

Fig 25: Disegno 3D del veicolo

Alcuni componenti, come thrusters, profondimetro e luci led, sono connessi alla struttura principale tramite supporti plastici (in ABS) realizzati tramite la macchina per la prototipazione rapida disponibile presso l'MDM Lab; i componenti in alluminio invece sono stati realizzati da un azienda specializzata (STERN PROGETTI srl).
La propulsione è garantita da 4 motori (capaci di controllare 4 dei 6 gradi di libertà del veicolo): due di questi sono montati a V in modo da controllare la profondità ed il movimento in direzione laterale; gli altri due invece sono orientati orizzontalmente per controllare l'angolo di imbardata e la velocità di avanzamento longitudinale. Il veicolo è equipaggiato con un vasto assortimento di dispositivi tra cui in particolare :

- piattaforma inerziale Xsens MTi;
- convertitore DC-DC che assicura le varie tensioni di alimentazione necessarie ai diversi dispositivi;
- display LCD posto all'interno del corpo del veicolo;
- PC industriale Commell SBC;
- 2 webcam una frontale, l'altra rivolta verso il fondale;
- Imagenex 852 ultra miniature echosounder.

I dispositivi sono fissati all'interno del corpo centrale tramite due piani in plexyglass removibili che possono essere inseriti ed estratti con facilità. In definitiva la struttura del veicolo garantisce:

- accesso all'elettronica di bordo semplice e veloce;
- orientazione dei thruster variabile;
- taratura della posizione del baricentro accurata tramite la distribuzione dei pesi all'interno dei vani batterie in alluminio;
- ridotto peso (circa 25 kg);
- ridotto prezzo grazie all'elevato numero di componenti progettati e realizzati in autonomia.

Il sistema di localizzazione del veicolo richiede che l'echosounder possa ruotare parallelamente all'asse di imbardata del veicolo. È stato quindi realizzato il dispositivo mostrato in figura 26. Con l'obiettivo di mantenere un costo ridotto per l'intero veicolo è stato deciso di impiegare un sistema di movimentazione basato su un azionamento da modellismo: Robotis Dynamixel ax-18A mostrato in figura 27. Il servo motore permette all'echosounder di ruotare con una precisione di circa 0.5° e con una velocità

di 30°/s; il compartimento in ABS che contiene il dispositivo è reso stagno per garantire la sicurezza del resto dell'elettronica in caso di trafilamenti attraverso i paraoli dell'albero. Nel complesso il modulo echosounder ha dimensioni piuttosto compatte e prestazioni comparabili con la maggior parte degli mechanically scanned imaging SONAR.

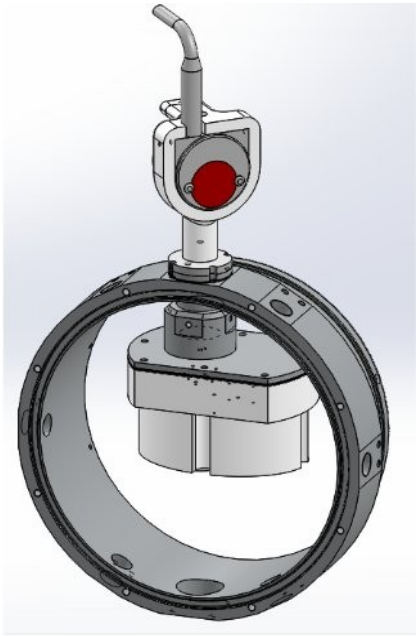

Fig 26: Assieme echosounder-azionamento

Fig 27: Foto del Robotis Dynamixel ax-18

2.1.3 Obiettivi e specifiche

Obiettivo team

Realizzare un veicolo che in autonomia sia in grado di eseguire il più alto numero di task proposti possibile, generando un log secondo le modalità indicate dal regolamento; per la fase di messa a punto è necessaria anche una suite di strumenti che permetta il controllo del veicolo in remoto ed il monitoraggio di tutti i parametri necessari per la messa a punto dello stesso.

Obiettivo individuale

Sviluppare il software impiegato nella gestione dei dispositivi Xsens MTi, Imagenex 852 echosounder, Pololu Maestro 6, Dynamixel ax-18. L'obbiettivo principale è realizzare un sistema di localizzazione che possa essere impiegato con successo nell'ambito della competizione SAUC-e 2013.

Specifiche del progetto

I dispositivi impiegati a tale scopo sono :

- computer di bordo Commell ;
- controllore per servocomandi/azionamenti Pololu Maestro 6.0;
- servocomando Dynamixel ax-18;
- Imagenex 852 ultra miniature echosounder;
- piattaforma inerziale Xsens MTi.

Il software dovrà essere integrabile con le applicazioni sviluppate dagli altri membri del team, in particolare deve essere compatibile con ROS (vedi paragrafo 3.2.1) e deve essere compilato su architettura GNU-LINUX. L'architettura del sistema di controllo è strutturata in maniera modulare gerarchica alla cui base si trovano i processi dedicati ai dispositivi di misura e controllo, al secondo livello i sensori dedicati al parsing dei dati mentre dal terzo livello in poi i veri e propri processi di controllo. Tali processi saranno costituiti da un generatore di traiettoria dedicato per ogni task ed un controllore PID comune.

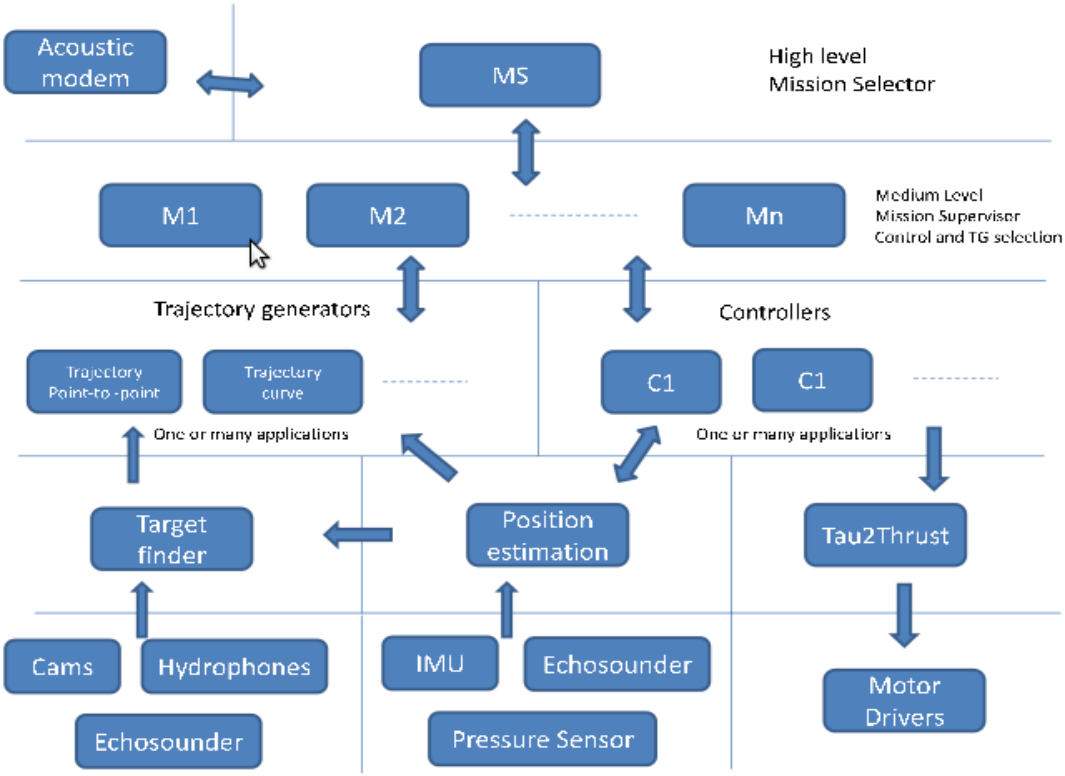

Fig 28: Schema indicativo dell'implementazione del software

2.2 Nemo Rov

2.2.1 Il progetto Nemo

Alle 21:42 del 13 Gennaio 2012, la nave da croclera costa Concordia ha urtato il più piccolo degli scogli de Le Scole situato a circa 500 m dall'isola del Giglio, provocando un parziale affondamento della nave e causando 30 morti, 2 dispersi e numerosi feriti.
Su richiesta del Consiglio Dei Ministri il dipartimento di Scienze della Terra dell'Università degli Studi di Firenze sta coordinando le attività di monitoraggio della nave, all'operazione prendono parte anche la struttura tecnica di missione della Protezione Civile con le forze operative (Vigili del Fuoco, Capitaneria di Porto, Marina Militare), con i Centri Funzionali del DPC e della Regione Toscana con Centri di Competenza (CNR-IRPI, ASI), enti di ricerca (EC-JRC-IPSC, OGS, Fondazione Prato Ricerche, Politecnico di Milano) ed aziende specializzate.

Fig 29: Il golfo del Giglio, luogo dell'affondamento

Il 18 Gennaio 2012 è stato installato un sistema di monitoraggio che permette di misurare in tempo reale le deformazioni dell'intero scafo con un'accuratezza inferiore al millimetro, il continuo lento movimento della nave

determinato dalle condizioni meteo e marine può ostacolare le operazioni di ricerca e recupero, inoltre, per questioni di sicurezza, i sommozzatori non possono raggiungere determinate zone del relitto: in questo contesto si colloca il veicolo Nemo ROV.

Nemo è un ROV (Remote Operated Vehicle) versatile e maneggevole progettato per operare in condizioni ambientali estreme, in particolare tramite una telecamera robotizzata deve essere in grado di esplorare anche le zone interne del relitto e deve essere semplice da pilotare in modo da rendersi utilizzabile anche per un utente generico. Il prototipo è stato realizzato grazie alla collaborazione fra il Dipartimento di Scienze della Terra ed il Laboratorio di Modellazione Dinamica e Meccatronica (MDM Lab) del Dipartimento di Ingegneria Industriale dell'Università degli Studi di Firenze.

Fig 30: Gli enti coinvolti nel progetto

2.2.2 Breve descrizione del veicolo

Nemo è costituito da un corpo cilindrico in alluminio contenente la strumentazione elettronica; il corpo centrale è chiuso anteriormente da una calotta semisferica in Plexiglass trasparente che garantisce alla telecamera robotizzata interna una visibilità di circa 180°, posteriormente la tenuta è garantita da un tappo circolare di alluminio appositamente forato per il passaggio dei cavi.

Il veicolo è dotato di 4 propulsori ad elica Crustcrawler High Flow 400HFS-L di cui 2 orizzontali per il controllo di velocità e imbardata e 2 verticali per il controllo di beccheggio e profondità.

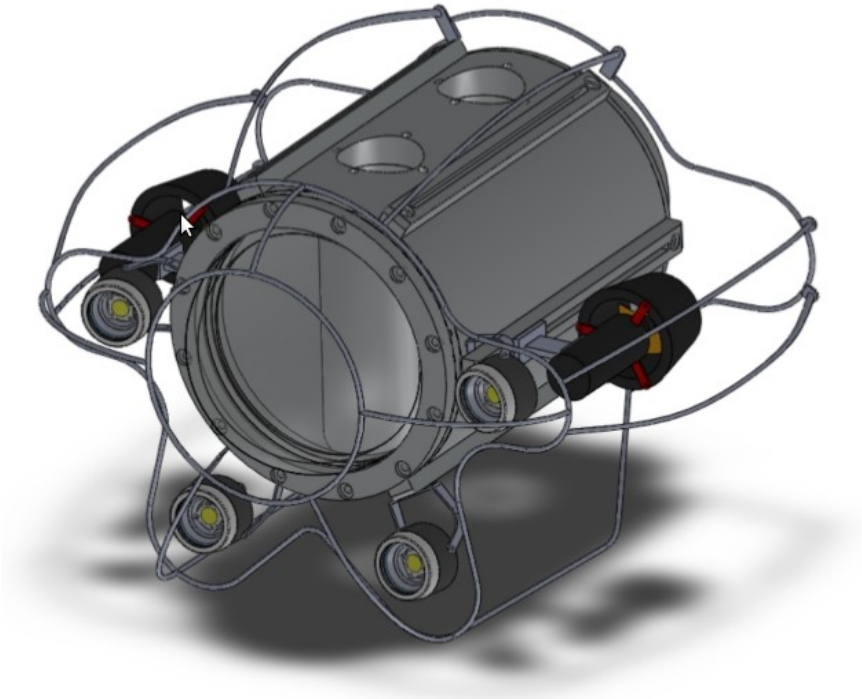

Fig 31: Disegno 3D di Nemo Rov

Il corpo è circondato interamente da una gabbia di protezione in alluminio che in futuro sarà avvolta in un materiale termo-plastico che assicuri un'ulteriore protezione. Il peso del veicolo in aria è di 17 kg che in acqua diventano 0.4 kg; tramite dei piccoli galleggianti distribuiti sulla gabbia esterna il veicolo può essere equilibrato con precisione evitando l'appruamento dovuto alla differenza di peso tra l'oblò frontale in plexyglass e il tappo posteriore in alluminio.

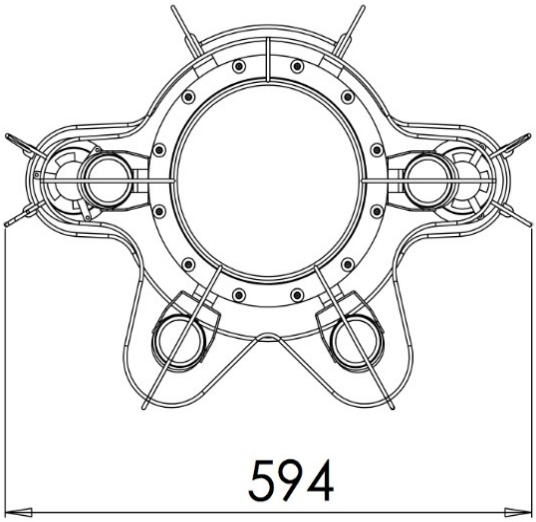

Fig 32: Vista frontale

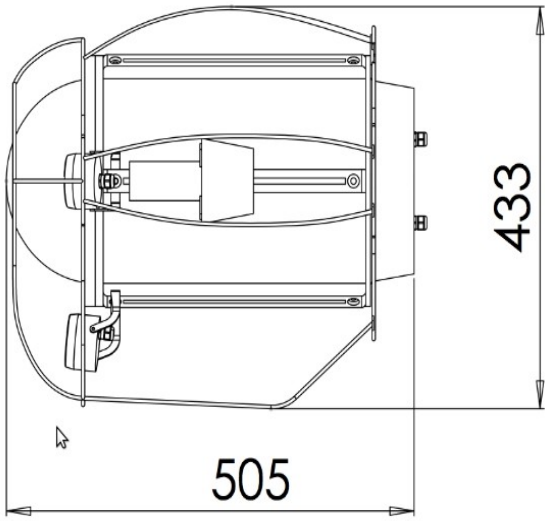

Fig 33: Vista laterale

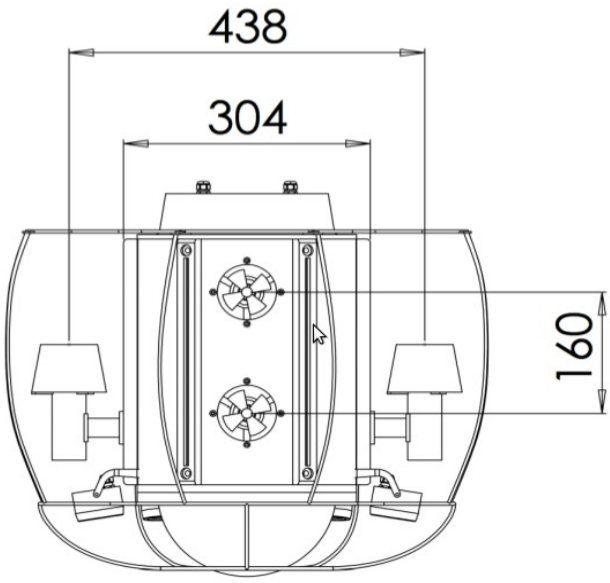

Fig 34: Vista superiore

Sistema elettrico e cavo

Nemo non dispone di un sistema di alimentazione autonomo, viene alimentato tramite un cavo alla tensione di 48 Volt, sul veicolo è presente un convertitore DC/DC che fornisce le tensioni necessarie ai dispositivi a bordo. Il cavo è composto da 3 componenti: alimentazione, cavo ethernet per la trasmissione dati e rinforzo in Dyneema che garantisce un elevato grado di resistenza alla trazione. Lungo tutta la lunghezza sono disposti galleggianti che in acqua garantiscono una leggera positività dell'insieme.

Fig 35: Foto del cavo di alimentazione

Fig 36: Il convertitore DC-DC impiegato sul veicolo

Elettronica e software

L'equipaggiamento del veicolo comprende:

- 2 computer di bordo Raspberry Pi;
- Piattaforma inerziale Xsens MTi;
- Profondimetro STS DTM;
- Imagenex Echosounder 852;
- Videocamera Samsung SNP-3120;
- Scheda di controllo azionamenti Pololu Maestro 6.

L'architettura del software è distribuita sui 2 computer di bordo che comunicano tramite un cavo LAN con un protocollo TCP IP. Il processo che implementa il controllore del veicolo riceve informazioni dai processi dedicati ai vari dispositivi e viene eseguito su uno dei computer di bordo; sul computer dell'operatore invece viene eseguito il processo dedicato all'interfaccia utente che permette all'operatore di pilotare il veicolo.

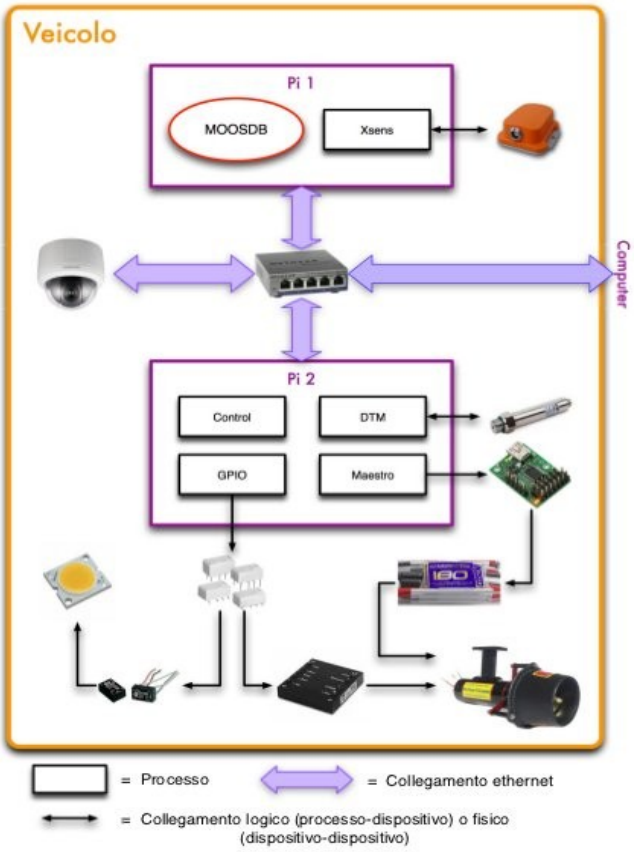

Fig 37: Struttura del software di Nemo Rov

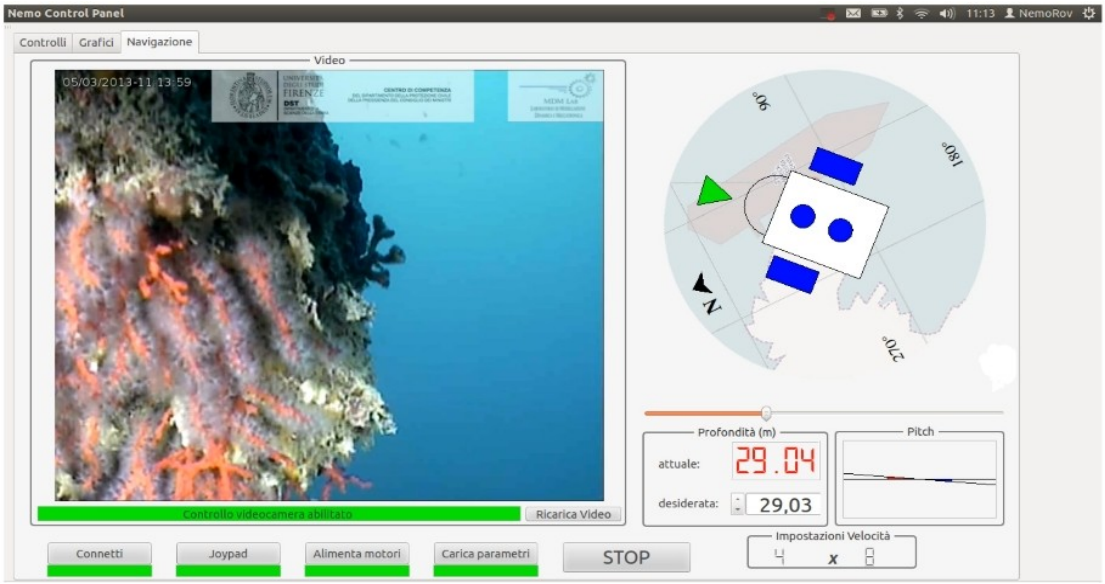

Fig 38: Screenshot del software impiegato per il controllo di Nemo ROV

2.2.3 Obiettivi e specifiche

Obiettivo del progetto

La realizzazione di un veicolo versatile e leggero dotato di una strumentazione sufficiente allo studio dello stato del relitto della nave Costa Concordia; in particolar modo il veicolo deve poter raggiungere quelle zone dello scafo che non possono essere esplorate dal personale umano per ragioni di sicurezza.

Obiettivo individuale

- Sviluppare il software impiegato nella gestione dei dispositivi Xsens MTi e Imagenex Echosounder. I due processi saranno indipendenti ma dovranno essere in grado di fondere le misure generando un'unica struttura dati che contenga l'orientazione dell'echosounder e le misure dell'eco relative ad un determinato istante di tempo; tale struttura verrà inviata ad un processo client eseguito da un'altra macchina tramite un protocollo TCP IP. Questi processi devono essere eseguiti sul Rasperry Pi in real time e quindi devono avere un costo computazionale contenuto;
- Sviluppare un processo client che riceva la struttura dati appena descritta e la elabori in real time in modo da generare un'immagine che rappresenti l'eco dell'ambiente in cui si trova il veicolo.

Il software dovrà essere integrabile con le applicazioni sviluppate dagli altri membri del team e deve essere compilato su architettura ARM-GNU-LINUX.

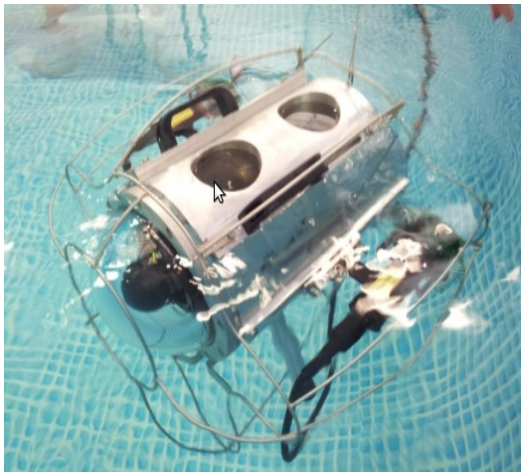

Fig 39: Il veicolo Nemo nella piscina dell'MDM Lab

2.3 Turtle

Il Turtle è un veicolo sviluppato dal SAUC-e team del Laboratorio di Modellazione Dinamica e Meccatronica del Dipartimento di Ingegneria Industriale dell'Università degli Studi di Firenze per la competizione Sauc E 2012; si tratta di un veicolo per alcuni aspetti ormai obsoleto rispetto a FeelHippo e a Nemo, tuttavia a suo tempo ha permesso lo sviluppo delle tecnologie tuttora impiegate nei prototipi più recenti.

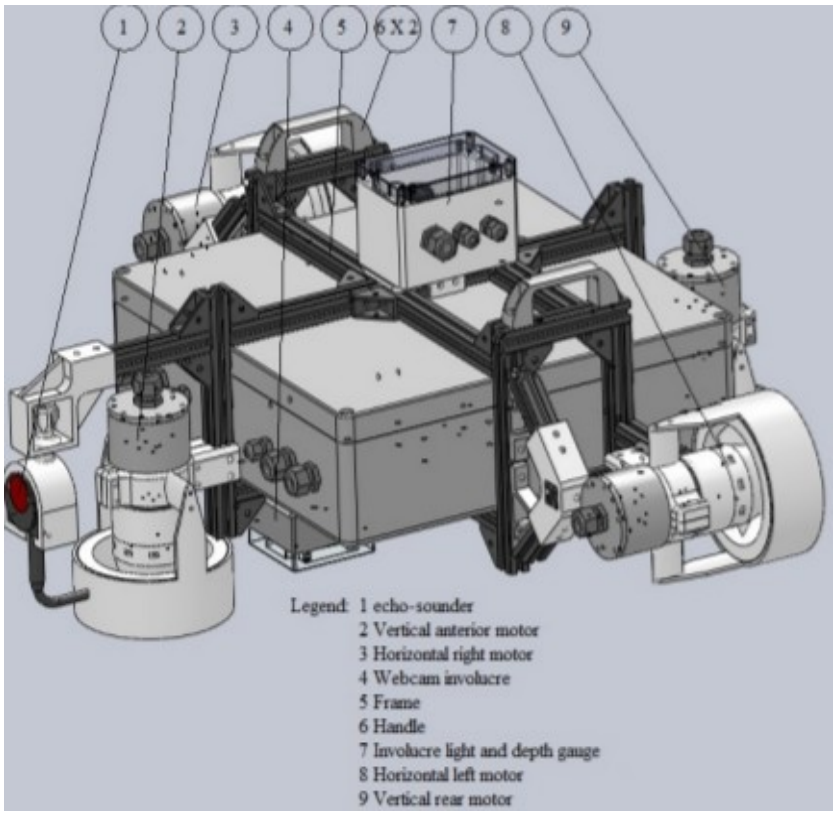

Fig 40: Il veicolo Turtle

2.3.1 Breve descrizione del veicolo

Il corpo centrale del veicolo è costituito da un case impermeabile per impianti; esternamente un telaio formato da profilati di alluminio collega rigidamente tutti i componenti della struttura per un peso totale di 27 kg. La propulsione del veicolo viene sviluppata da 4 motori

di cui 2 orizzontali (controllo di imbardata e velocità) e 2 verticali (controllo di profondità e beccheggio). I sensori sono:

- SONAR (in posizione fissa a prua);
- telecamera (in un case dedicato sotto il veicolo);
- profondimetro (ospitato nel piccolo case sopra il veicolo).

Il veicolo è equipaggiato con 4 batterie LiPo (24V, 8000mAh, 30 C) che assicurano un'autonomia di circa 2 ore in pieno funzionamento, l'elettronica è composta da:

- Pc Commell: pc industriale ad alte prestazioni;
- Scheda Arduino 1: si tratta di un dispositivo simile al Maestro che veniva impiegato per il controllo di alcuni sistemi secondari ;
- Scheda Pololu Maestro 6: impiegato anche nel Turtle 2 gestisce gli azionamenti dei 4 motori;
- SONAR Imagenex 852 ultra miniature echosounder;
- Profondimetro ;
- Piattaforma inerziale Xsens MTi;
- Web Cam .

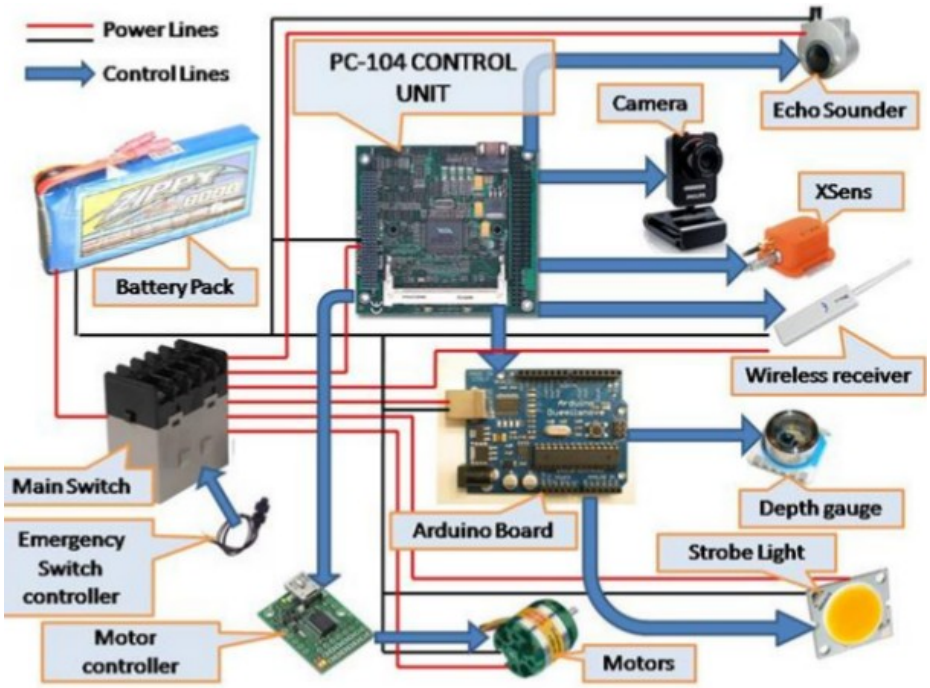

Fig 41: Equipaggiamento del veicolo

L'architettura del software è semplice: ad ogni sensore corrisponde un processo ed ogni processo comunica con gli altri tramite un database in uno schema "star shaped" nel quale ogni messaggio scambiato tra i processi passa attraverso il "main process" MOOSDB.

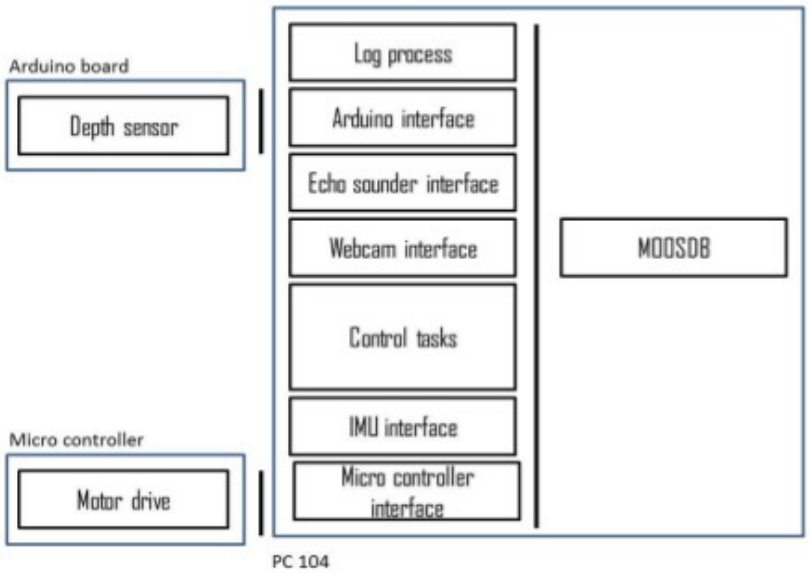

Fig 42: Struttura del software

I propulsori costituiscono l'aspetto più innovativo del progetto: i motori sono inseriti all'interno di un alloggiamento cilindrico realizzato in materiale plastico e la coppia viene tramessa tramite dei giunti magnetici senza bisogno di paraoli o fori.

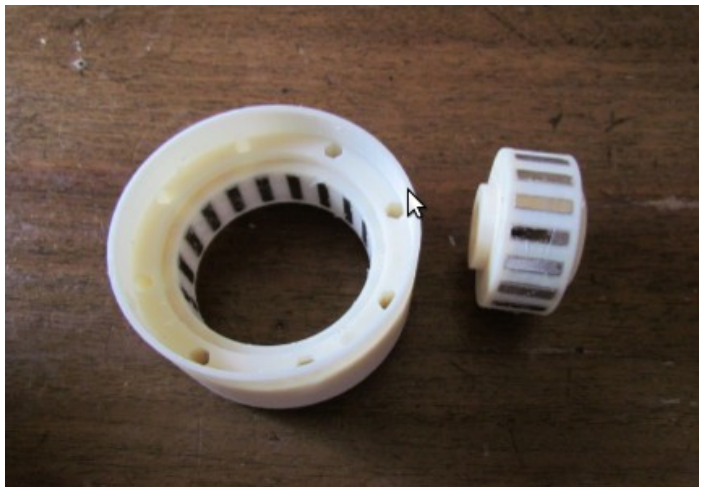

Fig 43: Giunto Magnetico impiegato per la trasmissione della coppia da motore a elica

2.3.2 Eredità tecnologica del progetto

Nel corso della settimana del SAUC-e 2012 il veicolo ha dimostrato grandi potenzialità che sono valse il premio "rookie of the year", tale esperienza è stata importante ed ha permesso di mettere in evidenza problematiche che nella fase progettuale non erano emerse completamente. In questo paragrafo si riportano brevemente i principali problemi emersi durante le prove e le rispettive soluzioni sviluppate nell'ambito dei progetti successivi Nemo e FeelHippo (l'analisi è incentrata sul software).

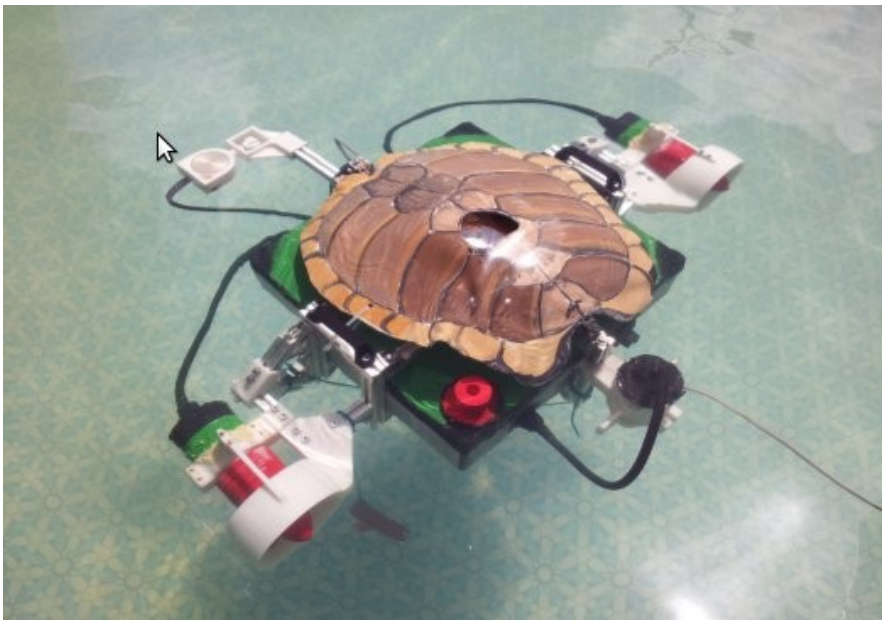

Fig 44: Il veicolo Turtle nella piscina del campo gara del CMRE

- Coppia dei motori inseriti nei thruster non sufficiente:
  - impiego di motori con una coppia maggiore e dotati di un riduttore, diminuzione delle dimensioni delle eliche.
- Elevata zona morta degli azionamenti:
  - nuovo settaggio degli azionamenti, introduzione di un riduttore sull'albero motore che riduce la zona morta.
- L'apertura/chiusura dello scafo richiedeva molto tempo:
  - impiego di stagnature basate su guarnizioni che non richiedono l'uso di silicone.

- Software di lettura-scrittura seriale inaffidabile:
  - È stato sviluppato un nuovo software basato sulle librerie standard del kernel di Linux (vedi paragrafo 5.2.1).
- Software da ricompilare ad ogni variazione dell'indirizzo dei dispositivi (cioè ad ogni collegamento/avvio del pc):
  - Impiego di indirizzi univoci per i dispositivi(vedi par 5.2.1).
- Profondimetro inaffidabile:
  - Realizzazione di un nuovo circuito stampato idustrialmente.

3 Ambiente di sviluppo del software

3.1 Ambiente di sviluppo

Quando si lavora in un team è importante determinare un ambiente di lavoro semplice e funzionale, deve essere concepito in modo tale da permettere ad ogni membro di lavorare con un sufficiente livello di autonomia, ma assicurandosi che il lavoro di ognuno si interfacci con facilità con il resto del sistema. Per questo motivo sono state definite alcune caratteristiche che tutto il software sviluppato deve avere: ad esempio il liguaggio di programmazione impiegato è il C++ ed il SO è Ubuntu con vers > 11.10. Unire il lavoro di programmatori diversi può essere più facile se si impiegano strumenti adeguati, solitamente si ricorre ad un middle-ware, inoltre sono stati impiegati alcuni framework per semplificare il lavoro dei programmatori.

3.2 Middleware

In termini generali, il middleware è il software che mette a disposizione di altri processi i servizi tipici di un istema operativo, a volte viene definito con il termine "collante per software". I middleware permettono ai programmatori di realizzare singoli programmi che assolvono ad una specifica funzione e che poi comunicano tramite i servizi messi a disposizione dal middlware.
Sistemi di questo tipo si stanno affermando sempre più in robotica e sono solitamente basati sul protocollo TCP che, oltre ad essere affidabile e performante, permette di implementare con facilità sistemi distribuiti o paralleli. Ne sono un esempio ROS (Robot Operating System) e MOOS () rispettivamente impiegati sui veicoli FeelHippo e Nemo.

3.2.1 ROS (Robot Operating System)

Fig 45: Logo di ROS

ROS (Robot Operating System) è un middleware molto diffuso impiegato in particolar modo nel controllo dei robot, implementa i servizi tipici di un sistema operativo come l'interfacciamento hardware ed i servizi di comunicazione tra i processi; inoltre mette a disposizione librerie che permettono di elaborare i dati dei sensori in modo semplice ed efficiente. Ad oggi ROS è compatibile con linux Debian e MAC OS X, inoltre ci sono delle versioni non ufficiali che permettono di lavorare su Fedora, Gentoo, Arch Linux e molte delle più diffuse distribuzioni linux. Come descritto nel seguito ROS si basa su una logica distribuita.

Concetti fondamentali:

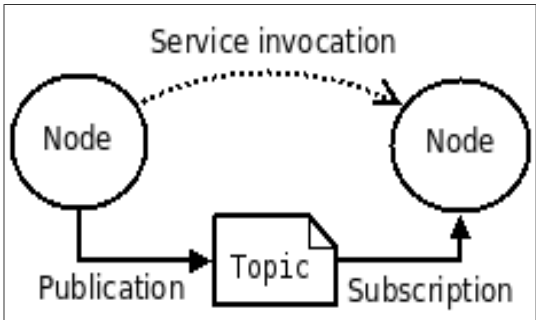

Fig 46: Meccanismo di comunicazione tra nodi

La filosofia di ROS è quella di creare un primo processo master che contiene le informazioni relative a tutti i "nodi", ovvero i vari processi del sistema. All'avvio, i nodi si collegano al "master" e recuperano tutte le informazioni relative agli altri nodi che consistono in "topic" e "servizi"; così facendo si viene a creare una rete di comunicazione tra i processi basata sul protocollo TCP-IP che presenta un elevato livello di robustezza e versatilità anche in presenza di un gran numero di nodi e ingenti flussi di dati.

Nodo: un nodo è un processo indipendente, tale processo può comunicare con gli altri nodi tramite le topic, i servizi ed il parameter server. Solitamente un sistema di controlo contiene molti nodi: nel caso di un veicolo sottomarino potrebbero esserci un nodo che genera la traiettoria, un nodo che legge il sensore di profondità, un nodo che comunica con l'azionamento dei thruster etc... L'uso dei nodi ha molti vantaggi: in primo luogo permette di suddividere il codice diminuendone la complessità e in caso di errori permette di isolare la causa del problema. Ogni nodo ha un graph resource name che lo

identifica in modo univoco rispetto al resto del sistema, le funzioni dedicate alla comunicazioni sono contenute nella ROS client library. Ros mette a disposizione dei tool che permettono di stampare graficamente i collegamenti tra i nodi.

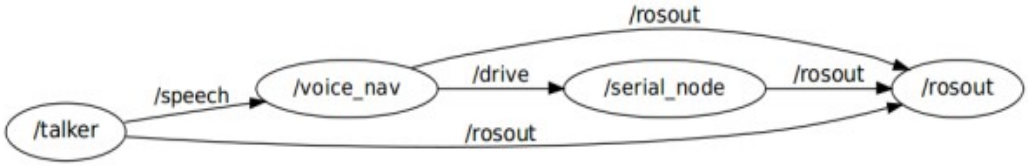

Fig 47: Un esempio di un grafo autoplottato della topologia dei nodi del sistema

Topic: le topic sono le vie attraverso cui i nodi scambiano messaggi; sono strutturate in maniera tale da disaccoppiare la produzione di informazioni dal consumo delle stesse. Se un processo vuole comunicare deve iscriversi ad una topic; non può sapere quali altri nodi vi sono collegati ma può leggere o scrivere sullo stesso. L'utilizzo delle topic è da intendersi come unidirezionale: uno o più nodi producono i dati mentre uno o più topic li consumano; si definisce un many-to-many one way transport.

Servizi: il flusso dati dei topic deve essere unidirezionale, i servizi invece permettono un flusso dati bidirezionale: si definisce una struttura dati in ingresso, una in uscita ed una funzione che riceve come argomento i dati in ingresso e restituisce in uscita i dati elaborati; si definisce un tipo di connessione one-to-one two way transport.

Bag: si tratta del file che contiene un log dei messaggi scambiati tra i nodi; si tratta di un potente strumento di debug che permette di simulare in modo semplice e realistico interi nodi del sistema.

Master: il master è il processo che gestisce le iscrizioni dei nodi ai topic ed ai servizi; fondamentalmente il ruolo del master è quello di mettere in comunicazione i nodi tra loro. Una volta messi in collegamento comunicheranno autonomamente tramite una connessione peer-to-peer. Il master inoltre contiene il parameter server.

Parameter Server: si tratta di un processo che, come un database, immagazzina determinate variabili che sono accessibili da tutti i nodi; non è indicato per un utilizzo ad alte prestazioni ma piuttosto per la condivisione di parametri tra più nodi che vengono acceduti e modificati con frequenze basse, a tali parametri si accede attraverso il master.

3.2.2 MOOS ivp

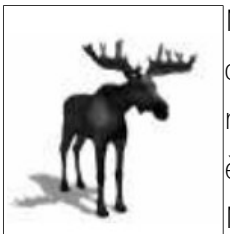

Fig 48: Il logo di MOOS

MOOS è un middleware sviluppato al MIT di Boston, con caratteristoche pensate per adattarsi ad applicazioni di robotica sottomarina; non ha le stesse potenzialità di ROS ma è di più semplice implementazione.

MOOS ha una topologia "star shaped": ogni processo facente parte del sistema ha una connessione con un "MOOS Database" che costituisce un area di memoria condivisa tra tutti i processi e rende possibile lo scambio dati tra gli stessi. La rete creata ha le seguenti caratteristiche:

- Nessun collegamento peer-to-peer tra i diversi processi;
- ogni client deve avere un nome univoco;
- ogni client non è a conoscenza dell'esistenza degli altri client;
- ogni client può comunicare con gli altri client solo attraverso MOOSDB.

Implementazione

I client di MOOS vengono implementati estendendo la classe CMOOSApp; il costruttore della classe MOOSApp legge il file "impostazioni.moos" e imposta i parametri di interesse.

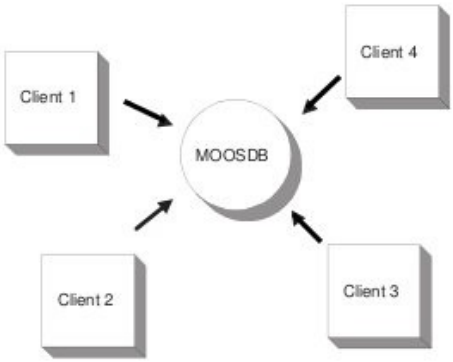

Fig 49: Il meccanismo di comunicazione "star shaped" dei nodi di ROS

- bool OnStartUp(): viene chiamata subito dopo il costruttore, solitamente viene utilizzata per inizializzare le variabili e impostare l'accesso alle periferiche hardware;
- bool OnConnectToServer():viene chiamata al momento della connessione con il server MOOSDB, solitamente viene utilizzata per "sottoscrivere" il processo alle variabili di cui vuole ricevere gli aggiornamenti;
- bool OnNewMail(MOOSMSG_LIST &NewMail): Ogni volta che MOOSDB riceve un aggiornamento su una determinata variabile comunica la variazione a tutti i processi iscritti tramite questa funzione;
- bool Iterate(): viene eseguita con una predeterminata frequenza dal processo, specificabile tramite il file di impostazioni impostazioni.moos . Solitamente la vera e propria elaborazione eseguita dal processo viene inserita all'interno di questa funzione.

Nel seguitoi si riporta il codice C++ che implementa un processo integrabile in un architettura MOOS, in particolare si possono notare le definizioni delle funzioni appena descritte.

```
#include <MOOSLIB/MOOSApp.h>
#include "MOOSClient.h"

int main (int n, char ** av){
        MOOSClient applicazioneMoos;
        applicazioneMoos.Run("nomeClient","impostazioni.moos");
        return 0;
}
class MOOSClient : public CMOOSApp
{
 public:
        MOOSClient();
        virtual ~MOOSClient();

protected:
        bool OnNewMail(MOOSMSG_LIST &NewMail);
        bool Iterate();
        bool OnConnectToServer();
        bool OnStartUp();
        …
}
MOOSClient::MOOSClient(){
        var1 = 0;
        var2 = 1;
        ...
}        // costruttore della classe CMOOSApp
```

```
bool MOOSClient::OnStartUp(){
        //inizializzazione porte logiche e dati
return true;
}
bool MOOSClient::OnConnectToServer(){
        //Iscrizione alle variabili di interesse
}
bool MOOSClient::OnNewMail(MOOSMSG_LIST &NewMail){
        //ricezione messaggi
}
bool MOOSClient::Iterate(){
        //Elaborazione dati
}
```

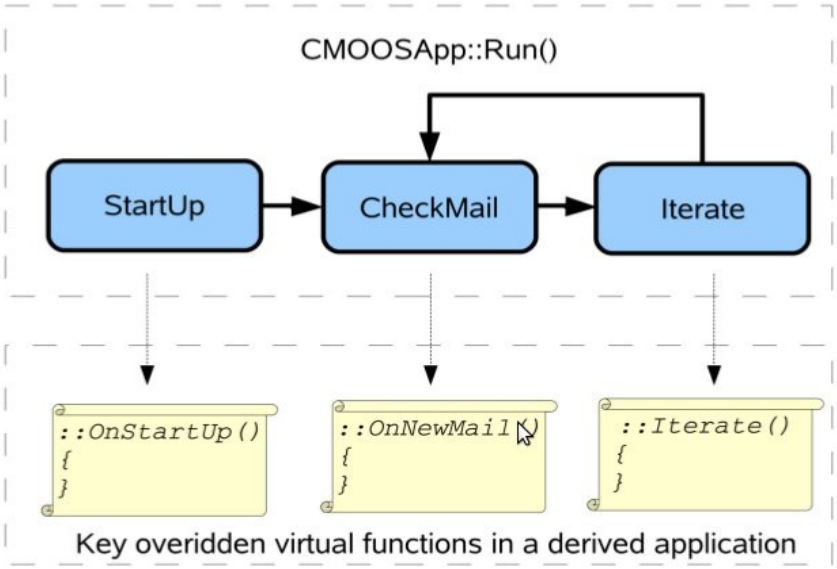

Fig 50: Meccanismo a eventi dei processi MOOS

3.3 Framework e Librerie

In questo paragrafo si riporta un breve elenco delle librerie e dei framework impiegati, focalizzandosi sui punti deboli/di forza degli stessi.

3.3.1 Il framework Qt

Qt (/ˈkjuːt/ "cute") è un framework multipiattaforma generalmente usato per lo sviluppo di software che necessita di una GUI (Graphical User Interface); viene inoltre utilizzato per sviluppare strumenti server anche in applicazioni CLI (Command Line Interface). Le librerie Qt sono fondate sul

linguaggio C++ e possono essere compilate su tutti i principali sistemi operativi (varie distribuzioni di linux, MAC OS X e Windows) e sono distribuite sotto licenza GPL.

Fig 51: Logo del framework Qt

Il framework Qt nasce sotto la Nokia come una suite di librerie per la telefonia mobile, in particolare per Symbian; nel tempo poi la compatibilità si è allargata alla quasi totalità dei SO mentre Symbian è stato deprecato. La portabilità delle Qt costituisce un vantaggio sotto molti aspetti ma ha condotto a inevitabili compromessi che, in determinati casi, rendono il codice poco performante dal punto di vista computazionale; soprattutto sotto l'aspetto grafico le Qt non hanno prestazioni paragonabili ad altri framework come OpenCv, OpenGL, Cuda o Visual C++.

Le librerie Qwt

Il termine Qwt è un acronimo di Qt Widgets for Technical applications e indica un insieme di oggetti, impiegati nello sviluppo delle interfacce grafiche per il software dei programmi tecnico scientifici.

3.3.2 Le librerie OpenCv

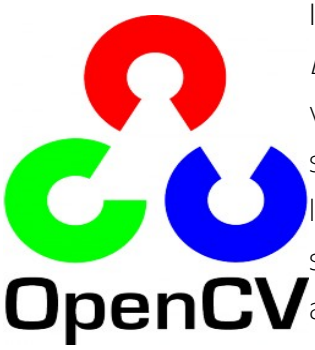

Fig 52: Logo delle OpenCv

Il termine OpenCV (*Open Source Computer Vision Library*) indica un insieme di librerie incentrate sulla visione computazionale per applicazioni real-time; sono sviluppate da intel ma sono open source sotto licenza BSD. Si tratta di librerie multipiattaforma sviluppate prevalentemente in C++ che comprendono anche alcuni componenti C; uno degli aspetti che ha determinato la fortuna di queste librerie è l'interfaccia CUDA-based per la GPU che permette un'elaborazione accelerata dei dati avvalendosi del chipset grafico del PC.

3.4 Sistema operativo

Linux Ubuntu / Debian

Fig 53: Logo di Ubuntu

Fig 54: Logo di Debian

Ubuntu è un sistema operativo basato sulla distribuzione linux Debian con un proprio ambiente desktop, il nome deriva da un'antica parola africana (zulu, ma anche xhosa) che significa "umanità verso gli altri", oppure "io sono ciò che sono per merito di ciò che siamo tutti". Per il progetto SAUC-e 2013 si è impiegato Ubuntu nella versione > 10.10 per il pc Commell e Raspbian (che è una distribuzione di Debian) per il Raspberry Pi.

Fig 55: logo di Raspbian

Per il controllo remoto di questi sistemi ci si avvale dei programmi standard di controllo remoto come SSH e VNC. La scelta è ricaduta su questi sistemi per la loro scalabilità e per la ampia disponibilità di software gratuito e ampiamente documentato; inoltre la quasi totalità dei framework d'interesse garantisce completa compatibilità con Linux Ubuntu.

4 Algoritmo di stima della posizione

In questo capitolo saranno esaminate le modalità di stima della posizione del veicolo. L'articolo [7] riporta un sistema di stima della posizione realizzato per il SAUCE 2012 dal team dell'università di Lubecca; in questo lavoro di tesi si ripropone un sistema simile ma con l'aggiunta di un modello cinematico del veicolo e con alcune variazioni .

4.1 Elenco ed analisi delle ipotesi

Nel paragrafo seguente vengono elencate le ipotesi sulla base delle quali l'algoritmo è stato realizzato, inoltre si valuta il livello di soddisfacimento di ognuna di queste.

Ipotesi

- L'asse di rotazione del SONAR deve essere parallelo alle pareti:
  - ipotesi non del tutto soddisfatta dato che il veicolo avrà delle oscillazioni lungo l'asse di rollio e beccheggio. Tuttavia la dinamica del veicolo lungo tali assi è così lenta a causa dell'attrito che tali oscillazioni risultano trascurabili.
- Errore trascurabile di stima dell'imbardata del veicolo:
  - prove sperimentali hanno dimostrato che l'errore di stima dell'orientazione è molto ridotto e può essere considerato trascurabile data l'elevata rumorosità del sensore sonar.
- Errore trascurabile di misura dell'encoder dell'azionamento che permette la rotazione dell'echosounder:
  - il servo Robotis Dynamixel ax-18 ha un errrore di inseguimento a vuoto di $300°/1024 \simeq 0.3°$; con la presenza del carico l'errore aumenta ma resta comunque molto ridotto.
- velocità del veicolo costante in modulo
  - ipotesi non del tutto soddisfatta, la velocità del veicolo non dovrebbe superare $0.5 m/s$.
- topologia dell'ambiente nota.
  - ipotesi sostanzialmente soddisfatta.

4.2 Algoritmo di localizzazione

L'algoritmo di localizzazione trattato in queste pagine è il risultato di una serie di esperimenti condotti nella piscina dell'MDM Lab di Pistoia ed è incentrato sugli scopi della competizione SAUC-e; il risultato quindi è un algoritmo fortemente specifico, per molti aspetti diverso dagli algoritmi normalmente impiegati per i sistemi di stima della posizione che solitamente si basano su tecniche SLAM.

La stima della posizione si suddivide in 5 fasi:

1. Ricezione dell'eco e stima della direzione dello stesso:
   - tale fase viene eseguita via software e consiste nell'accoppiamento dei dati prodotti dal SONAR e dalla piattaforma inerziale.
2. Estrazione della distanza parete-veicolo:
   - In questa fase l'eco ricevuto viene analizzato in modo da estrarne la distanza parete-veicolo.
3. Aggiornamento del buffer dati:
   - i dati vengono inseriti in un buffer che viene aggiornato ad ogni istante di campionamento. Tale buffer costitusce l'insieme dei dati a disposizione dell'algoritmo.
4. Misura della posizione del veicolo
   - La misura della posizione può essere effettuata con due sistemi: un metodo geometrico ed uno basato sulla trasformata di Hough.
5. Filtro di Kalman esteso
   - Le misure di posizione vengono filtrate ricorsivamente tramite un filtro di Kalman esteso che impiega un modello cinematico del veicolo basato sull'ipotesi di velocità costante.

Nei paragrafi successivi le varie fasi dell'elaborazione vengono descritte nel dettaglio.

4.2.1 Ricezione dell'eco e stima della direzione del beam

Si tratta di una fase complessa dal punto di vista software che viene eseguita secondo le modalità descritte nel capitolo 5. I dati elencati vengono inseriti in una struttura dati che poi verrà elaborata nelle fasi successive:

- Eco ricevuto dall'echosounder:
  - L'eco di ritorno viene campionato in 250 punti secondo le modalità indicate nel paragrafo 1.1.
- Direzione del beam dell'echosounder:
  - La direzione del beam è equivalente alla somma dell'angolo di imbardata del veicolo e della posizione dell'azionamento che movimenta l'echosounder.

In questo unico messaggio vi sono tutti i dati necessari al sistema di localizzazione, in particolare di seguito si riporta il messaggio ROS dedicato:

"rosImagenexData.msg"

```
int16 counter              //contatore messaggio
int8[252] echodata         //campionamento dell'eco ricevuto
int16 angle                //direzione echo in terna fissa
int16 imaginexdistance     //calcolo distanza proprietario
int8 minrange              //parametri di misura
int8 maxrange              //
int8 gain                  //
int8 pulselength           //
bool fake                  //flag che indica se si tratta di
                           un pachetto generato da una
                           simulazione
int16 poolheight           //dimesioni piscina simulata
int16 poolwidth            //
int16 walldistance         //distanza esatta parete virtuale
int16 virtualposx          //posizione virtuale veicolo
int16 virtualposy          //
```

Tale messaggio contiene anche i parametri caratteristici della simulazione, in particolare la posizione virtuale del veicolo.

4.2.2 Misura della distanza parete-veicolo

Prove empiriche hanno evidenziato che l'eco della parete è equivalente a

quello di un'onda quadra caratterizzata dai due parametri t_1 e t_2 : t_1 rappresenta il tempo di volo del segnale mentre t_2 è la durata di tale eco.

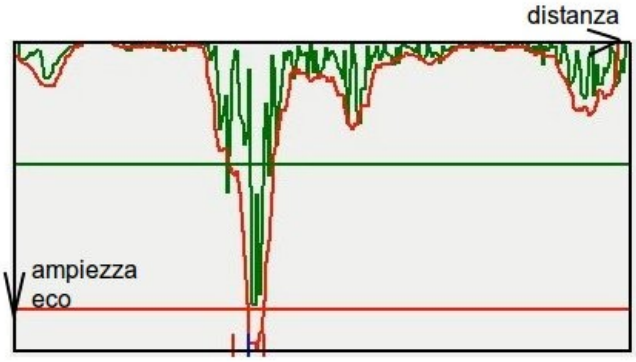

Fig 56: Eco ricevuto nel caso di beam orientato in direzione di una parete: la linea verde indica l'eco ricevuto mentre la linea rossa indica la funzione di crosscorrelazione calcolata tra l'eco ricevuto ed un'onda quadra caratterizzata da parametri t1 = 0 e t2 = 0.3 m

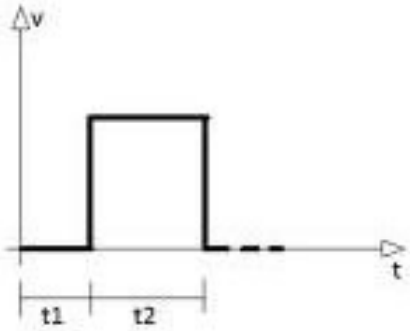

Fig 57: Onda quadra caratterizzata da parametri t1 e t2.

Per misurare la distanza parete-veicolo si è implementato un sistema basato sulla funzione di crosscorrelazione, che per comodità riportiamo:

$$R_{fg}(t)=\int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

Nei sistemi di telerilevamento si utilizza spesso un sistema di questo tipo: viene effettuata una crosscorrelazione tra il segnale inviato ed il segnale ricevuto e la funzione ottenuta presenta un picco in prossimità del tempo di volo. In questo caso si impiega un approccio simile: la crosscorrelazione viene effettuata tra l'eco ricevuto e l'eco della parete della piscina e la crosscorrelazione avrà un picco in prossimità della distanza parete-veicolo.

4.2.3 Filtro euristico

A partire dai dati a disposizione, tramite un filtro euristico, si stima la distanza parete-veicolo. La filosofia è quella di eliminare tutti i casi dubbi e concentrarsi su poche misure affidabili, Il filtraggio si fonda sulle seguenti ipotesi :

- assenza di eco dopo la parete;
- Eco di una parete equivalente ad un'onda quadra.

La funzione di crosscorrelazione viene analizzata da lontano verso vicino, il primo campione che soddisfa tutte le condizioni sotto elencate corrisponde alla distanza stimata della parete:

- L'intensità della crosscorrelazione deve essere superiore ad una soglia predeterminata(soglia 2).
- Non ci devono essere picchi oltre la parete che superino una soglia predeterminata (soglia 1)

Il successo di questa fase è fortemente vincolato all'ampiezza delle soglie 1 e 2, infatti queste sono direttamente accesibili tramite degli oggetti slider nella GUI corrispondente (vedi paragrafo 5.4.3).

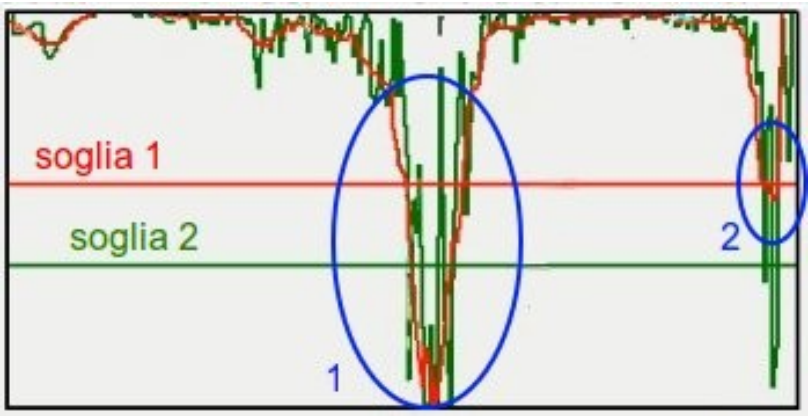

Fig 58: La linea verde corrisponde al segnale campionato dall'echosounder mentre la linea rossa corrisponde alla crosscorrelazione calcolata secondo le modalità indicate nel paragrafo 4.2.2. Le linee orizzontali indicano le soglie 1 e 2: in questo caso il segnale viene scartato poiché la parete viene identificata nel punto 1 ma è presente un picco oltre la parete nel punto 2.

4.2.4 Aggiornamento buffer dati

Una volta elaborati i dati secondo le modalità appena descritte, questi vengono inseriti in un buffer che rappresenta tutta l'informazione disponibile per la misura della posizione. Il buffer dati è costituito da n oggetti del tipo:

$$m_k = [d_k, \theta_k]$$

Ogni oggetto m_k è riferito ad un certo istante di tempo e contiene:

- distanza veicolo-parete d_k
- direzione dell'eco θ

La dimensione del buffer è prefissata: ad ogni iterazione il campione più vecchio viene scartato mentre il più recente viene aggiunto.

$$\begin{bmatrix} d_k & \theta_k \\ d_{k-1} & \theta_{k-1} \\ \vdots & \vdots \\ d_{k-n} & \theta_{k-n} \end{bmatrix} \begin{matrix} \leftarrow [d_k, \theta_k] \\ \\ \rightarrow [d_{k-n-1}, \theta_{k-n-1}] \end{matrix}$$

I dati contenuti nel buffer possono essere impiegati per generare un'immagine che rappresenta l'ambiente circostante il veicolo, tale immagine equivale ad una matrice $A^{M \times N}$. Nel seguito si riporta lo pseudocodice relativo alla creazione dell'immagine:

$$\begin{aligned} & se(k < n) \\ & \qquad m_k = M/2 + \cos(\theta_k) d_k \\ & \qquad n_k = M/2 + sen(\theta_k) d_k \\ & \qquad A(m_k, n_k)_k = A(m_k, n_k)_k + \beta_k \\ & \qquad k = k + 1 \end{aligned}$$

Dove:

$n = dimensione\ del\ buffer$
$M = dimensione\ matrice$
$[d_k, \theta_k, x_{k|k-1}, y_{k|k-1}] = dati\ estratti\ dal\ k-esimo\ elemento\ del\ buffer$
$\beta_k = parametro\ che\ pesa\ i\ campioni\ rispetto\ al\ tempo$

La matrice ottenuta può essere plottata come un'immagine in cui ad ogni cella corrisponde un cerchio centrato nelle coordinate della cella stessa.
Generare un'immagine rappresentativa del buffer non è indispensabile poiché non è necessario ai fini dell'esecuzione dell'algoritmo; tuttavia è fondamentale per l'utente soprattutto in fase di taratura dei parametri, permettendo un feedback grafico facilmente intellegibile.

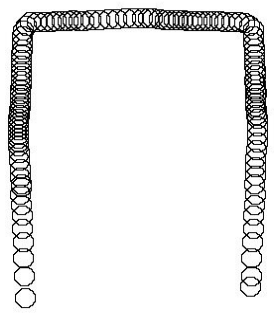

Fig 59: Immagine rappresentativa del buffer nel caso di una simulazione ambientata nel campo gara del CMRE

4.2.5 Misura della posizione

In questo paragrafo viene descritta la fase più importante dell'algoritmo, ovvero la misura della posizione.

4.2.5.1 Metodo geometrico

Si tratta del metodo più semplice e più utilizzato dai team nell'ambito della competizione SAUC-e: conoscendo la distanza da due pareti ortogonali della piscina e mediante banali relazioni geometriche, è possibile risalire alla posizione del veicolo. Questo sistema considera solo le misure la cui direzione è ortogonale ad una delle pareti e quindi gran parte delle misure

viene scartata.

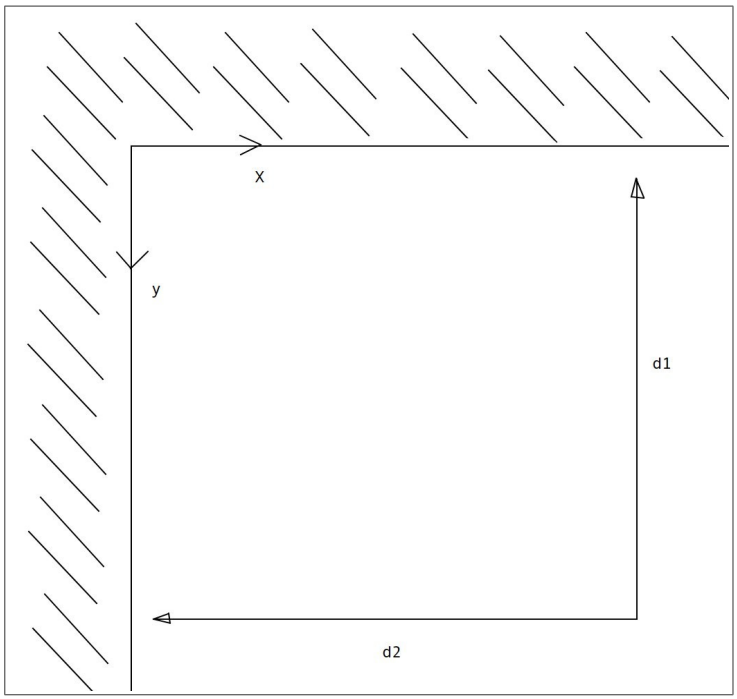

Fig 60: Schema del metodo geometrico

Vantaggi:

- semplicità di implementazione;
- robustezza elevata.

Svantaggi:

- Frequenza di aggiornamento bassa;
- Impiega poche misure e quindi non sfrutta tutta l'informazione disponibile.

4.2.5.2 Metodo basato sulla trasformata di Hough

La trasformata di Hough è stata descritta precedentemente; come mostrato nel paragrafo 1.5 tale trasformata si basa su una procedura di voting che permette di valutare la verosimiglianza di ogni coppia di parametri; segue la ricerca dei massimi locali che corrispondono alle linee effettivamente presenti nell'immagine.

Le figura 61 riporta i dati acquisiti nel corso di una simulazione del campo di gara del CMRE, la figura 62 contiene i dati relativi a 200 misure ed una rotazione dell'echosounder di 360°.

Come si nota dalla figura 62, la trasformata di Hough presenta 3 picchi (cerchiati di rosso); dato che la direzione delle 3 pareti è nota la ricerca dei massimi non avviene su tutto lo spazio ma solo sulle linee relative a 0° e 90° (nelle immagini riga 1 e riga 2).
Il numero di misure elevato (200 misure) garantisce un livello di precisione e di robustezza maggiore rispetto al caso del metodo geometrico.

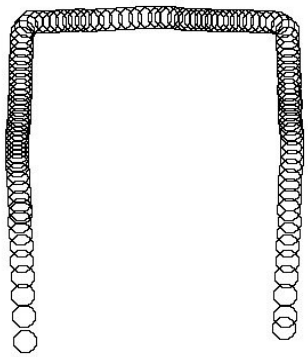

Fig 61: Buffer Dati relativo ad una simulazione ambientata nel campo gara del CMRE

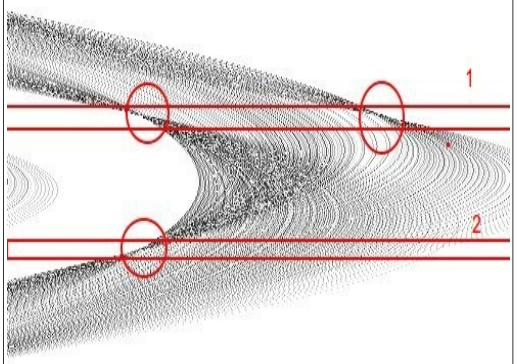

Fig 62: Trasformata di Hough applicata al buffer delle misure: punti piu scuri corrispondono a valori maggiori. I massimi dello spazio parametrico sono cerchiati di rosso.

Vantaggi

- Robustezza elevata;
- Impiego di tutta l'informazione disponibile;

Svantaggi:

- Implementazione complessa.

4.2.6 Filtraggio ricorsivo

L'ultimo passo dell'algoritmo di localizzazione è il filtraggio ricorsivo delle misure di posizione; tale filtraggio viene eseguito da un filtro di Kalman esteso. Lo stato del sistema comprenderà unicamente la posizione del

veicolo (espressa nel sistema di riferimento indicato nel paragrafo 1.4) ed il modulo della velocità.

$$\hat{\boldsymbol{X}}_k = \begin{bmatrix} \hat{\boldsymbol{x}}_k \\ \hat{\boldsymbol{y}}_k \\ |\boldsymbol{v}_k| \end{bmatrix}$$

Dove $[\boldsymbol{x}_k, \boldsymbol{y}_k, |\boldsymbol{v}_k|]$ indica la posizione del veicolo all'istante k mentre $|v_k|$ indica il modulo della velocità. Come indicato nel paragrafo 4.1 si ipotizza che la velocità del veicolo sia costante e quindi vale la relazione:

$$|\boldsymbol{v}_{k+1}| = |\boldsymbol{v}_k| \quad \forall k$$

Il modello del sistema è un modello cinematico che può essere indicato nel seguente modo:

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k+1} \\ \hat{\boldsymbol{y}}_{k+1} \\ |\boldsymbol{v}_{k+1}| \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cos(\theta_k) \\ 0 & 1 & sen(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}_k \\ \hat{\boldsymbol{y}}_k \\ \boldsymbol{v} \end{bmatrix} + \begin{bmatrix} \boldsymbol{q}_k \\ \boldsymbol{w}_k \\ 0 \end{bmatrix}$$

Dove $\{\boldsymbol{q}_k\} = N(0, \sigma_q^2)$ e $\{\boldsymbol{w}_k\} = N(0, \sigma_w^2)$ con $q \perp w$; θ indica l'angolo di imbardata del veicolo e $\boldsymbol{v}_k$ la velocità dello stesso. Le equazioni di misura invece saranno:

$$\boldsymbol{m}_k = \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{y}_k \end{bmatrix} + \begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{r}_k \end{bmatrix}$$

Dove $\{\boldsymbol{u}_k\} = N(0, \sigma_u^2)$ e $\{\boldsymbol{r}_k\} = N(0, \sigma_r^2)$ con $\boldsymbol{u} \perp \boldsymbol{r}$. Date queste premesse è possibile scrivere le equazioni di aggiornamento di stato e di misura, definiamo:

$$Q = \begin{bmatrix} \sigma_q & 0 \\ 0 & \sigma_w \end{bmatrix} \quad R = \begin{bmatrix} \sigma_u & 0 \\ 0 & \sigma_r \end{bmatrix} \quad F_k = F(\theta_k) = \begin{bmatrix} 1 & 0 & \cos(\theta_k) \\ 0 & 1 & sen(\theta_k) \\ 0 & 0 & 1 \end{bmatrix}$$

$$P_k = \begin{bmatrix} p_{1,1}^k & p_{1,2}^k \\ p_{2,1}^k & p_{2,2}^k \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Quindi avremo:

Equazioni di aggiornamento di misura:

$$K_k = \hat{P}_{k|k-1} C' (C \hat{P}_{k|k-1} C' + R)^{-1}$$
$$\hat{\boldsymbol{X}}_{k|k} = \hat{\boldsymbol{X}}_{k|k-1} + K_k (\boldsymbol{m}_k - C \hat{\boldsymbol{X}}_{k|k-1})$$
$$\hat{P}_{k|k} = \hat{P}_{k|k-1} - K_k (C \hat{P}_{k|k-1} C' + R) K_k'$$

Equazioni di aggiornamento dello stato:

$$\hat{x}_{k+1|k} = F_k \hat{x}_k$$
$$\hat{P}_{k+1|k} = F_k \hat{P}_k F_k' + Q$$

5 Realizzazione del software

5.1 Analisi degli obiettivi

Per comodità si riportano gli obiettivi analizzati nei paragrafi 2.1.3 e 2.2.3:

Obiettivo individuale SAUC-e

- Sviluppare i due processi ROS impiegati nella gestione dei dispositivi Xsens MTi e Imagenex echosounder. I due processi saranno indipendenti ma dovranno essere in grado di fondere le misure generando un'unica struttura dati che contenga l'orientazione del veicolo e la misura dell'echosounder relativi ad un determinato istante di tempo; tale struttura verrà inviata ad un processo client eseguito da un'altra macchina tramite un protocollo TCP IP;
- realizzare un algoritmo di stima della posizione basato sui dati di echosounder e e piattaforma inerziale che possa essere impiegato con successo nella competizione SAUC-e 2013;
- realizzare un processo eseguibile in remoto che permetta la messa a punto dell'algoritmo di localizzazione.

Obiettivo individuale Nemo

- Sviluppare i due processi MOOS impiegati nella gestione dei dispositivi Xsens MTi e Imagenex echosounder. I due processi saranno indipendenti ma dovranno essere in grado di fondere le misure generando un'unica struttura dati che contenga l'orientazione del veicolo e la misura dell'echosounder relativi ad un determinato istante di tempo; tale struttura verrà inviata ad un processo client eseguito da un'altra macchina tramite un protocollo TCP IP. Questi processi devono essere eseguiti sul Rasperry Pi in real time e quindi devono avere un costo computazionale contenuto;
- sviluppare un processo eseguibile in remoto che riceva la struttura dati appena descritta e la elabori in real time in modo da generare una immagine che rappresenti l'eco dell'ambiente in cui si trova il veicolo.

I due obiettivi hanno molti aspetti in comune, è importante scrivere un

codice che sia modulare e quindi composto da classi che possano essere assemblate con facilità. Tali classi devono avere le seguenti caratteristiche:

- Qgni classe deve essere indipendente dalle altre dal punto di vista della compilazione, in particolare deve includere tutte le sue dipendenze;
- ogni classe deve avere un buffer di ingresso e di uscita che renda possibile l'interfacciamento con le altre classi; l'accesso a tale buffer deve essere regolato da un semaforo (in modo che le due classi in comunicazione non vi accedano contemporaneamente). Solo le funzioni di lettura scrittura del buffer hanno visibilità pubblica, le altre devono avere visibilità protetta;
- ogni classe deve pubblicare su linea di comando informazioni relative al suo stato.

Per la realizzazione dei vari processi sono state sviluppate 5 classi che vengono "assemblate" a seconda degli scopi; ogni classe è caratterizzata da un thread indipendente.

- SerialHandler: si tratta della classe che implementa il protocollo di comunicazione seriale. Si tratta del codice di livello più basso e impiega numerose funzioni C che comunicano direttamente con il kernel del S.0 Linux Ubuntu. Tale classe è stata sfruttata per la gestione dei seguenti dispositivi:
    - Imagenex 852 ultra miniature echosounder;
    - Xsens MTi;
    - Pololu maestro 6;
    - Dynamixel ax-18.
- InterfaceHandler: si tratta di un thread che gestisce la comunicazione tra i buffer di ingresso-sucita delle varie classi;
- GuiHandler: si tratta della classe che gestisce la GUI (Graphical User Interface);
- TCPHandler: si tratta della classe che gestisce la trasmissione dei dati tra i vari processi implementati; anche nel caso in cui si trovino su computer diversi;
- MoosInterface: si tratta della classe che implementa il protocollo di comunicazione del middleware MOOS;
- ROSInterface: si tratta della classe che implementa il protocollo di comunicazione del middleware ROS.

5.2 Implementazione delle classi

In questo paragrafo analizzeremo con un livello dettagliato gli aspetti legati alle singole classi esaminando vantaggi e compromessi delle scelte fatte. Ogni classe sarà accompagnata da una breve descrizione del suo funzionamento e dei suoi aspetti più importanti, inoltre ne verrà fornito il grafico UML. Il termine UML sta per Unified Modeling Language o "linguaggio di modellazione unificato"e indica un linguaggio object-oriented impiegato nella schematizzazione grafica del software; nella letteratura di settore costituisce uno standard de-facto per la schematizzazione del software.
Nasce nel 1966 per opera di Grady Booch, Jim Rumbaugh e Ivar Jacobson (detti "i tre amigos") e nel tempo si è affermato fino all'ufficializzazione da parte di OMG della versione 2.0 nel 2004 anche se negli anni successivi ha continuato ad evolversi. L'UML svolge un'importante funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti, gran parte della letteratura di settore usa UML per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile ad un vasto pubblico.

5.2.1 Classe SerialHandler

La classe serialHandler è la classe che gestisce la comunicazione con i dispositivi tramite porta seriale; in particolare gestisce la trasmissione dei dati con i seguenti :

- Imagenex echosounder;
- Pololu Maestro 6;
- Xsens MTi;
- Dynamixel ax-18.

Ogni reimplementazione è basata sul protocollo di comunicazione del rispettivo dispositivo, gli altri aspetti del codice restano invariati.

Device managment in linux
La gestione delle porte di I/O del pc è uno degli aspetti che contraddistingue maggiormente i sistemi operativi, in linux tutti i dispositivi rilevati dal kernel

vengono inseriti nella cartella virtuale **/dev**, e vengono acceduti come dei file, in particolare i dispositivi seriali vengono inseriti nella cartella **/dev/serial/** e vengono ordinati per indirizzo in **/dev/serial/by-address** o per id in **/dev/serial/by-id**.

Nel seguito si riporta l'header della classe che aiuta a capire l'effettiva organizzazione delle funzioni ed i tipi dei dati.

SerialHandler.h

```
#include <unistd.h>
#include <fcntl.h>
#include <asm-generic/ioctls.h>



class serialUSB {
public:
    serialUSB();
    virtual ~serialUSB();
    bool Start(int freq, string dev);
    sensorData Read()
    sensorData Data;
    sensorParam Param;
    void setParam(void)
protected:
    int counter;
    bool init(const char * dev, int Freq);
    void parseData(char* , int);
    int set_port(int port);
    int open_port(const char * dev);
    pthread_t scrittura;
    pthread_t lettura;
    sensorData *Data;
    sensorData setting;
 }
```

La classe esegue una procedura prefissata, si accede al passo successivo solo se il precedente è stato portato a buon fine:

1. La classe viene creata da un'altra classe gerarchicamente più alta;
2. il membro Start() viene chiamato e quindi vengono fissati indirizzo del dispositivo e frequenza di lettura (massima);
3. viene chiamato il metodo init() che verifica se il dispositivo è effettivamente collegato e se si dispone dei privilegi necessari per

accedervi;

4. il metodo open_port() apre la porta interessata e set_port() ne setta i parametri e modalità di accesso;
5. vengono avviati i thread di lettura-scrittura che interagiscono direttamente con la porta seriale:
   - il thread di scrittura effettua un determinato numero di richieste al secondo al sensore fornendo i parametri di misura impostati;
   - il thread di lettura attende che ci siano dati nel buffer del kernel relativo alla porta e verifica che non ci siano errori tramite l'analisi del prefisso a inizio messaggio e del checksum al termine. I dati letti, sotto forma di array di caratteri, vengono inviati alla funzione SensorData.

Le classi gerarchicamente più alte possono comunicare con serialHandler() tramite la funzione Read() che restituisce l'ultima struttura dati generata da parseData() e con setParam() che imposta i parametri del sensore.

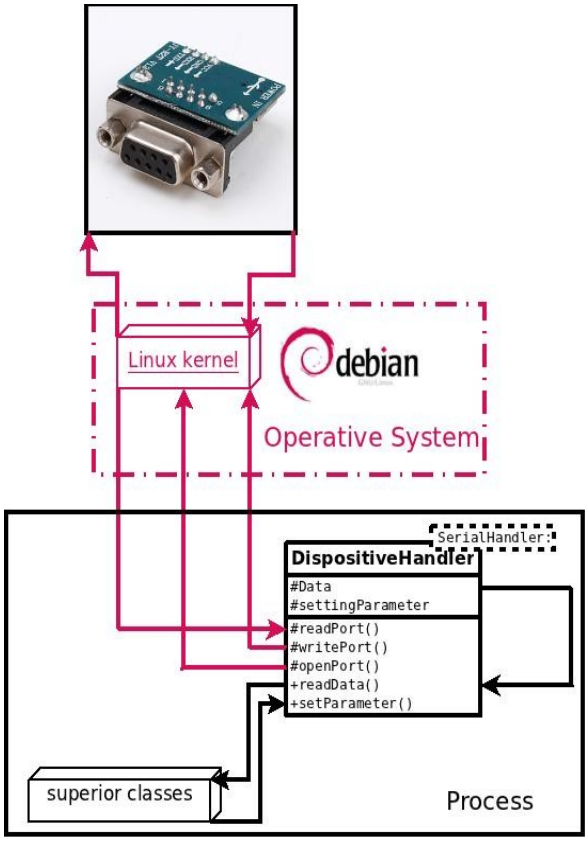

Fig 63: UML della classe SerialHandler

5.2.2 Classe TcpHandler

TcpHandler è la classe impiegata per la comunicazione tra i processi, si distingue dalle classi MoosHandler e RosHandler poiché è totalmente indipendente e non richiede l'ausilio di alcun meta sistema operativo. La comunicazione implementa il protocollo TCP.

Il protocollo TCP
TCP è un protocollo di comunicazione orientato alla connessione, ovvero lo scambio di dati è preceduto da una fase di "handshake" nella quale mittente e destinatario negoziano una connessione che rimane attiva anche in assenza di scambio di dati e che viene esplicitamente chiusa quando non più necessaria. Il servizio offerto dal TCP è il trasporto di un flusso di byte bidirezionale tra due applicazioni in esecuzione su host differenti (ma anche sullo stesso host); si tratta di un protocollo full-duplex in quanto permette di trasmettere contemporaneamente nelle due direzioni. Il protocollo, tramite meccanismi di timeout e di acknowledgment, garantisce che i dati trasmessi, se giungono a destinazione, lo facciano in ordine ed una volta sola "at most once". Tcp contiene strumenti per il controllo della congestione e fornisce un servizio di multiplazione delle connessioni ad un host tramite il meccanismo delle porte.

Tale protocollo presenta numerosi vantaggi:

- elevata flessibilità:
- compatibile con ogni sistema operativo;
- servizio di riinvio dei pacchetti;
- algoritmi anti congestione implementati a basso livello;
- totale disaccoppiamento tra le due parti in comunicazione (possono trovarsi su SO diversi);
- latenza molto bassa nel caso di reti locali;

La classe TcpHandler è in pratica costituita da due classi che però dal punto di vista implementativo sono indivisibili:

- TcpClient;
- TcpServer data.

Tali classi sono un estensione delle classi QAbstractSocket e QTcpServer che

fanno parte della libreria QNetwork del framework Qt; le funzionalità di queste due classi sono ampiamente trattate nella documentazione allegata ed in questa sede se ne discuteranno solo gli aspetti fondamentali.
La comunicazione può essere gestita in *synchronous mode* o *asynchronous mode:* l'approccio sincrono si basa sul blocco del flusso di esecuzione fino al verificarsi di una determinata condizione, l'approccio asincrono invece si basa su un meccanismo a eventi: il verificarsi di un evento genera un segnale che forza l'esecuzione di una funzione.

È stato scelto l'approccio asincrono per la classe TcpClient e sincrono per la classe TcpServer:

- classe TcpClient: può connettersi ad un altra classe TcpServer e può riceve dati da essa;
- classe TcpServer: accetta una connessione da una classe TcpClient e può inviare dati ad essa.

Perchè la comunicazione sia bidirezionale, ognuno dei processi coinvolti deve contenere entrambe le classi, la comunicazione Tcp è suddivisa in più fasi:

- Macchina server:
  - vengono istanziate le classi TcpServer e TcpClient;
  - viene rilevato l'indirizzo ip della macchina ;
  - la classe TcpServer viene messa in ascolto sull'indirizzo ip rilevato (porta 60000).
- Macchina client:
  - Vengono istanziate le classi TcpServer e Tcpclient;
  - Viene rilevato l'indirizzo ip della macchina;
  - La classe TcpServer viene messa in ascolto sull'indirizzo ip rilevato (porta 50000);
  - La classe TcpClient si connette ad un indirizzo ip statico o fornito dall'utente (che deve essere l'indirizzo della macchina server);
- Macchina Server:
  - Viene rilevata e accettata una richiesta di connessione;
  - La classe TcpClient si connette all' indirizzo ip ed alla porta che ha effettuato la richiesta.

Le classi gerarchicamente superiori a TcpServer possono inviare dati aggiungendoli al buffer di TcpServer, le classi gerarchicamente superiori a

TcpClient possono leggere i dati svuotando il buffer di TcpClient. Nel seguito si riportano gli header delle sue classi: è possibile vedere che la comunicazione è stata implementata impiegando un numero minimo di funzioni e che la memoria impiegata per i dati viene allocata tramite un oggetto QList che, impiegando l'allocazione dinamica della memoria, assicura un impiego ottimizzato della stessa.

tcpclient.h

```
#include <Qt/QtNetwork>
#include <QList>
#include <imagenex.h>

class TcpClient : public QThread
{
    Q_OBJECT
public:
    Client(QObject *parent = 0);
    ~Client();
    void connectionRequest(QHostAddress saddress, int port);
    Data ReadBuffer();
    QHostAddress ipFinder();
protected:
    void run();
    bool connected;
    QList<EchoData> readBuffer;
    QHostAddress ServerAddress;
    QHostAddress ClientAddress;
    int serverPort;
    int clientPort;
};
```

tcpserver.h

```
#include <Qt/QtNetwork>
#include <QList>
#include <imagenex.h>
class Server : public QTcpServer
{
    Q_OBJECT
public:
    TcpServer(QObject *parent = 0);
    ~Server();
    void listenConnection(QHostAddress saddress, int sport);
    QHostAddress ipFinder();
    void writeBuffer(EchoData data);
protected:
```

```
    QList<EchoData> writeBuffer;
    bool connected;
    QHostAddress serverAddress;
    QHostAddress clientAddress;
    int serverPort;
    int ClientPort;
    QTcpSocket socket;
    void writeBuff();
protected slots:
    void incomingConnection(int socketDescriptor);
```

Il seguente grafico UML riassume graficamente quanto appena descritto.

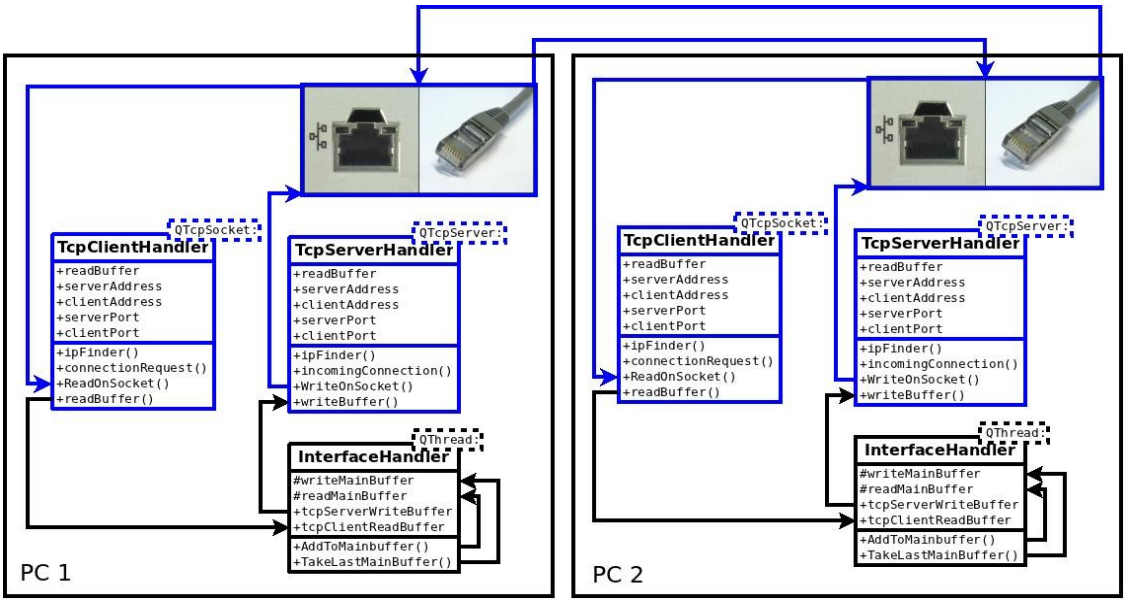

Fig 64: UML della classe TCPHandler

5.2.3 Classe MoosHandler

Si tratta della classe incaricata della comunicazione con il database di MOOS, come già mostrato nel paragrafo 3.2.2, tale classe si basa su un loop infinito e quindi deve essere istanziata da un thread indipendente (vedi figura 65) altrimenti genererebbe un blocco del flusso di esecuzione.

La classe reimplementa le funzioni già descritte nel paragrafo 3.2.2, si registra sui topic di interesse e ne riporta le variazioni sulle sue variabili locali che essendo pubbliche possono essere accedute liberamente: a differenza delle altre classi non c'è allocazione dinamica della memoria e quindi non è necessario regolare l'accesso ai dati tramite semafori. Nell'header sono riportate le modalità di istanziazione delle variabili:

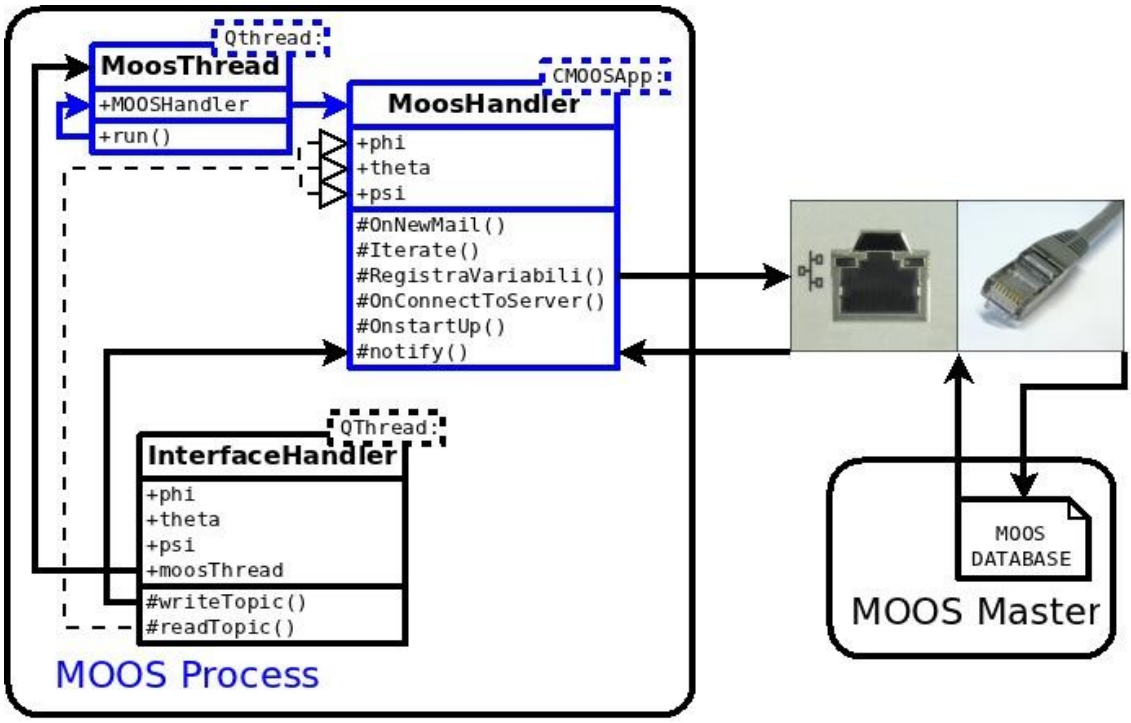

Fig 65: UML relativo alla classe MOOSHandler

```
#include <MOOSLIB/MOOSApp.h>
#include <QDebug>

class Moos : public CMOOSApp
{
public:
        XsensMoos();
        float data1;
        float data2;
        float data3;
protected:
        bool OnNewMail(MOOSMSG_LIST &NewMail);
        bool Iterate();
        bool OnConnectToServer();
        bool OnStartUp();
        void RegistraVariabili();
};
```

5.2.4 Classe RosHandler

Si tratta della classe dedicata alla comunicazione tra processi in contesto ROS, si tratta quindi dell'analogo di MoosHandler rispetto al middleware ROS.

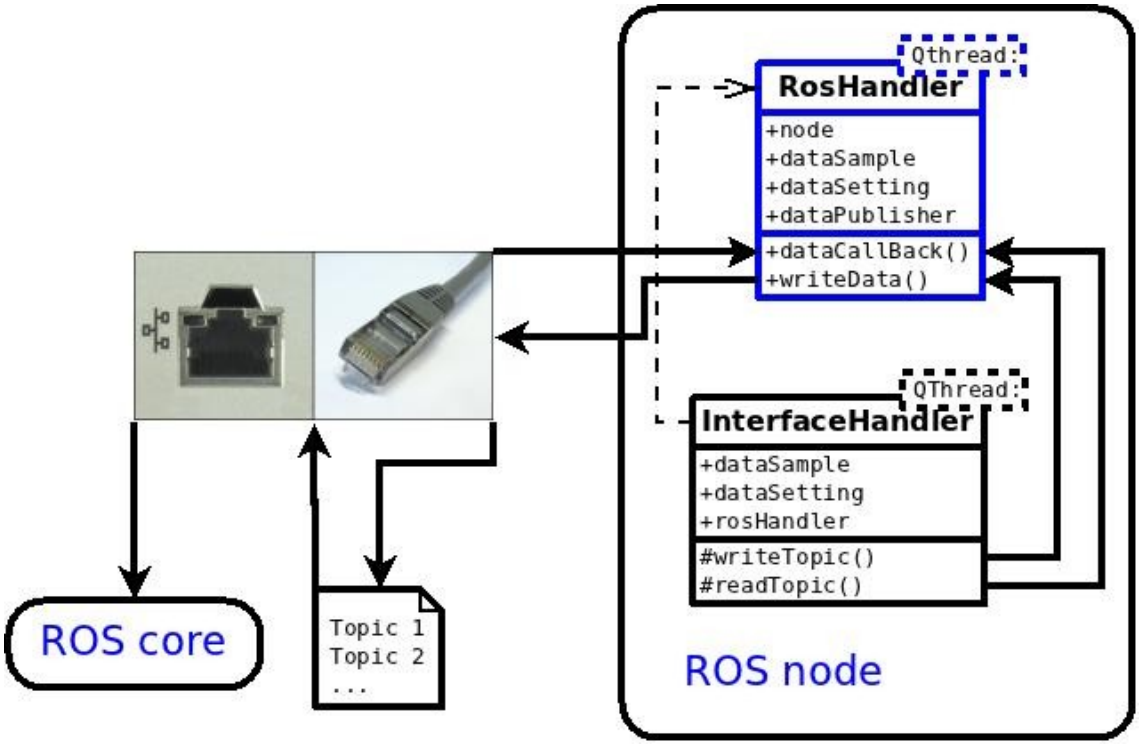

Fig 66: UML della classe ROSHandler

Le modalità di funzionamento di ROS sono già state descritte nel paragrafo 3.2.1; come nel caso di MOOS la ricezione dei messaggi in ROS si basa su un loop infinito implementato a basso livello. Anche in questo caso infatti, la classe ROSHandler deve essere istanziata da un thread indipendente; sotto il lato implementativo il meccanismo è semplice e si ripete per ogni topic:

- viene istanziato un oggetto `ros::Publisher` che permette di pubblicare su un determinato topic;
- viene dichiarata una callback che viene eseguita ogni volta che un messaggio viene pubblicato sul topic e riceve come argomento il puntatore al messaggio ricevuto.

Nel seguito si riporta l'header della classe che riporta gli oggetti appena descritti:

```
#ifndef ROSHANDLER_H
#define ROSHANDLER_H

#include <QList>
#include <QThread>

using namespace std;

class RosHandler: public QThread
```

```
{
    Q_OBJECT
public:
    explicit RosHandler(QObject *parent = 0);
    virtual ~RosHandler();

    ros::NodeHandle n;

    rospackage::Data DataSegment;
    QList<rospackage::Data>DataBuffer;
    ros::Publisher DataPublisher;
    void DataCallback(const
      rospackage::Data::ConstPtr& msg);
    void writeData(rospackage::Data msg);

    void run();

};

#endif  /* ROSHANDLER_H */
```

5.2.5 Classe InterfaceHandler

La classe InterfaceHandler ha lo scopo di mettere le varie classi in comunicazione tra loro. Per rendere sicura la trasmissione dei dati ogni classe deve avere un buffer di scrittura ed uno di lettura, tali buffer devono essere accessibili tramite funzioni pubbliche che implementino una logica thread safe di accesso ai dati. La classe InterfaceHandler è in grado di trasferire i dati tra le classi gerarchicamente inferiori, in particolare gestisce la comunicazione tra le classi descritte nei paragrafi precedenti che vengono istanziate nell'header di ImaginexInterface. Nel seguito si riporta l'header relativo ad un processo che legge i dati dalla classe `SerialHandler` e li visualizza con la classe `GUIHandler.`

InterfaceHandler.h

```
#include "serialhandler.h"
#include "guihandler.h"
#include <QDebug>
#include <QList>

class ImagenexInterface : public QThread
{
    Q_OBJECT
public:
```

```
        explicit Interface(QObject *parent = 0);
        SerialHandler sensor;
        GUIHandler GUI;
        void GUIHandlerWriteBuffer(GUIData data);
        GUIData GUIHandlerReadBuffer();
        void SensorHandlerWriteBuffer(SensorData data);
        SensorData SensorHandlerReadBuffer();
        QList<GUIData> ClientReadBuffer;
        QList<SensorData> XsensWriteBuffer;
        void run();
};
```

L'UML mostrato in figura 67 mostra il processo che implementa l'accoppiamento dei dati provenienti dalla piattaforma inerziale XSens MTi e dall'Imagenex 852 ultra miniature echosounder. I dati vengono in primo luogo letti tramite due estensioni delle classi serialHandler (XsensMTiHandler e Imagenex852Handler), in seguito sono fusi dalla classe InterfaceHandler e quindi trasmessi via rete tramite le due classi TcpHandler e TcpServer.

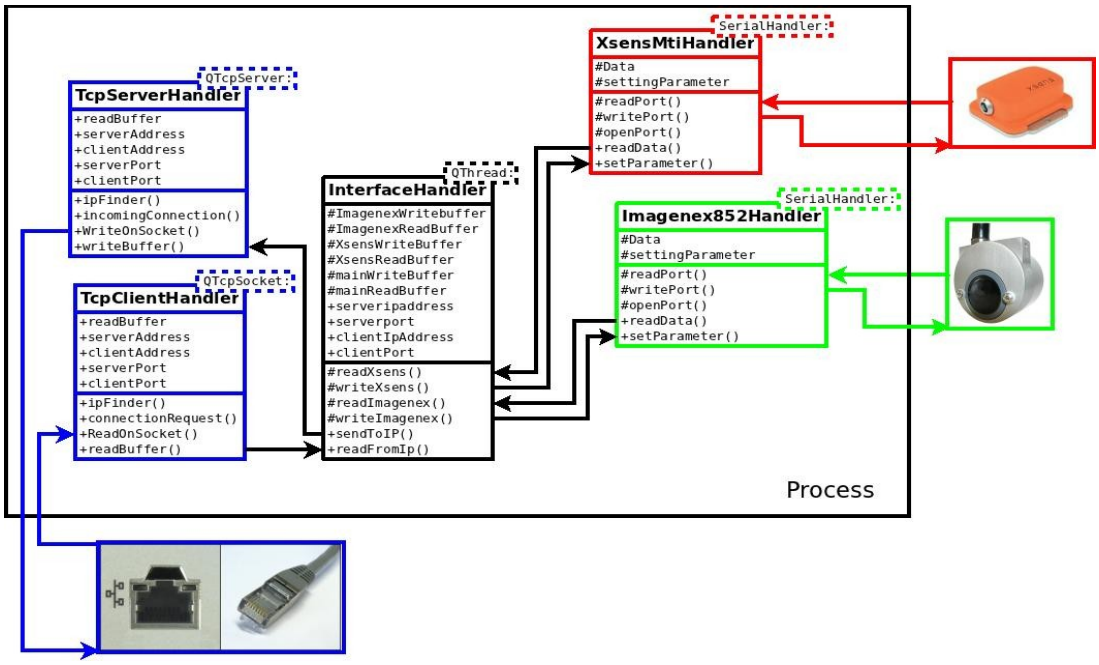

Fig 67: UML relativo ad un esempio di impiego della classe InterfaceHandler

5.3 Caratteristiche di alto livello del software

5.3.1 Schema UML Nemo

Le due immagini 68 e 69 riportano alcuni dei processi facenti parte del software di Nemo: i processi denominati con il termine "client" sono eseguiti sul computer di bordo del veicolo mentre quelli denominati con il termine "server" sono eseguiti sul computer dell'operatore. I processi sono troppo numerosi perche sia possibile una trattazione completa; in questa sede vengono trattati solo XsensServer, EchoServer ed EchoClient.

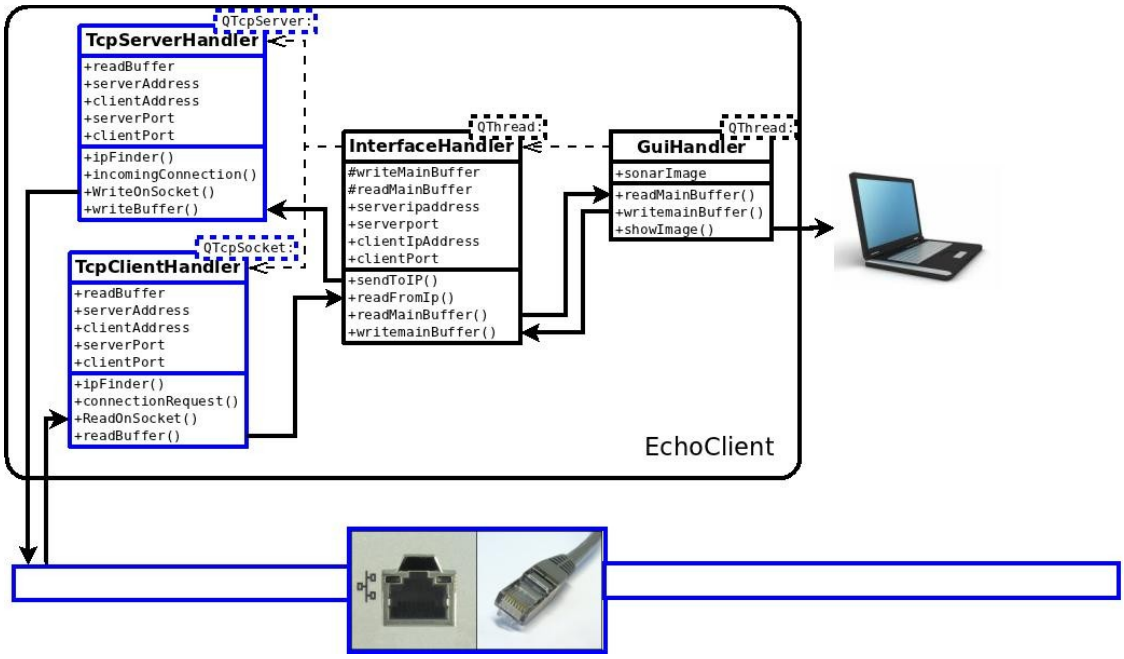

Fig 68: Schema UML del processo EchoClient

EchoClient

Si tratta del processo che gestisce l'interfaccia utente impiegata dal pilota del veicolo; i dati pubblicati dal processo EchoServer vengono ricevuti ed elaborati per ottenere un immagine acustica dell'ambiente circostante secondo le modalità già indicate nel paragrafo 1.2.

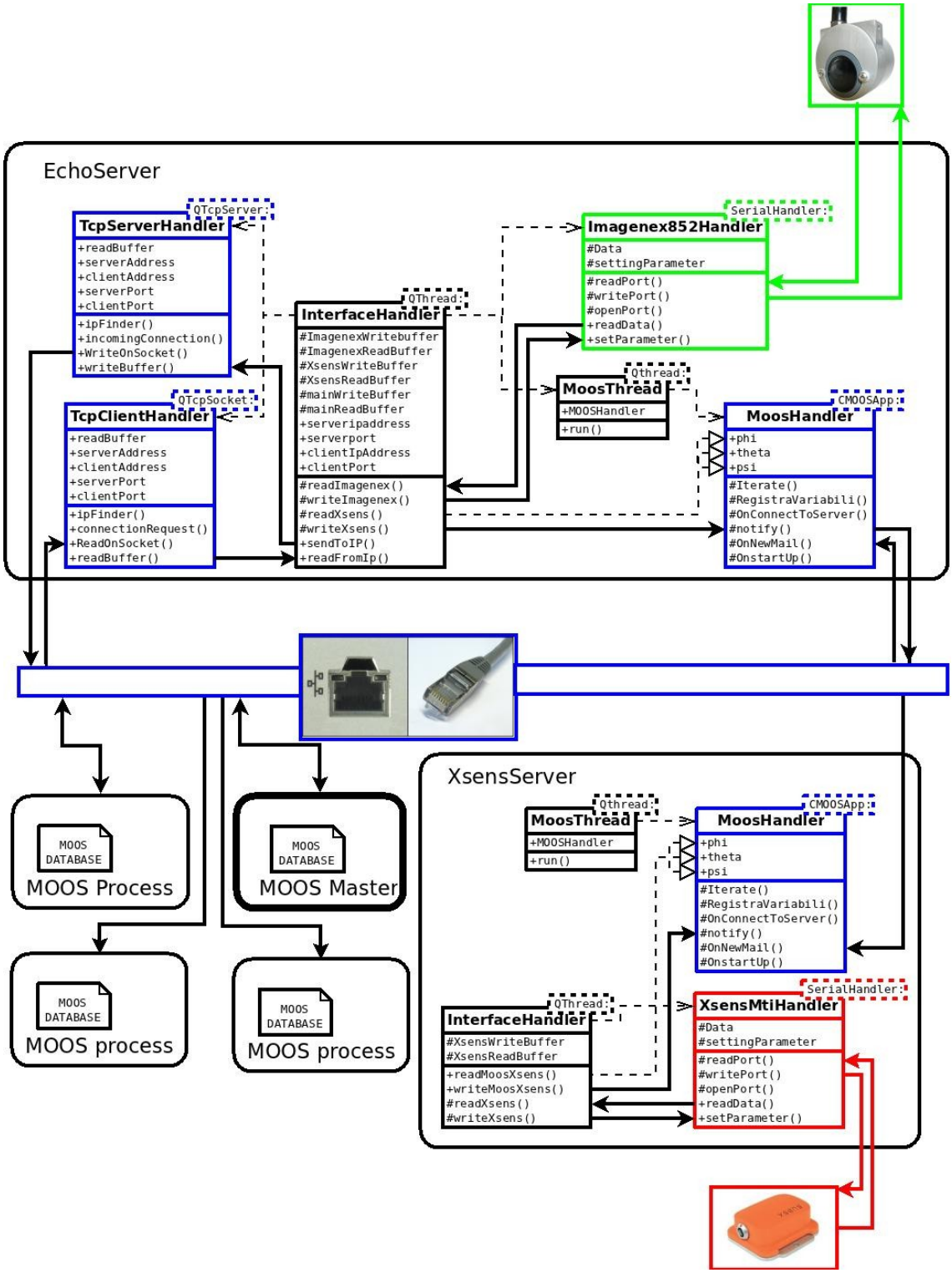

Fig 69: Schema UML del processo EchoServer

EchoServer

Si tratta del processo dedicato all'echosounder; anche in questo caso la comunicazione con il sensore viene effettuata tramite un'estensione della classe serialHandler, i dati letti vengono poi inviati agli altri processi tramite la classe TcpHandler.

XsensServer

Tale processo si basa sull'estensione della classe serialHandler, la sua funzione è quella di leggere i dati relativi all'orientazione del veicolo generati dall'IMU XSens ed in seguito pubblicarli sul database in modo da renderli disponibili agli altri processi.

5.3.2 Schema ROS di FeelHippo

Per esprimere in modo chiaro e semplice la topologia del sistema possiamo avvalerci del programma rqt_graph di ROS che permette di schematizzare con facilità i processi ed i messaggi che vengono scambiati tra i processi. Nel caso del sistema di localizzazione di FelHippo il software appena indicato plotta su schermo lo schema indicato in fig 70.

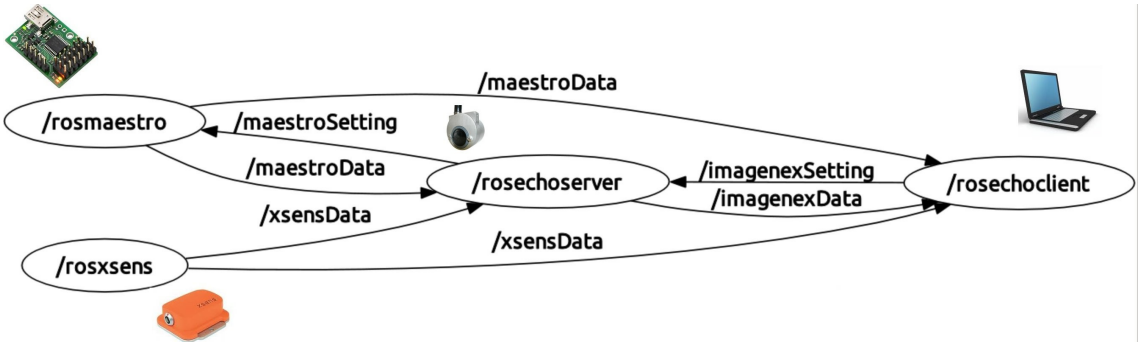

Fig 70: Schema ROS dei nodi impiegati nel sistema di localizzazione

Lo schema viene generato in automatico e riporta l'esatta topologia del sistema di localizzazione; i processi sono sviluppati in modo analogo rispetto quelli presenti su Nemo ma con la semplice differenza che la classe MoosHandler è sostituita con la classe RosHandler. Il sistema è composto da 4 processi detti nodi:

RosMaestro
Si tratta del nodo dedicato al dispositivo Pololu Maestro 6, tale dispositivo viene impiegato per il controllo degli azionamenti dei motori e del servo motore che viene impiegato per far ruotare l'echosounder.
Pubblicando un messaggio sulla topic dedicata, ogni nodo può settare il duty cycle di uno dei canali del dispositivo Pololu Maestro 6: per esempio pubblicando sulla topic dedicata un messaggio di tipo /MaestroSetting è possibile settare i parametri del dispositivo.

RosXsens
si tratta del nodo dedicato al dispositivo IMU Xsens MTi, i dati generati dal sensore vengono pubblicati su una topic dedicata e tutti i nodi del sistema possono accedervi.

RosEchoServer
Si tratta del Nodo dedicato al dispositivo Imagenex 852 ultra miniature echosounder, ha numerose funzioni; in particolare:

- Lettura, tramite porta seriale, dei dati generati dal sensore Imagenex 852;
- acquisizione dei dati provenienti dal sensore echosounder e dalla piattaforma inerziale;
- Invio dei dati su un topic dedicato.

RosEchoClient
È il processo che elabora i dati pubblicati dal nodo RosEchoServer: genera una immagine acustica dell'ambiente circostante e implementa l'algoritmo di localizzazione che stima la posizione del veicolo. Questo processo è stato sviluppato in due versioni: una con la GUI che permette ad un utente umano di osservare i dati (importante durante i test) ed una dotata di una essenziale CLI che invece sarà effettivamente impiegata nel veicolo durante l'esecuzione autonoma dei task.

5.4 Graphical user interface (Gui)

5.4.1 Aspetti software

La classe GuiHandler è la più onerosa dal punto di vista computazionale, purtroppo ad oggi le funzioni proprietarie di Qt basate sull'evento QPaintEvent non supportano l'accelerazione grafica e sono piuttosto povere sotto molti aspetti. Il framework Qt garantisce totale compatibilità con le librerie OpenGL che garantirebbero un frame rate migliore ed un minore uso della CPU; tuttavia si è scelto di non implementarle in questo lavoro a causa dell'elevata complessità di queste ultime.
Il termine Gui (Graphical User Interface) indica un tipo di interfaccia che permette all'utente del programma di interagire con la macchina tramite una serie di oggetti grafici convenzionali come barre e pulsanti; si tratta di uno dei punti di forza del Framework Qt che presenta notevoli potenzialità in

questo senso.

5.4.2 Dati di interesse

Inanzitutto è importante elencare quali sono i dati che effettivamente l'utente deve conoscere e quali deve poter modificare:

- Immagine acustica dell'ambiente circostante
  - Il pilota del veicolo deve poter visualizzare l'immagine sullo schermo di un computer;
- grafico in tempo reale dell'eco ricevuto
  - Il pilota del veicolo deve poter visualizzare in tempo reale l'eco ricevuto dal sensore sonar;
- valori numerici di interesse
  - Il pilota deve aver modo di visualizzare sullo schermo tutti i parametri di misura e, nel caso di uno UAV, tutte le informazioni relative allo stato del sistema di localizzazione.

Inoltre l'utente deve avere modo di variare con facilità i seguenti dati:

- parametri del sistema di localizzazione (caso AUV)
  - Durante la fase di testing del sistema di localizzazione, l'utente che effettua il tuning del sistema deve poter variare con facilità i parametri dell'algoritmo (vedi capitolo 4).
- parametri di misura (caso ROV)
  - Durante la navigazione deve essere possibile variare i parametri di misura del sonar (vedi paragrafo 1.1)

5.4.3 Descrizione della GUI

La figura 71 mostra uno screenshot relativo ad un'acquisizione effettuata nella piscina del Laboratorio di Modellazione Dinamica e Meccatronica di Pistoia. Buona parte dello schermo è dedicata alla visualizzazione dell'immagine acustica, l'utente può aumentarne lo zoom, spostarla e adattarla allo schermo tramite le più diffuse modalità di interazione con le immagini.

Dato che l'echosounder genera circa 10 misure al secondo l'immagine viene aggiornata sullo schermo con una frequenza di 10 Hz, ma nel caso in cui l'utente effettui alcune operazioni sull'immagine come spostamenti o zoom la frequenza aumenta fino a 25 Hz in modo da garantire fluidità.
Accanto all'immagine c'è uno svariato numero di slider che permettono di modificare i vari parametri in modo semplice, come se si trattasse di una console di comando. Nel riquadro nero a sinistra viene visualizzata l'eco di ritorno in tempo reale (linea verde) edi il segnale crosscorrelato (linea rossa).

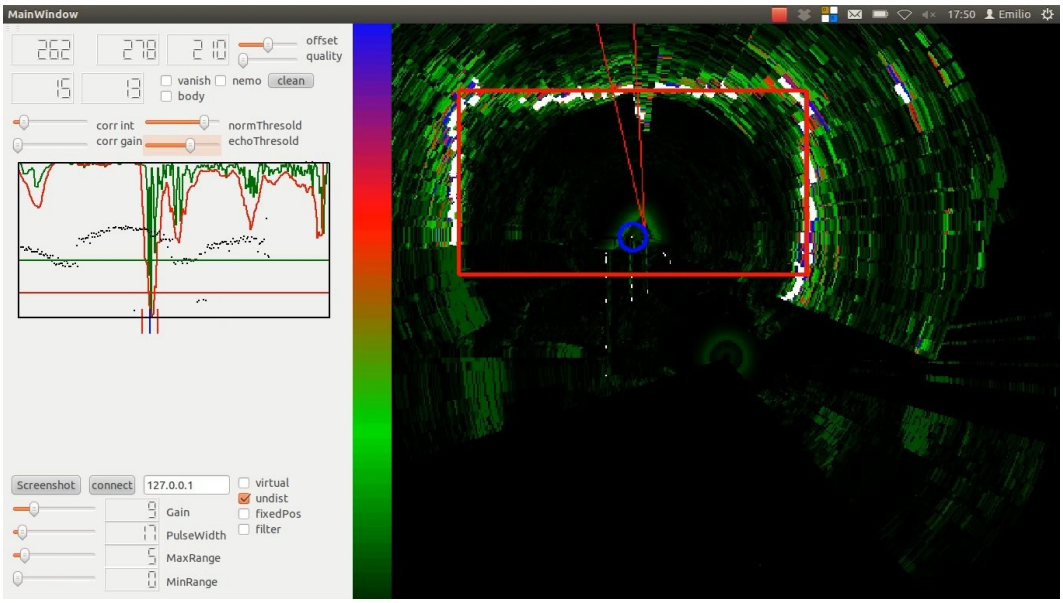

Fig 71: ScreenShot della GUI di Nemo ROV

Dal punto di vista computazionale il refresh dell'immagine in real time è un operazione molto onerosa tuttavia è indispensabile durante la fase di testing poichè migliora sensibilmente la fruibilità dei dati.

5.4.4 Echosounder virtuale

Immergere in acqua il veicolo richiede buona parte dei componenti del team e la messa in funzione della massima parte dei dispositivi; di conseguenza ogni test richiede molto tempo. Risulta chiaro come sia importante risolvere tutti i problemi tecnici prima di effettuare le prove; in questo caso si è scelto

di implementare un sensore virtuale.

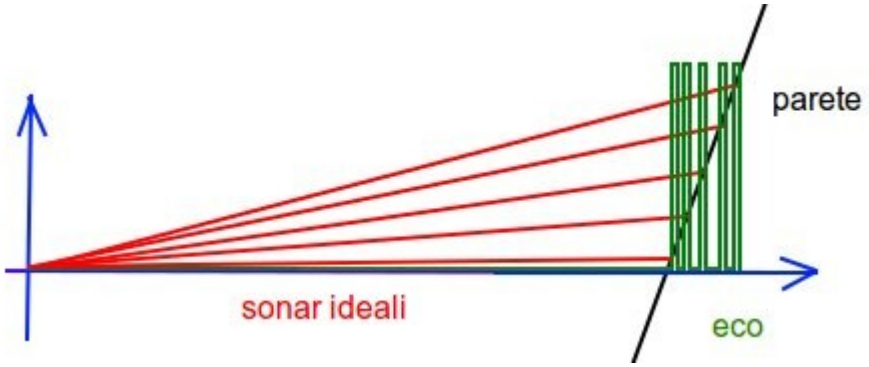

Fig 72: L'immagine riporta il meccanismo di generazione dell'eco virtuale: in rosso sono riportati i beam ideali, in nero la parete virtuale ed infine le linee verdi rappresentano l'eco generato

Come si vede nella figura 72 l'eco calcolato è la somma degli echi di k beam ideali, come dimostrano le prove sperimentali il sistema risulta un buon modello, in particolare risultano ben modellati:

1. l'indebolimento dell'eco quando il beam ha un angolo di incidenza molto stretto e viceversa eco molto forte quando il beam è ortogonale alla parete.
2. spigoli lontani "stondati": feonomeno dovuto all'ampiezza angolare non nulla del beam.

Tuttavia il sensore virtuale trascura i seguenti fenomeni:

- Presenza di oggetti all'interno del campo gara come boe o cavi.

Infine l'immagine successiva riporta un paragone tra una prova in piscina ed il suo analogo simulato.

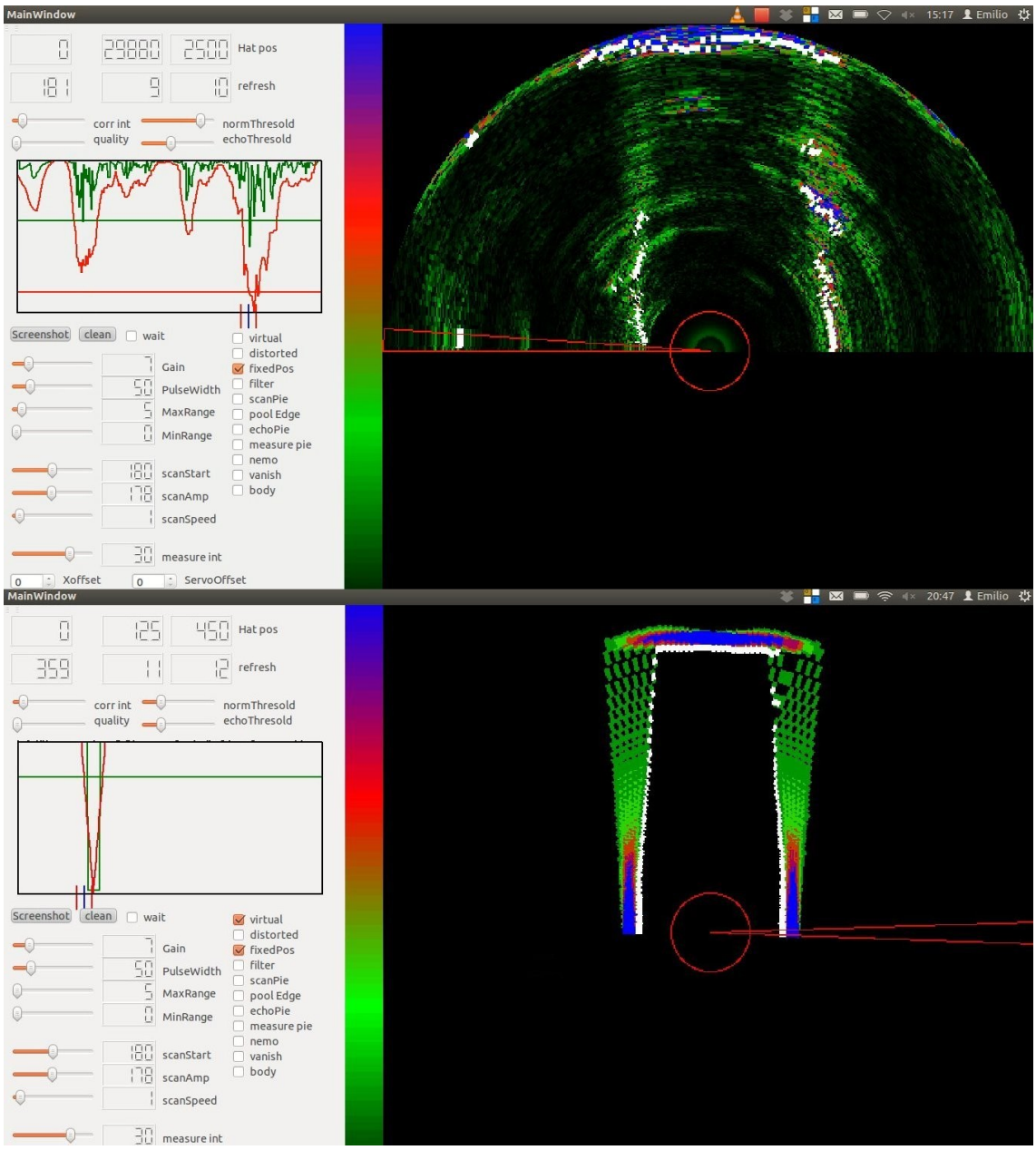

Fig 73: Confronto tra un'acquisizione simulata delle pareti della piscina (sotto) ed un'acquisizione reale (sopra). La piscina è rettangolare ed ha una lunghezza di 5 m ed una larghezza di 3m.

6 Risultati sperimentali

Nel seguente paragrafo si riportano i risultati ottenuti con il software descritto nei capitoli precedenti. Le prove sono state svolte nella piscina dell'MDM Lab di PIstoia. Si tratta di un ambiente di lavoro difficile per il sensore poiché, a causa delle ridotte dimensioni della piscina, il segnale viene riflesso varie volte prima di disperdersi completamente. Tutte le riflessioni successive alla prima sono da considerarsi come una componente rumorosa addizionale che nel campo gara non dovrebbero manifestarsi grazie alle dimensioni maggiori.

Fig 74: Un'immagine della piscina dell'MDM Lab

6.1 Generazione di immagini acustiche (ROV)

In questo paragrafo viene valutato il software impiegato per la generazione di immagini acustiche da impiegare per pilotaggio di veicoli pilotati in remoto (ROV). In questo caso non è necessaria l'estrazione di parametri dall'immagine; è invece necessario generare un'immagine sufficientemente nitida da permettere al pilota del veicolo di riconoscere la conformazione dell'ambiente circostante.

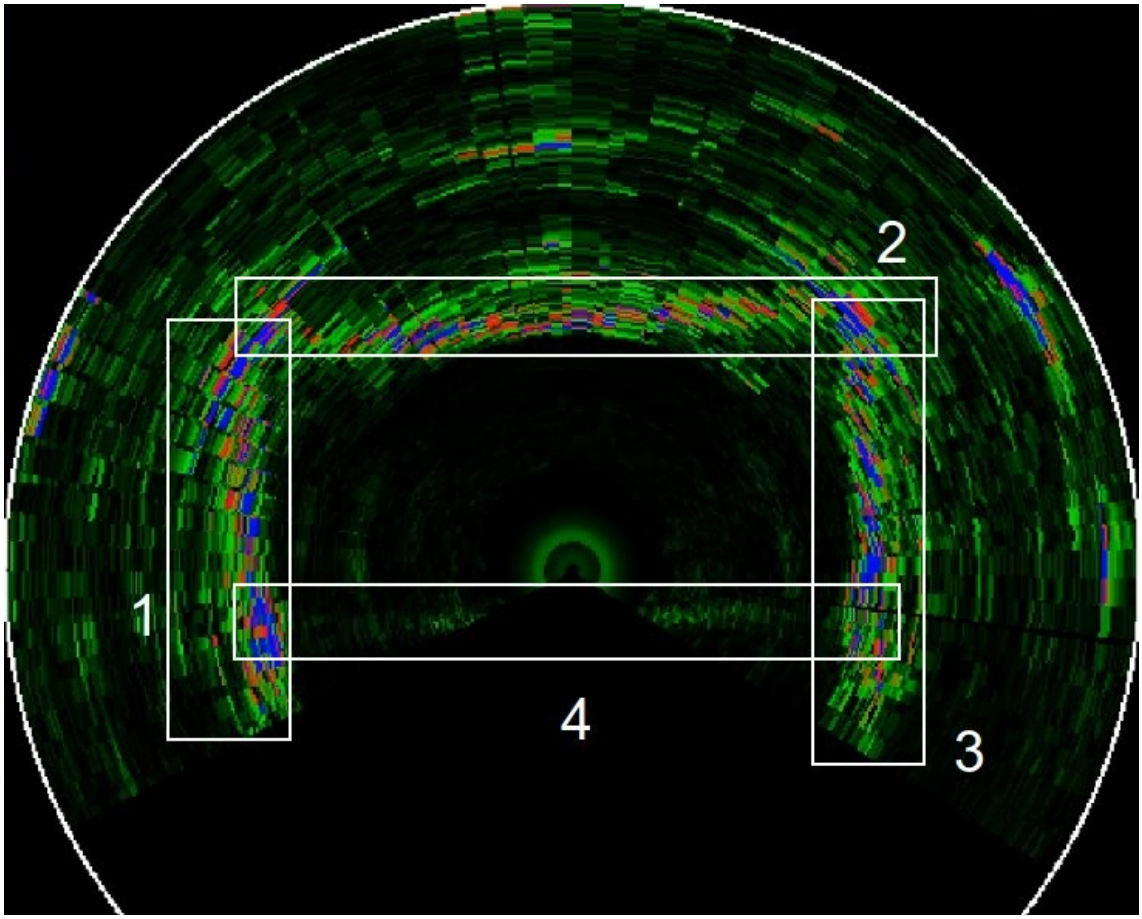

Fig 75: Immagine acustica della Piscina dell'MDM Lab di Pistoia

L'immagine mostra una scansione della piscina di Pistoia che ha una forma rettangolare con lati di dimensioni rispettivamente 6 e 2,5 m. Nell'immagine sono chiaramente riconoscibili 3 delle 4 pareti della piscina evidenziate dai 4 rettangoli bianchi; è presumibile che anche in spazi più ampi e complessi le immagini generate continuino ad essere nitide e possano realmente essere d'aiuto al pilota di un ROV per osservare l'ambiente in cui il veicolo deve muoversi.

6.2 Stima della posizione del veicolo

I test in piscina hanno dimostrato che l'algoritmo di localizzazione descritto nel capitolo 4 è in grado di localizzare il veicolo in modo sufficientemente robusto. Le immagini 76, 77 e 78 mostrano 3 delle fasi di elaborazione dell'algoritmo:

- Fase 1:
  - In questa fase viene realizzata l'immagine acustica dell'ambiente circostante.
- Fase 2:
  - Secondo le modalità indicate nel paragrafo 1.2 è stata generata l'immagine corrispondente al buffer dati dell'algoritmo.
- Fase 3:
  - L'immagine 78 riporta la trasformata di Hough applicata al buffer dell'algoritmo. Sono chiaramente presenti i massimi globali relativi alle pareti 1 e 2.

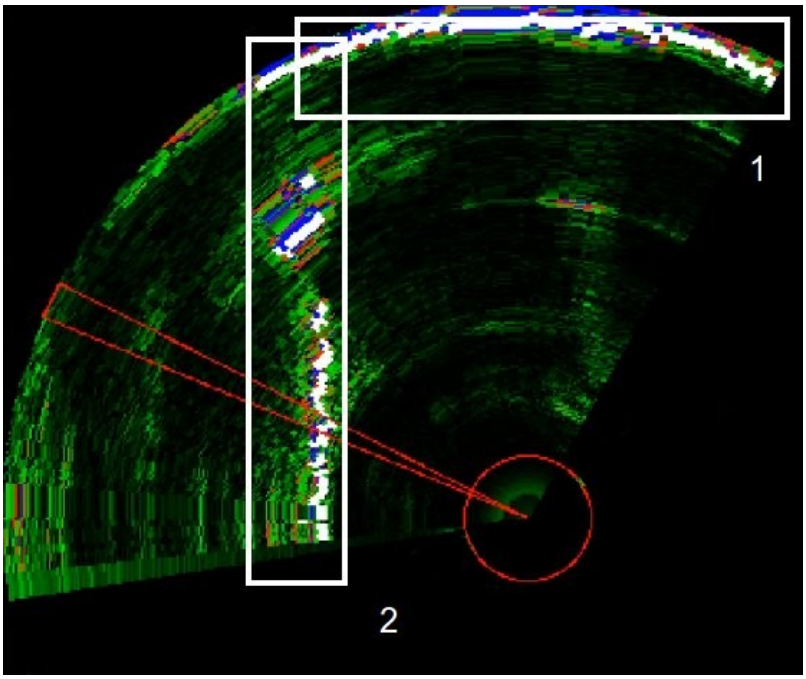

Fig 76: Immagine acustica della piscina ottenuta mediante la rotazione dell'echosounder

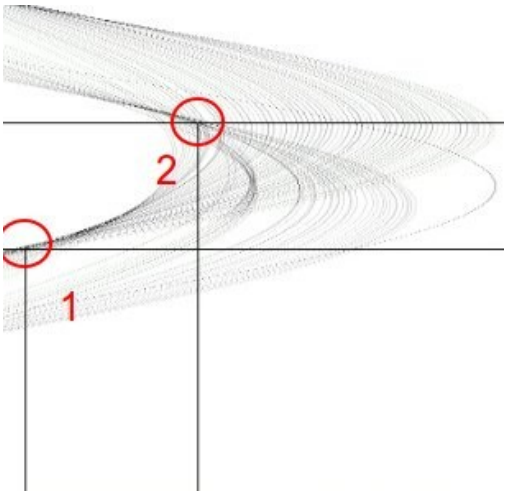

Fig 78: Trasformata di Hough applicata ai dati contenuti nel buffer; nei cerchi rossi sono eveidenziati i massimi relativi alla pareti 1 e 2.

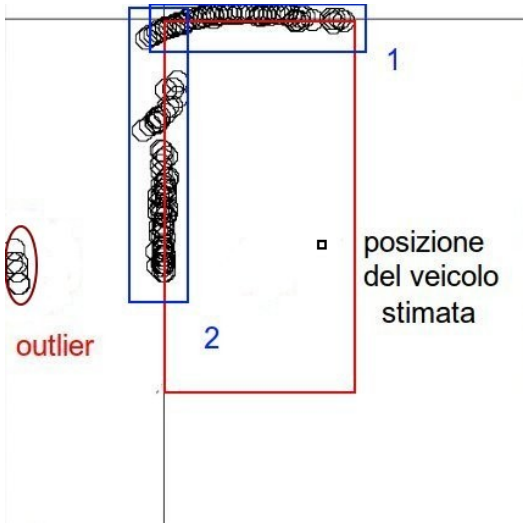

Fig 77: Immagine rappresentativa del buffer dell'algoritmo.

Conclusioni

Questo lavoro di tesi si è collocato nell'ambito della competizione SAUC-e a cui ho partecipato nell'edizione 2012, ed a cui parteciperò nel 2013. Lo scopo di questo lavoro è lo sviluppo di un sistema di localizzazione basato sull'impiego del sensore Imagenex 852 ultra miniature echosounder, un sensore SONAR unidirezionale. Il sistema di localizzazione realizzato in questo lavoro sarà impiegato nel veicolo FeelHippo sviluppato dal Laboratorio di Modellazione Dinamica e Meccatronica del Dipartimento di Ingegneria Industriale dell'Università degli Studi di Firenze per la competizione SAUC-e 2013.

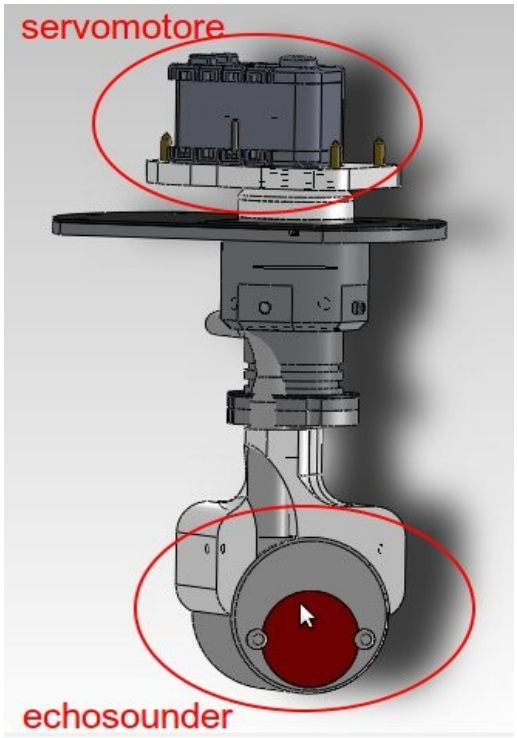

Fig 79: Disegno cad dell'assieme echosounder-azionamento

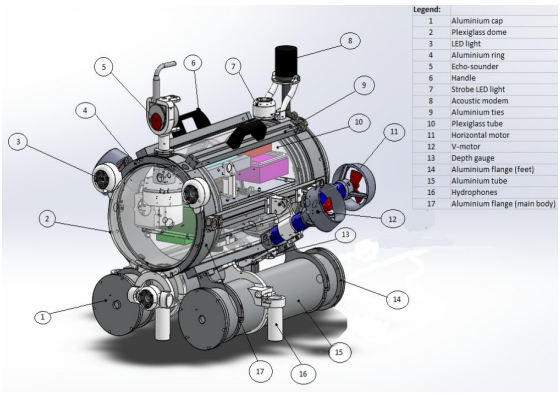

Fig 80: Disegno CAD del veicolo FeelHippo

Per superare le limitazioni dovute alla unidirezionalità del sensore echosounder è stato sviluppato un sistema meccanico dotato di un servomotore che permette al sensore di ruotare rispetto al proprio asse verticale. Di fatto l'aggiunta di questo grado di libertà ha trasformato l'echosounder da sensore monodirezionale ad un "mechanically scanned imaging sonar", ovvero un sensore in grado di realizzare immagini acustiche dell'ambiente circostante.

Fig 81: Foto della piscina dell'MDM Lab di Pistoia

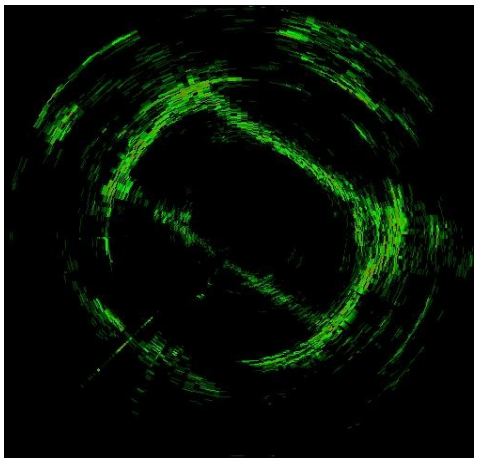

Fig 82: Immagine acustica della piscina dell'MDM Lab

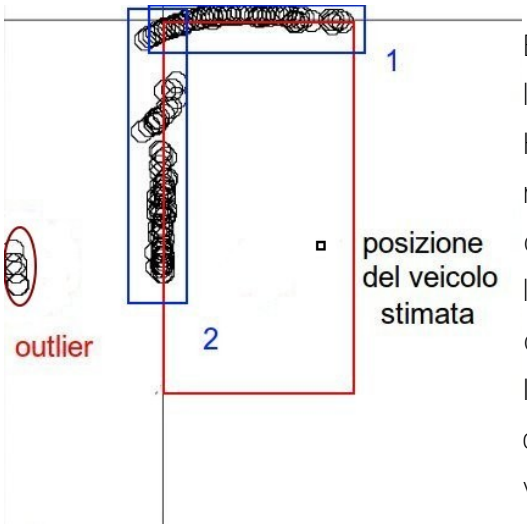

Fig 83: Immagine rappresentativa del buffer dell'algoritmo.

È stato realizzato un algoritmo di localizzazione basato sulla trasformata di Hough e sul filtro di Kalman che ha mostrato un buon comportamento durante i test e che sarà in grado di localizzare il veicolo rispetto alle pareti dellcampo gara del SAUC-e.

Infine è stato realizzato il software, conforme alle direttive del team, che verrà sfruttato dal processo di controllo del veicolo FeelHippo durante la competizione SAUC-e 2013.

La generazione di immagini sonar è stata inoltre impiegata sul veicolo Nemo ROV; è stata realizzata una GUI (Graphical User Interface) che permette al pilota di visualizzare l'ambiente circostante e quindi di evitare gli ostacoli che potrebbero danneggiare il veicolo .

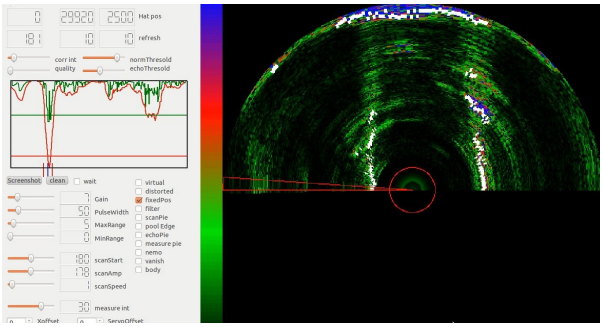

Fig 84: Screenshot della GUI realizzata per il veicolo Nemo ROV

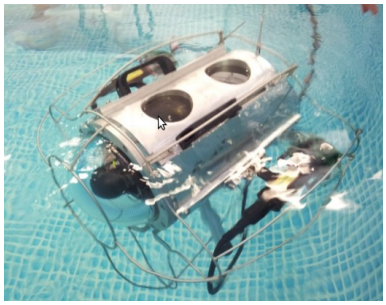

Fig 85: Foto del veicolo Nemo ROV

Sviluppi Futuri

Il sistema di localizzazione sviluppato in questo lavoro di tesi si fonda sulla conoscenza a priori del campo di gara che, grazie alla sua semplice topologia rettangolare, permette di localizzare il veicolo tramite l'identificazione di due sole pareti del bacino. Per sviluppare un sistema che sia efficiente anche al di fuori del campo gara del SAUC-e è necessario sviluppare un algoritmo basato su tecniche SLAM[8] (Simultaneous Localization And Mapping) che permetta al veicolo di localizzarsi ed in contemporanea identificare l'ambiente circostante.

Potrebbe essere interessante anche l'aggiunta di un ulteriore grado di lbertà al sensore che, in tal caso, permetterebbe di realizzare immagini 3D.

Il filtraggio ricorsivo inoltre, che in questo lavoro di tesi è stato sviluppato basandosi su un modello cinematico del veicolo, potrebbe essere migliorato con l'impiego di un modello più raffinato che tenga conto delle spinte effettuate dai propulsori e degli attriti dovuti alla viscosità dell'acqua.

Dal punto di vista software infine, la visualizzazione delle immagini potrebbe essere migliorata tramite l'impiego delle librerie grafiche OpenGL che permetterebbero maggiore libertà nella visualizzazione e manipolazione delle immagini.

Descrizione dei dispositivi impiegati

Imagenex 852 ultra miniature echosounder

Imagenex è un azienda canadese che opera nel settore dei sensori SONAR, il catalogo dell'azienda spazia dalle apparecchiature impiegate per la pesca ai kit per veicoli sottomarini autonomi comprendenti eco scandagli e SONAR multi beam.

Fig 86: Imagenex 852 ultra miniature echosounder

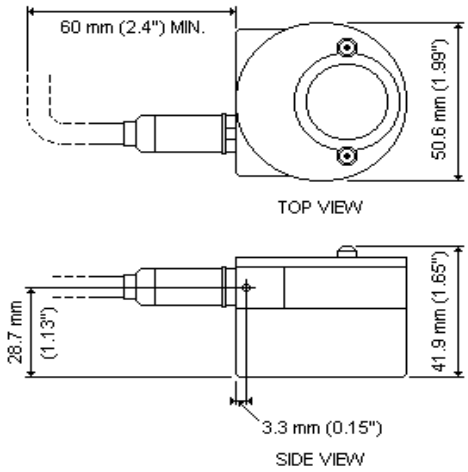

Fig 87: Dimensioni dell'Imagenex 852

L' Imagenex 852 ultra miniature echosounder è un sensore a basso costo diffuso principalmente tra i pescatori che lo usano per misurare la differenza dal fondo e rilevare la presenza di pesci; si tratta di uno dei più piccoli sensori SONAR in commercio comparabile solo con il Micron DST SONAR che però presenta un costo maggiore. Le prestazioni posso essere riassunte nella seguente tabella:

Frequenza dell'impulso	675 kHz
Trasduttore	Conico

Ampiezza forma d'onda	10°
Risoluzione massima	20 mm
Lunghezza zona d'ombra	150 mm
Profondità massima	300 m
Tipo di interfaccia	RS-232
Tensione di alimentazione	22-30 V DC < 1.5 W
Peso in aria	185 g (compreso cavo)
Distanza massima misurabile	50 m

Vantaggi:

- Basso costo
- Piccolo e leggero
- Protocollo di comunicazione semplice

Svantaggi:

- trasduttore conico (impossibile vedere i dettagli)
- Range limitato

Raspberry Pi

Raspberry PI è un PC di dimensioni poco superiori a quelle di una carta di credito costituito da una singola scheda; è stato sviluppato in Inghilterra dalla Raspberry Pi foundation con l'intento di promuovere l'insegnamento delle basi dell'informatica nelle scuole.
Il raspberry PI contiene un chipset Broadcom BCM2835 comprendente un processore ARM1176JZF-S da 700 MHz che può essere overcloccato fino a circa 1.0 Ghz tramite tool disponibili nella distribuzione dedicata Raspbian.
La GPU VideoCore 4 è sufficientemente potente da eseguire video in hd ed i 512 MByte di ram sono sufficienti per molte distribuzioni linux come Raspbian che è la distribuzione dedicata basata su linux Debian, infine la scheda di rete permette il controllo remoto del sistema. Il costo limitato (appena 26 €) ne ha determinato una rapida diffusione che garantisce un buon livello di debug e un'ampia letteratura in rete.

Fig 88: Il Raspberry Pi

Vantaggi:

- Basso costo;
- Ampia letteratura disponibile in rete;
- hardware potente per applicazioni embedded o controllo remoto;
- Disponibile una distribuzione ad-hoc di linux che comprende tutto il necessario;

Svantaggi:

- Scomodo da usare con tastiera e mouse;
- Hardware insufficiente per operazioni computazionalmente onerose come compilazione programmi, compressione/decompressione file.

Pololu micro Maestro 6

Pololu è un'azienda di Las Vegas fondata da alcuni studenti del MIT nel 2002; opera nel settore della robotica low-cost e produce dispositivi molto diffusi nell'ambito dell'hobbistica. Il Pololu micro Maestro 6 è uno dei prodotti più noti, permette di controllare con facilità fino a sei pin input-output che possono essere utilizzati in vari modi; solitamente vengono impiegati per il controllo di azionamenti di motori e servocomandi.

Il cuore della scheda è il PIC contenente un interfaccia usb-seriale e le 6 porte di I/O, il dispositivo non contiene un convertitore DC-DC e quindi è piuttosto sensibile alle variazioni della tensione di alimentazione tuttavia essendo alimentabile tramite USB questo limite non costituisce un problema.

Fig 89: Pololu Maestro 6

Vantaggi:

- Costo ridotto: 21 €;
- elevata semplicità di implementazione;
- 6 canali di I/O in uscita;
- L'elevata diffusione assicura la presenza di tutorial e esempi su internet.

Svantaggi

- Porta USB fragile (si è dissaldata poche ore prima della qualificazione nel SAUC E 2012);
- output ridotto al minimo: mancanza di feedback anche a livello software.

Servocomando HS-785HB

Hitec opera principalmente nel campo del modellismo, in particolare in quello dei servocomandi a basso costo: si tratta di dispositivi all-in-one composti da:

- Un azionamento analogico;

- un motore in corrente continua;
- un riduttore con rapporto di riduzione elevato (circa 60/1);
- un encoder solidale con il giunto di uscita (costituito da un trimmer).

Il dispositivo impiega il chip IC1 SN28654 che è un azionamento pwm a basso costo che riceve in ingresso una tensione proporzionale alla resistenza del trimmer. La quasi totalità dei servocomandi ha una porta a 3 pin:

- Alimentazione;
- massa ;
- segnale di controllo.

Fig 90: gearbox del servomotore

Il segnale di controllo è un'onda quadra a 50 Hz con un duty cycle che varia da 1 ms a 2 ms che corrispondono a -720° e 720° di angolo in uscita; la tensione di alimentazione è 6-9 V. Nel complesso si tratta di un dispositvo semplice e robusto; l'elevata coppia dovrebbe permettere al servo di ruotare con velocità e precisione sufficienti.

Max operating speed	60°/s a 4.8 V
Output Torque	1.078 Nm
Weight	0.11 Kg
Size	59 – 29 – 50 mm

Vantaggi:

- elevata semplicità di implementazione;
- costo ridotto;
- coppia elevata (circa 1 Nm);
- elevata inerzia del motore a causa dell'elevato rapporto di riduzione.

Svantaggi:

- Non si ha nessun feedback relativo alla posizione del braccio;
- la precisione è piuttosto bassa (circa 2-3°).

Fig 91: Il servomotore Hitech HS-785HB

Xsens Mti

Xsens è un azienda svedese nata nel 2000 specializzata nella cattura inerziale dei movimenti del corpo umano, la Mti è una piattaforma inerziale composta da una IMU (inertial measurement unit) e da una CPU che stima in tempo reale lo stato cinematico del veicolo, in particolare calcola in tempo reale l'orientazione del sensore nello spazio in terna NED (Noth East Down) o NWU (North West Up); permette inoltre l'accesso alle misure dei sensori inerziali (accelerometro triassiale, magnetometro triassiale, giroscopio triassiale) e quindi l'implementazione di algoritmi di stima diversi da quello proprietario. Si tratta di una delle piattaforme inerziali più valide a livello commerciale, ed

è stata impiegata da molti team partecipanti al SAUC-e.

Fig 92: Una foto dell'XSensMTi

Vantaggi

- Alto livello di precisione
- Alto grado di immunità ai disturbi elettromagnetici
- Alta frequenza di campionamento (fino a 120 misure al secondo)

Svantaggi

- Costo elevato

Commell SBC mainboard

Commell è un'azienda di Taiwan che opera principalmente nel settore dei pc industriali, tra i leader nella produzione di schede madri in formato mini ITX. La commel SBC mainboard è una scheda madre ad alte prestazioni compatibile con i processori di ultima generazione e corrisponde a tutti gli effetti al più potente dei personal computer desktop ma con ingombri ridotti. Il convertitore DC-DC in ingresso garantisce un ampio intervallo di tensioni di alimentazione:da 9 a 24 Volt. Già impiegata nel corso della competizione SAUC-e 2012 si è dimostrato essere un pc potente e affidabile.

Vantaggi:

- Elevata potenza di calcolo;
- dimesioni ridotte;
- ampia disponibilità di porte.

Svantaggi:

- Cosumo elevato (circa 20 w);
- costo elevato.

Fig 93: Il pc industriale Commel SBC

Robotis dynamixel ax-18

Robotis è un'azienda giapponese leader nella produzione di servocomandi digitali per il modellismo e per la ricerca; si tratta di un dispositivo dotato di un'elettronica digitale ed un interfaccia seriale che permette di conoscere l'angolo misurato dall'encoder del dispositivo.
Si tratta di un dispositivo del tutto analogo al hitech HS-785HB tuttavia presenta prestazioni superiori sotto diversi aspetti: in particolareal contrario del servo Robotis il servo Hitec non restituisce in uscita la misura dell'encoder. I test effettuati in laboratorio inoltre, hanno dimostrato che l'errore in inseguimento del servo Robotis è molto minore.

Peso	55.9g
Dimensioni	32 x 50.1 x 40 mm
Rapporto di riduzione	1:254 (con ingranaggi in plastica speciale)
Tensione operativa	9 - 12 Volts (raccomandato 11.1 V)
Corrente di Stand-By	50mA
Coppia massima	18Kgf*cm @ 12 V / 2.2A

Massima velocità a vuoto	97RPM @ 12V
Interfaccia di comunicazione	TTL Half Duplex
Sensore di posizione	Potenziometro
Risoluzione	300° (1024) o rotazione continua
Motore	Coreless

Fig 94: Il servo motore Robotis Dynamixel ax-18

Vantaggi:

- Dimensioni ridotte;
- Coppia elevata;
- Disponibilità di librerie proprietarie per la realizzazione del software .

Svantaggi:

- Richiede un convertitore seriale-usb proprietario

Bibliografia

[1] T. Fossen, “Guidance and Control of Ocean Vehicles”, 1st ed., John Wiley and Sons, Chichester, UK, 1994.

[2] MOOS-IvP Software, http://moos-ivp.org .

[4] Qt Project, http://qt-project.org .

[5] Qwt Library, http://qwt.sourceforge.net .

[6] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Robotica , "Modellistica, pianificazione e controllo”, 3rd ed., McGraw-Hill, Milano, 2008.

[7] Dariush Forouher, Jan Hartmann, Marek Litza, Erik Maehle , “Sonar-based FastSLAM in an Underwater Environment Using Walls as Features ”, 15th International Conference on Advanced Robotics , 2011.

[8] David Ribas, Pere Ridao Jos´ Neira, Juan D. Tard´ s , “SLAM using an Imaging Sonar for Partially Structured Underwater Environments ”, International Conference on Intelligent Robots and Systems , 2006.

[9] Wajahat Kazmi, Pere Ridao, David Ribas and Emili Hern´ ndez , “Dam Wall Detection and Tracking Using a Mechanically Scanned Imaging Sonar”, IEEE International Conference on Robotics and Automation , 2009 ,

Le " iniziano qui

UNIFICA NOTAZIONE Iniziale nome. Cognome

UNIFICA NOTAZIONE Iniziale nome. Cognome

UNIFICA NOTAZIONE Iniziale nome. Cognome