ISSN/2499-5347

GIACOMO CUPERTINO

Dipartimento Tecnologie Energetiche
Smart Energy
Intelligenza Distribuita e Robotica per l'Ambiente e la Persona
Centro Ricerche Casaccia

VENUS

Veicolo per Navigazione Subacquea e Sorveglianza
Programmazione di Base – Prove Preliminari

RT/2020/11/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

GIACOMO CUPERTINO

Dipartimento Tecnologie Energetiche
Smart Energy
Intelligenza Distribuita e Robotica per l'Ambiente e la Persona
Centro Ricerche Casaccia

VENUS

Veicolo per Navigazione Subacquea e Sorveglianza
Programmazione di Base – Prove Preliminari

RT/2020/11/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

Ringraziamenti

A tutti coloro che hanno supportato e consentito la conduzione
delle prove di VENUS al lago di Bracciano.
Ai colleghi del Laboratorio di Robotica che hanno prestato a vario titolo
il loro prezioso tempo, utile al conseguimento di questi primi risultati.

VENUS

Veicolo per Navigazione Subacquea e Sorveglianza
Programmazione di Base – Prove Preliminari

Giacomo Cupertino

Riassunto

VENUS è un prototipo di robot autonomo interamente sviluppato nei Laboratori ENEA – DTE-SEN-IDRA, per missioni di esplorazione / monitoraggio / sorveglianza subacquea. Gli aspetti trattati riguardano sia l'architettura hardware che i programmi base associati, relativi alla gestione di tutti i sottosistemi ed i dispositivi del veicolo. Viene descritta la rete interna LAN insieme alle interconnessioni logiche e fisiche tra i vari sistemi di calcolo e tra i dispositivi. Sono infine illustrate le prove preliminari. I test sono stati condotti nella piscina del Laboratorio e poi al lago di Bracciano, in acque poco profonde.

Parole chiave: robotica subacquea, prototipo sperimentale, applicazioni robotiche, interfaccia uomo-macchina, programmazione orientata agli oggetti.

Abstract

VENUS is a prototype of an autonomous underwater robot fully developed at ENEA – DTE-SEN-IDRA Labs, to carry out search / monitoring / surveillance tasks. The aspects here detailed are those both about the hardware architecture and the associate basics programs relative to the management of all the subsystems and devices of the vehicle. The internal LAN is described together with the logical and physical interconnections among all the diverse computing systems and devices. Preliminary tests are finally illustrated. The tests were performed in the Robotics Labs pool and then at Bracciano lake, in open shallow-water.

Keywords: *underwater robotics, experimental prototype, robotics applications, human-machine interface, object oriented programming.*

INDICE

1. Introduzione

Il VENUS (VEicolo per la Navigazione sUbacquea e la Sorveglianza) è un prototipo di robot sottomarino, la cui struttura (nel seguito anche *vessel* o veicolo) è principalmente costituita da lega di alluminio.

La principale linea ispiratrice da cui deriva questo progetto trae spunto da una serie di considerazioni sulle potenzialità applicative in ambito di robotica subacquea che vengono di seguito riportate:

- l'ostilità all'uomo dell'ambiente di lavoro sottomarino a profondità che partono da qualche decina di metri;
- un contesto applicativo nell'ambiente sommerso sempre più orientato all'*IoUT* (*Internet of Underwater Things* [**1**]);
- una sempre crescente attenzione del settore alla sensorialità diffusa *UMSN* (*Underwater Mobile Sensor Network* – Reti di sensori mobili sottomarini [**2**]).

Quanto detto fa comprendere come l'esistenza e la semplice produzione in piccola serie di un veicolo a basso costo originariamente studiato per il monitoraggio e la sorveglianza delle cosiddette *shallow water* (acque poco profonde), possa rispondere ad una esigenza sempre più diffusa.

Appare evidente il vantaggio di disporre di un veicolo autonomo (*Autonomous Underwater Vehicle* o AUV) operante contemporaneamente ad altri, per determinati tipi di missione, come ad esempio l'esplorazione di vaste aree di fondale. Una stretta interazione tra veicoli con idoneo sistema di comunicazione ne aumenterebbe l'efficienza e la riuscita della missione in tutte quelle operazioni nelle quali il coordinamento ed il conseguente scambio dati è essenziale.

Il VENUS, da un punto di vista metodologico e strettamente tecnico, può essere visto come un processo automatico che necessita di controlli. Tra questi, quello digitale [**3**] è un tipo di controllo in retroazione nel quale è presente almeno un calcolatore che effettua una elaborazione a tempo discreto della legge di controllo: l'algoritmo a cui fa riferimento elabora il segnale proveniente dai trasduttori e produce uno o più comandi sugli attuatori.

Per poter funzionare, ogni dispositivo di bordo richiede un controllo locale, perciò un controllore digitale, posto in sua prossimità, assolve a questo scopo attraverso un codice opportuno. Ad esempio, la bussola necessita di uno specifico *driver* per poter leggere la direzione; il propulsore, di un software che consenta di regolare la velocità di rotazione dell'elica; il trasduttore di pressione (profondimetro), di opportuno script che sia in grado di convertire, in maniera affidabile, il dato grezzo in dato di profondità e così via.

A monitorare i dati e ad attuare il controllo complessivo del sistema, sovrintendono specifiche unità di elaborazione, descritte in seguito, attraverso il software di controllo preposto a gestire tutti i diversi dispositivi presenti a bordo.

L'obiettivo di questo lavoro è approfondire gli aspetti legati all'architettura elettronica e descrivere il software di base atto al suo governo. Questa serie di programmi ha costituto il punto di partenza per lo

sviluppo della fase successiva del lavoro che ha visto l'implementazione del Sistema di Controllo complessivo del robot trattato in [**5**].

Dai *test preliminari* condotti al lago di Bracciano, si traggono delle prime conclusioni sulle reali potenzialità di utilizzo del veicolo in questione, utilizzando l'interfaccia uomo-macchina implementata e qui descritta.

Si rinvia ad altri documenti per approfondimenti inerenti la descrizione relativa alla struttura dello scafo ed ai suoi sistemi di bordo [**4**] e l'implementazione del Sistema di Controllo [**5**] sul prototipo.

2. Dispositivi e loro interconnessioni

Al suo interno il VENUS è dotato di vari dispositivi elettrici, elettronici e meccanici (*cfr.* [**4**]), con le loro peculiari funzioni. Il traffico dati che intercorre tra loro viene condiviso con più controllori digitali, a loro volta interconnessi.

Ogni controllore digitale presente a bordo è un vero e proprio computer ed assolve determinate funzioni. Come accennato, questi computer condividono le informazioni attraverso una rete locale (**fig. 1.1**), ribattezzata *L*ocal *V*enus *N*etwork. Grazie a questa rete dati, il VENUS può colloquiare con un computer remoto per mezzo del quale l'operatore instaura il collegamento (mediante PC o laptop o tablet) tramite rete WiFi configurata in modalità *ad-hoc*.

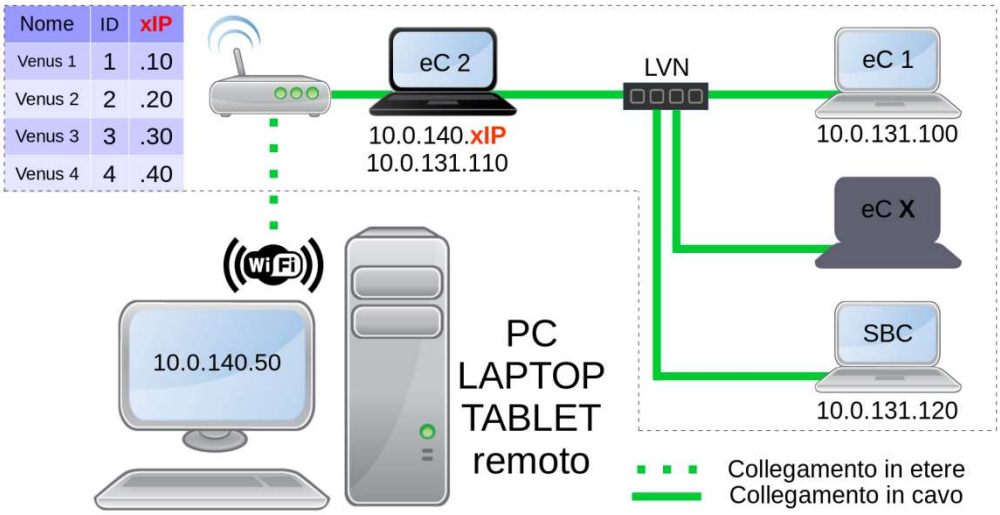


Figura 1.1 – Architettura della rete Venus locale (***L**ocal **V**enus **N**etwork*) comprendente anche un PC remoto (*Stazione di Controllo*)

All'interno del VENUS, come illustrato in **fig. 1.1**, vi sono due tipologie di computer:

- una viene definita 'a computer incorporato' (in sigla *eC – embedded Computer*), concepita appositamente per un utilizzo specializzato all'esecuzione di determinati compiti (*special purpose*) e viene normalmente impiegata per svolgere funzioni più specifiche e ricorsive;
- l'altra, che è un vero e proprio 'computer a singola scheda' (in sigla *SBC – Single Board Computer*), è concepito per impieghi più generici (*general purpose*) e quindi atto a garantire una maggiore

flessibilità di utilizzo, oltre che una maggiore potenza di calcolo rispetto alla prima tipologia, conservando comunque una notevole compattezza rispetto ad un PC tradizionale.

Nella **figura 1.2** viene illustrato lo schema di collegamento dei vari dispositivi di bordo ai computer.

Come mostrato nella **fig. 1.2**, in coerenza con quanto già illustrato in [**4**], questo schema sintetizza l'architettura dati che risulta composta da:

- cinque *moduli elettronici*;
- un sottosistema costituito da una scheda elettronica a bordo della quale vi è il *SERVO Controller* ;
- due dispositivi elettro-meccanici (motore e sonar);
- tre computer;
- un *hub-switch* che interconnette questi ultimi alla rete locale.

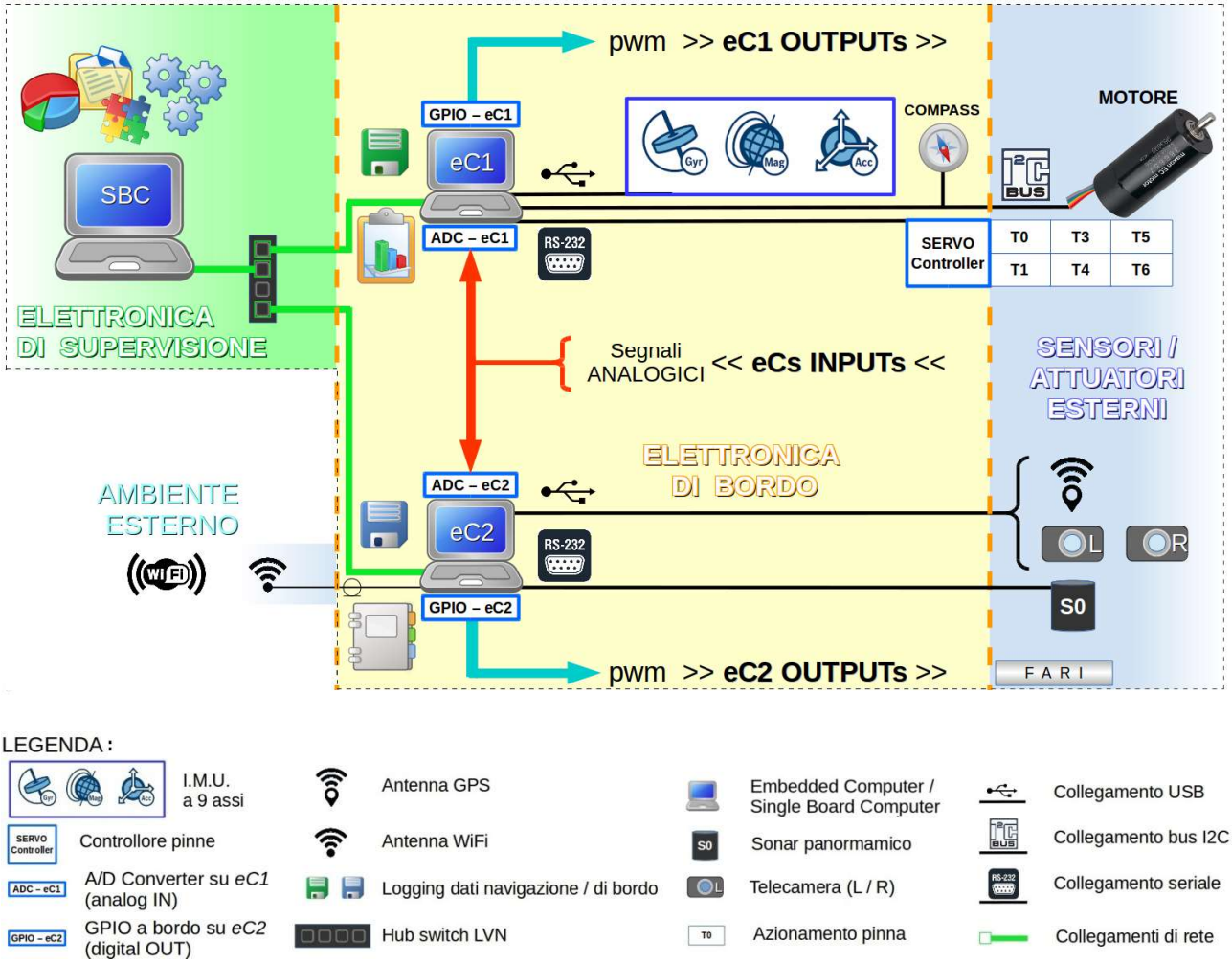


Figura 1.2 – Architettura dati e ripartizione delle risorse di calcolo all'interno del VENUS, suddivisa per sezioni

Ora si passano in rassegna i dispositivi e le loro funzionalità.

Per facilitare la lettura, si riporta il nome del dispositivo relativo all'identificativo in **fig. 1.2** tra parentesi quadre:

- *Bussola elettronica [COMPASS]*, dispositivo digitale a 12 bit che legge la direzione di marcia del

veicolo rispetto al Nord secondo un angolo compreso tra 0 e 359,9° e lo invia su bus seriale *I2C*;

- *Unità di misura inerziale [IMU]*, sensore inerziale a nove assi (costituito da giroscopi, accelerometri e magnetometri – tre per ogni asse cartesiano *[x, y, z]*) ad alta sensibilità, caratterizzato da una risoluzione dati a 16 bit e da un tempo minimo di campionamento pari a 4ms;
- *Modulo di rilevazione della posizione globale [GPS]*, che fornisce le coordinate di latitudine e longitudine al computer a cui è collegato;
- *Webcams [Left e Right]*, videocamere a regolazione automatica e ad alta risoluzione che consentono la cattura di singoli fotogrammi stereoscopici o registrazioni video in *full-HD*;
- *[SERVO controller],* modulo elettronico a bordo di *scheda madre* appositamente sviluppata (**fig. 1.2** – *cfr. [S-SENS1]* in [**4**]), per la conversione del dato numerico digitale proveniente dal computer in posizione angolare sull'azionamento di ciascuna pinna;
- *sonar panoramico [S0]*, dispositivo elettro-acustico che fornice, tramite connessione seriale, informazioni legate al profilo del fondale e altre informazioni utili ai fini di un eventuale *obstacle avoidance* al computer a cui è collegato;
- *propulsore [MOTORE]*, controllato per via digitale su bus *I2C* mediante scheda elettronica appositamente sviluppata ed interposta tra computer e motore (*cfr. [S-MOT]* in [**4**]), non riportata in **fig. 1.2** per non appesantire troppo il disegno. La scheda elettronica consente la gestione della potenza sviluppata dall'elica, secondo opportuni parametri (come il verso di rotazione, il *set-point* della velocità di rotazione, la frenatura del *rotore*) preimpostati dal computer a cui è interconnessa;
- *[eC1]*, computer *embedded* che si occupa prevalentemente di acquisire e processare i dati di navigazione e gestirne i relativi attuatori. Esso traccia anche tutti i dati di sua competenza su vari file di *log*;
- *[eC2]*, computer *embedded* che si occupa di acquisire e processare i dati provenienti dai servizi di bordo, quali quelli acustici (legati al funzionamento del sonar panoramico *[S0]*), quelli relativi alle coordinate geografiche (restituite dal modulo GPS) ed eventuali flussi video generati dalla/le telecamera/e. Questo computer gestisce anche i fari, la comunicazione WiFi con l'esterno ed, infine, registra ciò che avviene sul veicolo: dal livello di carica delle batterie allo stato generale del sistema, elaborando anche statistiche energetiche. L'*eC2* è anche sede del *master* per il *middleware ROS* [**5**] e traccia tutti i dati di sua competenza su vari file di *log*;
- *[SBC]*, allo stato attuale funge da computer per la visione artificiale, potendo in futuro accentrare a sé tutte quelle operazioni di calcolo complesso riconducibili ad una singola macchina che richiedono una maggiore velocità di esecuzione o di post-elaborazione. I risultati delle decisioni e/o dei calcoli possono essere reindirizzati agli *hosts* tramite la *LVN*. L'*SBC* può anche fungere da data-center generico e, ove richiesto dalla particolare applicazione, registrare video stereoscopici in alta definizione (*fullHD*) per conservarli in memorie dedicate per estrarne informazioni specifiche o al solo scopo di archivio.

Nella **fig. 1.2** vengono anche riportati dei generici flussi di dati rappresentati da due colorazioni: frecce rosse, che indicano gli ingressi analogici (*Analog/Digital Converter – ADCs*) provenienti dai sensori, e

frecce verdi, che indicano le sole uscite digitali (*General Purpose Input Output – GPIOs*) con protocolli diversi dallo standard (come invece sono l'USB, la seriale, ecc…).

Questi flussi viaggiano su piste che collegano tra loro le varie schede elettroniche presenti a bordo (*cfr.* [**4**]) e non vengono quindi raffigurate nella sezione centrale di **fig.1.2**.

Segue la **tab. 1.1** dove sono riportati anche i componenti non compresi o non esplicitati in **fig. 1.2**, ma che rappresentano comunque elementi fisici presenti sul veicolo (*cfr.* [**4**]), oggetto di controllo software.

NOME DEL COMPONENTE		FUNZIONE
SIGLA	DESCRIZIONE	
Ls	Led di Stato	Led RGB indicante lo STATO generale del VENUS (sistema in allarme, info da trasmettere, ecc...)
Lb	Led batterie	Led RGB di segnalazione stato di carica BATTERIE
SOG	Segnalazioni Oggetti Galleggianti	Gruppo di led bianchi che offrono all'esterno del veicolo una segnalazione luminosa di STATO
H0	Sonda di temperatura/umidità	Misurare temperatura ed umidità relativa interna
P0	Trasduttore di pressione	Fornire una misura della profondità di immersione
T1 e T2	Azionamenti pinne anteriori	Gestire le sei pinne di bordo, in base alle azione di controllo esercitate dal *SERVO Controller* relativo
Da T3 a T6	Azionamenti pinne posteriori	

Tabella. 1.1 – Riepilogo delle componenti meccaniche/elettroniche a bordo del VENUS, non comprese nelle sezioni ELETTRONICA DI BORDO e SENSORI/ATTUATORI ESTERNI di **fig. 1.2**

3. Controllori digitali di bordo e funzioni specifiche

Riguardo a questo argomento, si espongono le caratteristiche tecniche dei computer impiegati e le principali funzioni ad essi assegnati a bordo ai fini della programmazione di base.

Il computer riportato nella prima colonna di **tab. 2.1** *[eC]* è un tipo di computer a singola scheda, estensione del concetto di *SoC* (*System on Chip*), che si colloca a metà tra un computer completo ed un microcontrollore. Ricade nella categoria degli *eCs* in quanto il suo processore, basato su architettura *ARM*, è studiato per compiere operazioni specializzate e ripetitive. Questa macchina, a differenza di un computer classico (del tutto assimilabile ad un *SBC* – seconda colonna di **tab. 2.1**), presenta delle limitazioni sulla disponibilità di *driver* o di pacchetti specifici altrimenti diffusi e facilmente reperibili in rete per la maggior parte delle macchine che si usano al lavoro (*PC desktop* o *laptop*).

Questi sistemi elettronici specializzati, dalle compatte dimensioni e relativamente economici, possiedono tuttavia un sistema operativo vero e proprio residente sulla seconda delle due partizioni in cui è suddivisa la scheda *microSD*, unico supporto di *memoria di massa* disponibile. Senza questo supporto il sistema non potrebbe iniziare nemmeno la fase di avvio, in quanto esso non può prescindere dalla lettura dei primi *settori* dalla prima partizione, sulla quale risiede il *bootloader* (*U-boot*).

COMPONENTE INTERNO		TIPO DI COMPUTER	
		eC	*SBC*
	Processore	ARM Cortex-A8 @1GHz	INTEL Quad-core 64bit @2,56GHz
	Memoria RAM	256Mb RAM + 256Mb NAND	8Gb RAM DDR3L
	Memoria di massa	Slot per µSD card	eMMC 32Gb + slot per µSD card
	Audio	x2 mini-jack audio: out + mic	S/PDIF out
	Scheda grafica	(optional)	Intel HD Graphics 700MHz
	Uscite video	HDMI	HDMI + x2 miniDP++
	Networking	10/100 Base-T Ethernet	Gigabit Ethernet
	USB	x1 2.0 tipo A + x1 OTG tipo µA	x3 USB 3.0 tipo A
	Serial ports	x1 UART + Console su USB tipo µA	x1 UART
	Altre interfacce	x1 I2C, x1 SPI ports	x1 I2C, x1 SPI ports
		6 A/Ds input	6 A/Ds input
		8 GPIOs	12 GPIOs
	Multimedia	-	H264 HW Video encode/decode
	Sistema operativo	Linaro 12.04 (Linux per piattaforme *ARM*)	Linux Mint 19.2 *Tina ed. Cinammon* (Ubuntu 18.04 *Bionic*)
	Dimensioni	105mm x 40mm	120mm x 85mm

Tabella. 2.1 – Riepilogo delle principali caratteristiche tecniche dei computer (*embedded* ed *SBC*) a bordo del VENUS

Come descritto precedentemente, la funzione principale della coppia di *eCs* presenti a bordo è quella di gestire tutti i dispositivi periferici locali fisicamente collocati in loro prossimità per renderli disponibili al sistema esterno ai fini del controllo.

In **tab. 2.2** e **2.3** sono specificate le principali caratteristiche hardware associate rispettivamente all'*eC1* ed all'*eC2*, così come configurate a bordo.

In sintesi, la lettura delle tabelle seguenti indica che la *eC1* è deputata fondamentalmente alla navigazione del veicolo, mentre la *eC2* al controllo delle telecamere e, più in generale, ai servizi collocati in prua.

In coerenza con quanto riportato in **tab. 1.1** ed in **fig. 1.2** e per offrire una maggiore continuità di lettura rispetto ad altri documenti collegati al presente rapporto, nelle **tabelle 2.2** e **2.3** si indicano tra parentesi quadre tutti quei componenti fisici del veicolo come identificati in [**4**].

Sempre nelle **tabelle 2.2** e **2.3** si riporta tra parentesi graffe il nome dell'omonimo nodo *ROS* [**5**] o del *demone* che gestisce e monitora il processo associato al software di controllo del dispositivo corrispondente alla **fig. 1.2**.

TIPO INTERFACCIA		IDENTIFICATIVO COMPUTER: ***eC1***		
		Identificativo ingresso/uscita	*Tipo codifica/ BaudRate*	*Funzione controllata – {Nome nodo ROS [3] o demone}*
AI	***ADCs***	*2*	10 bit – 20Hz	Lettura analogica di pressione *[P0] – {PRESSOSTATO}*
AI		*7*	10 bit – 20Hz	Lettura binaria stato *[S-MOT] – {SAFEMON}*
DO	***GPIOs***	*pwm1 - 32kHz**	7 bit	Gestione ventola interna raffreddamento – *{FAN}*
DO		*pwm11 – 0Hz**	7 bit	Reset *[SERVO Controller] – {TIMONI}*
DO		*pwm8 – 0Hz**	7 bit	Azionamento FRENO motore *[S-MOT] – {MOTORE}*
DO		*pwm9 – 0Hz**	7 bit	Inversione di marcia F/R *[S-MOT] – {MOTORE}*
I/O	*I2C*	*SDA / SCL*	100 kHz_{CLK}*	Lettura dati a 12 bit dalla bussola – *{BUSSOLA}* Regolazione potenza a 8 bit su propulsore – *{MOTORE}*
I/O	*USB*	*USB Host*	≤12,5 kHz*	Gestione flusso dati a 16bit da IMU – *{IMU}*
DO	***UART 1/tx***	*txd3*	9,6 kbps	Invio dati verso *[SERVO Controller] – {TIMONI}*
DI	***UART 1/rx***	*rxd3*	9,6 kbps	Ricezione dati sonda temperatura *[H0] – {SAFEMON}*
I/O	*Console*	*USB Console*	115200 – 8n1	Gestione / configurazione computer tramite PC esterno
LEGENDA:		I: Input - O: Output - I/O: Digital Input/Output – A: Analog – D: Digital ADC: Analtog / Digital Converter - GPIO: General Purpose Input/Output 0Hz sta per un pwm degenerato in funzionamento on/off *: limiti legati all'hardware o per configurazioni dei *driver* corrispondenti		

Tabella 2.2 – Elenco delle configurazioni sulle interfacce utilizzate in coda al veicolo e gestite da *eC1* [**6**]

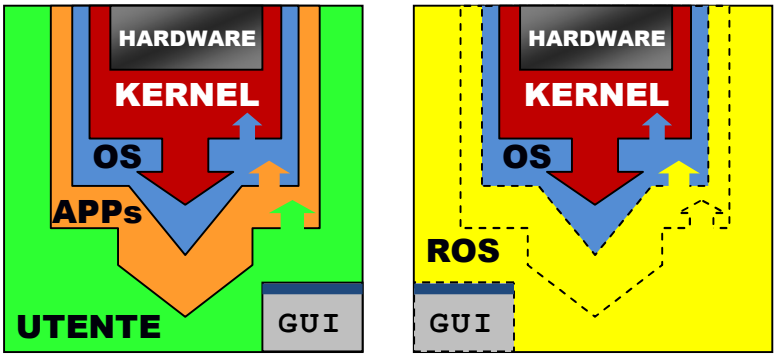
TIPO INTERFACCIA		IDENTIFICATIVO COMPUTER: ***eC2***		
		Identificativo ingresso/uscita	*Tipo codifica/ BaudRate*	*Funzione controllata – {Nome nodo ROS [3] o demone}*
AI	***ADCs***	*2*	10 bit – 20Hz	Lettura corrente assorbita dal veicolo – *{SOC}*
AI		*3*	10 bit – 20Hz	Lettura soglia di scarica batterie <20% – *{SOC}*
AI		*4*	10 bit – 20Hz	Lettura soglia di batterie cariche >90% – *{SOC}*
AI		*5*	10 bit – 20Hz	Lettura livelli di luminosità esterna – *{SOC}*
AI		*7*	10 bit – 20Hz	Lettura soglia di scarica batterie <8% – *{SOC}*
DO	***GPIOs***	*pwm0 - 0kHz**	7 bit	Predisposizione attivazione modem ottico – *{OTTICO}*
DO		*pwm9 – 0Hz**	7 bit	Accensione e spegnimento fari – *{VIDEO}*
DO		*pwm10 – 0Hz**	7 bit	Gestione *[SOG] – {PWM170_TEST}*
DO		*pwm170 – 0Hz**	7 bit	Gestione led di Stato veicolo *[Ls] – {PWM170_TEST}*
I/O	***USB***	*USB Host*	≤480 Mbps	Gestione flusso video da telecamera/e – *{VIDEO}*
				Gestione flusso dati NMEA da modulo GPS – *{GPS}*
I/O	***UART 1***	*rxd3/txd3*	115,2 kbps	Gestione flusso dati da e verso sonar *[S0] – {SONAR}*
I/O	***Console***	*USB Console*	115200 – 8n1	Gestione / configurazione computer tramite PC esterno
LEGENDA:		I: Input - O: Output - I/O: Digital Input/Output – A: Analog – D: Digital ADC: Analtog / Digital Converter - GPIO: General Purpose Input/Output 0Hz sta per un pwm degenerato in funzionamento on/off *: limiti legati all'hardware o per configurazioni dei *driver* corrispondenti		

Tabella 2.3 – Elenco delle configurazioni sulle interfacce utilizzate in testa al veicolo e gestite da *eC2* [**6**]

4. Struttura dati e loro flusso

Il software correlato all'hardware sin qui esposto rappresenta l'assieme delle istruzioni che possono essere eseguite da ciascun computer a bordo del veicolo, affinché, nel suo complesso, possa svolgere determinati compiti. Questi gruppi di istruzioni, genericamente chiamati programmi, possono essere scritti con un linguaggio macchina a basso livello o con un linguaggio di alto livello.

Immaginando un generico computer come un insieme di tante macchine elementari tutte tra loro interconnesse, è utile fare riferimento alla **fig. 3.1** che rappresenta una struttura architetturale del software a strati. Ogni linea nera continua indica il confine tra due strati attigui, mentre, se tratteggiata, indica una separazione meno rigida tra gli strati, tanto da 'confondersi' gli uni con gli altri (*astrazione dell'hardware*).



(a) *(b)*

Figura 3.1 – Raffigurazione intuitiva a strati della più diffusa rappresentazione di architettura software (ogni riquadro/colore rappresenta uno strato di software):
a) architettura diffusamente accettata ***b)*** architettura in presenza di *middleware ROS*

L'insieme dei programmi oggetto del presente lavoro, parte dai comandi e dai dispositivi messi a disposizione dal sistema operativo (in seguito identificato dalla sigla *OS – Operating System*), basato sul *kernel* nei casi di *OS* Linux, per estendersi in maniera semplice ed intuitiva all'utente finale (**fig. 3.1**), attraverso interfaccia grafica (*GUI – Graphical User Interface*).

A partire dalle due illustrazioni di **fig. 3.1**, si esplicitano brevemente entrambi i metodi usati per far svolgere al robot determinati compiti. Si richiama il nome del corrispondente strato software di **fig. 3.1** tra parentesi quadre, per semplicità di lettura:

- un primo metodo, caratterizzato da più passaggi di strato, è basato sull'esecuzione di una serie di distinti programmi di governo dei vari dispositivi di bordo (**fig. 1.2**), tra cui i cosiddetti *driver*, in minima parte residenti nello strato azzurro *[OS]* di **fig 3.1a**. Questo primo set di codici è completato da un più vasto insieme di programmi residenti nello strato arancione *[APPs]* di **fig. 3.1a** e da una *GUI* residente nello strato verde più esterno e più vicino all'*[UTENTE]*;
- un secondo metodo (**fig. 3.1b**), che è basato su uno specifico *middleware* denominato *ROS (Robot Operating System)* [7], è costituito da un insieme di servizi precaricati in memoria e tra loro integrati. Ciò consente, a diversi processi appartenenti allo stesso ambiente, di poter interagire tra

loro in maniera flessibile e ad un livello di *astrazione dell'hardware* più elevato [**5**]. Questo è un sistema che 'cortocircuita' la *GUI* all'*OS*, consentendo un 'salto' di strati intermedi senza pesare direttamente sull'utente finale. Esso è sicuramente un metodo più diretto e più efficiente rispetto al primo, ai fini dello sfruttamento delle risorse di sistema, in quanto una parte sostanziale delle comunicazioni e delle chiamate di sistema non sono più a carico del programmatore che è libero di concentrarsi sul *core* del modulo software da sviluppare.

Quanto descritto al primo punto rappresenta l'oggetto del presente lavoro. Seguendo tale metodologia, viene scritta una collezione di programmi, il cui assieme costituisce il cosiddetto *software di base*. Il codice sorgente ad esso associato, viene sviluppato utilizzando codifiche molto rigide (*linguaggi di programmazione*); esso fa capo a specifici utenti e viene raccolto in omonime cartelle.

Nonostante i suoi limiti, questo primo metodo, sia pur migliorabile, ha consentito di raggiungere i risultati preliminari esposti alla fine del presente documento. Questo software si è rivelato propedeutico al successivo sviluppo di un più avanzato software basato sulla seconda metodologia che implementa il Sistema di Controllo e che viene esaurientemente trattato ed approfondito in [**5**].

I programmi all'interno del VENUS possono presentarsi secondo quattro forme distinte, in base alla loro origine:

- *File eseguibili ottenuti da una cross-compilazione basata su linguaggio C*: questi file sono specificatamente compilati da un PC tradizionale, opportunamente configurato in base alla macchina di destinazione su cui dovranno girare i programmi. Per esempio, i primi modelli di *eCs* di cui era equipaggiata la primissima versione del prototipo, denominata VENUS 1, ha richiesto una cross-compilazione di programmi verso una architettura di tipo *ARM*;
- *File eseguibili ottenuti da una compilazione locale in sorgente C*: in questo caso si tratta di file il cui sorgente (con estensione **.c*) è presente sulla stessa macchina su cui si vuole effettuare la compilazione del programma e da cui poi ne verrà lanciato l'omonimo eseguibile corrispondente.
  Il più delle volte questo tipo di compilazione ha lo svantaggio di richiedere all'*OS* librerie specifiche che su sistemi *embedded* (come *eC1* ed *eC2*) potrebbero risultare difficili da reperire;
- *Script di shell o di bash shell* (in seguito, brevemente, script) che rappresenta una classe specifica di programmi concepiti per essere eseguiti all'interno di una *shell.* Ogni singola riga di codice richiamata da questi script creati a bordo del VENUS, richiede direttamente al *kernel* una determinata azione basandosi su istruzioni e comandi preesistenti nell'*OS* e/o richiamando programmi specifici;
- *Demoni*: sono dei particolari programmi, precaricati già in fase di avvio dell'*OS*, che possono essere generati secondo le tre suddette modalità e che lavorano in *background*, senza l'interazione dell'utente.

Prima di procedere alla rassegna dei vari programmi adottati a bordo dei VENUS di cui si è già fatto cenno, è bene fare un'ultima precisazione sulle configurazioni di base aggiuntive che i *OSs* installati sui computer hanno richiesto: sono stati scaricati applicativi preesistenti, di cui alcuni comuni a tutti i computer

presenti a bordo (quali per esempio *Network Time Protocol* e *Secure SHell*) e/o altri specifici all'uso (ad es. *tools* per la cattura video, utility per bus *I2C* e per driver video *v4l2*). Ciascuno di questi programmi viene installato proprio tenendo conto delle funzioni a cui i singoli *eCs* o *SBC* sono predestinati (prua, poppa, funzioni di supervisione e/o registrazione video).

5. Software di Base

Tutto il software sviluppato può essere schematizzato nelle seguenti quattro classi:

A. SCRIPT DI AVVIO *CUSTOMIZZATI* che vengono richiamati già nella fase di avvio dell'*OS*;

B. DEMONI, distinti per ogni computer e peculiari dei processi associati alla loro esecuzione;

C. PROGRAMMI SPECIFICI, distinti per ogni computer e peculiari dei dispositivi ad essi interconnessi;

D. HMI (*Human-Machine Interface*) che consente all'operatore remoto la completa interazione con la macchina. Questo programma di interfacciamento a finestre di dialogo, finalizzato all'effettuazione dei primi test sul veicolo, richiama ulteriori programmi basati sul software accennato ai due punti precedenti.

A – SCRIPT DI AVVIO *CUSTOMIZZATI*

Questi script di avvio sono detti *customizzati* perché intendono estendere le funzionalità dell'*OS* alle peculiarità del veicolo ospitante. Hanno in generale la funzione di compiere tutte le operazioni propedeutiche all'utilizzo delle varie interfacce di controllo presenti a bordo dei vari computer e, data la loro importanza, hanno la principale caratteristica di essere lanciati tutti nella sotto-fase di avvio dell'*OS*. Hanno il fondamentale compito di rendere visibili tutti i dispositivi all'interno dell'*OS*. Svolgono infine funzioni di: sincronizzare gli orologi di sistema; abilitare, inizializzare e definire il verso dei vari *GPIOs* impiegati; tenere traccia della durata delle singole sessioni per poterne stilare statistiche di utilizzo; resettare gli *hub-usb*; disattivare eventuali funzioni di risparmio energetico sui vari dispositivi connessi; configurare il sistema secondo le esigenze che ogni computer richiede in base alla funzione che è chiamato a svolgere (es. definire gli indirizzi ed il router di rete nel caso delle schede *Ethernet*).

Questi script svolgono anche la funzione di resettare tutte le condizioni di avviso/allarme che si fossero generate in precedenti sessioni e che in fase di avvio/riavvio non si ritiene opportuno vengano mantenute attive (*cfr.* "*reset_ALARM*" in **fig. 4.1**).

Questi script, proprio perché avviati con l'*OS*, sono tutti accomunati dalla caratteristica di essere lanciati a partire dallo stesso percorso di sistema (ovvero *"/etc/rc2.d"* in *OS Linux*) e richiedono quindi privilegi da *amministratore* per poter essere eseguiti correttamente. Senza questi script, non solo alcuni dispositivi non potrebbero essere riconosciuti dall'*OS*, rendendo nulla la possibilità di controllo da parte dei rispettivi

programmi a valle, ma sarebbe addirittura impossibile comunicare con l'intero veicolo in quanto mancherebbe una essenziale corretta definizione della *LVN* (**fig. 1.1**).

B – DEMONI

Questi programmi hanno la funzione di monitorare quei parametri critici che possono pregiudicare il buon funzionamento e la sicurezza del VENUS, quali autonomia energetica ed aspetti legati alla *Safety*. Il loro principale scopo è intervenire con determinate azioni, eventualmente anche di *halt*, come lo spegnimento degli *OSs* di bordo. Se ne offre una rapida panoramica:

B.1) *SOC* (acronimo di *State Of Charge*), programma residente su *eC2* e scritto in *C*, riveste un ruolo fondamentale per il monitoraggio continuo sullo stato di carica delle batterie. Monitora la corrente assorbita con una certa cadenza temporale, stimando la percentuale di carica delle batterie e memorizzandone in un file il valore; per svolgere al meglio il suo compito, inoltre, è anche in grado di riconoscere se il veicolo è impiegato in laboratorio, cioè alimentato a banco, congelando così l'avanzamento alla scarica.

È deputato anche a stilare statistiche relative ai tempi di scarica complessiva, ai cicli di carica e scarica ed a stimare l'autonomia residua delle batterie;

B.2) *SAFEMON* (*SERV* – relativo ai servizi di bordo), programma residente su *eC2* e scritto in *C*, opera in modalità semplificata il monitoraggio continuo del livello di carica delle batterie ma, a differenza del demone precedente, basa le proprie rilevazioni su livelli di criticità grossolani (livello di tensione sotto il 20% della carica massima piuttosto che sotto l'8%). Decide, inoltre, su eventuali azioni di *halt,* che possono portare fino allo spegnimento totale del veicolo se dovessero permanere condizioni in cui il gruppo batterie operi troppo vicino alla scarica completa;

B.3) *SAFEMON* (*NAV* – relativo alla navigazione), residente su *eC1* e scritto in *C*, effettua un monitoraggio continuo di parametri ritenuti critici, determinando il distacco del motore se si verificano determinate condizioni quali: anomalia lettura profondimetro; superamento di un tempo limite dall'ultimo avvio dell'elica; raggiungimento di una profondità limite. Altro compito cui assolve è decidere sulla partenza e sull'arresto della ventola di raffreddamento, condizionata a letture di temperatura ed umidità al di fuori di prefissati *range* considerati normali. Decide, infine, su eventuali azioni di *halt*, se dovessero permanere le condizioni critiche rilevate;

B.4) *SAFEMON* (*GEN* – di stato generale), residente sull'*SBC* e scritto in *C*, che, allo stato attuale, monitora le temperature dei singoli processori e della scheda medesima. In futuro potrà supervisionare altri parametri di carattere generale ritenuti critici. Segnala eventuali azioni da intraprendere agli altri computer in rete;

B.5) *FAN*, residente su *eC1* e scritto in *C*, si prende in carico l'azionamento della ventola interna di raffreddamento, leggendo lo stato a cui si deve portare a partire da un file a cui accedono in scrittura tutti i programmi che effettuano un monitoraggio termico (*cfr.* demoni ***B.3*** e ***B.4*** e programma TEMP in ***C.12***);

B.6) *PWM170_TEST*, residente su *eC2* (*cfr.* **fig. 4.1**) e scritto in *C*, si occupa della gestione e della segnalazione luminosa verso l'esterno (*[SOG]* e *[Ls]* in [**4**] – *cfr.* **tab. 1.1**) al fine di comunicare intuitivamente lo stato generale del VENUS (cioè se operativo, in allarme, in blocco o spento/guasto) e rendere così visibili eventuali variazioni di stato ritenute significative per l'operatore remoto.

I demoni descritti da ***B.1*** a ***B.5*** hanno in comune la caratteristica di avere almeno una chiamata al programma specifico *alarm* (*cfr.* **fig. 4.1**) che sarà approfondito nella sezione seguente, mentre ***B.6*** riassume l'effetto complessivo di queste diverse chiamate.

C – PROGRAMMI SPECIFICI

Questi programmi, che possono presentarsi sia in forma di script che di eseguibili compilati localmente, svolgono più nel dettaglio tutte quelle operazioni che si rendono necessarie al funzionamento di tutti i dispositivi collegati ai rispettivi computer. A tal proposito si fornisce un elenco dei principali programmi, tutti appartenenti allo strato *[APPs]* di **fig 3.1a**: alarm, bussola*, pressostato*, IMU*, motore*, timoni*, fari*, video*, ottico*, GPS, sonar, temp ed arresto/riavvio. I programmi identificati da un asterisco, costituiscono la base di lavoro per i corrispondenti nodi in *ROS* trattati esaurientemente in [**5**]. Per coerenza con quest'ultimo documento, si fa coincidere il nome dei programmi coi nomi dei corrispondenti nodi in *ROS.*

Per una più semplice lettura, in seguito si offre anche una rappresentazione grafica molto intuitiva di ogni programma.

C.1 – ALARM – FIG. 4.1

Residente su *eC2*, questo programma scritto in *C* e che include imprescindibilmente i file "*ID.txt*" e "*alarmTAB.h*", è cruciale per la sicurezza complessiva del veicolo. Esso demanda eventuali azioni correttive ad altri programmi che possono essere così tracciate.

Il primo file, residente su *eC2*, svolge una funzione centrale nell'utilizzo dei programmi del VENUS, custodendo l'*ID* (*IDentification number*) a cui tutti gli allarmi fanno riferimento.

Il secondo file, unico ed identico per tutti i computer di bordo e residente su ciascuno di essi, enumera tutti i possibili stati di funzionamento del VENUS e ne definisce le soglie di riferimento a bordo.

Il fondamentale compito del programma è riconoscere, quindi, i vari stati di funzionamento possibili dell'intero veicolo e le sue due principali funzioni possono essere così sintetizzate:

- tenere un registro di bordo su cui annotare tutti gli eventi di tipo informativo (*Info*), di avviso (*Warning*) e/o di allarme (*Alarm*) che si verificano nel corso del funzionamento;
- segnalare le più significative variazioni di stato intervenute nel sistema, sia secondo una logica di gravità che secondo una scala di priorità, attraverso un demone specifico (*cfr.* ***B.6***). Lo stato generale del VENUS risulta così sintetizzato e facilmente interpretabile da un operatore esterno

mediante opportuna codifica luminosa attraverso i dispositivi led *[SOG]* ed *[Ls]* richiamati in **tab. 1.1** e descritti in [**4**].

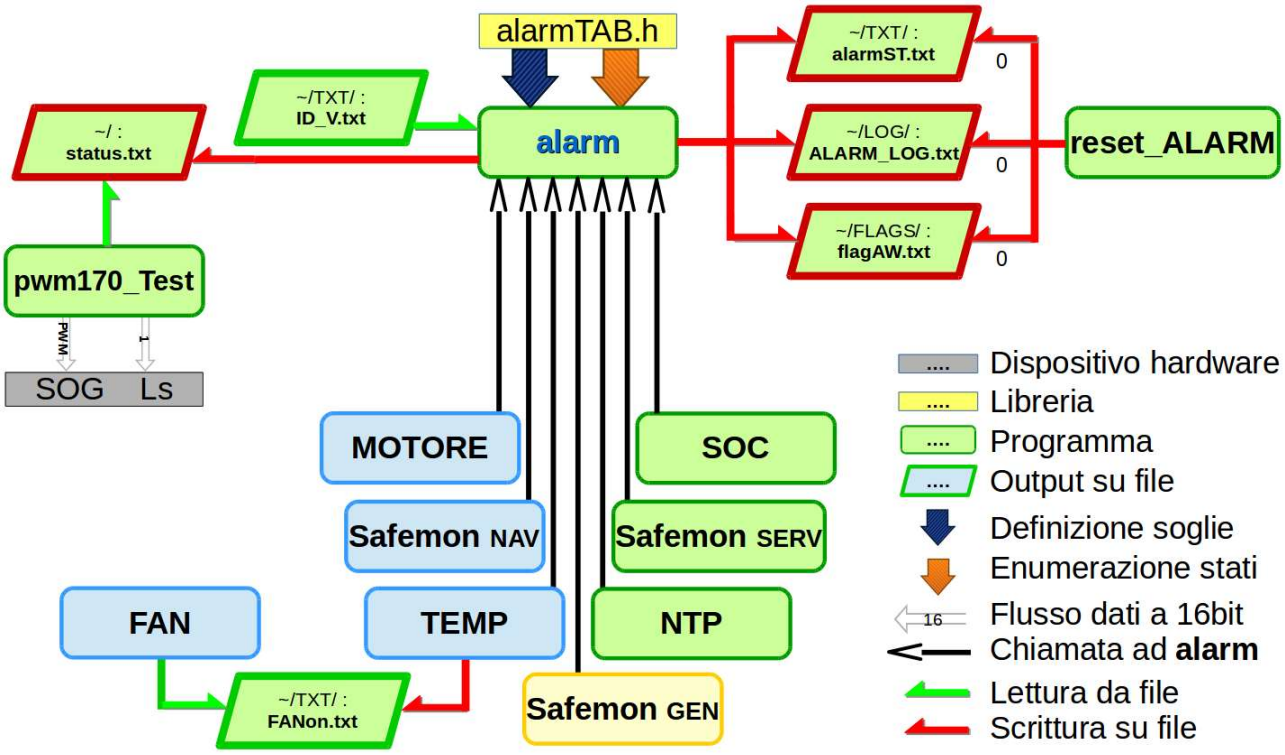


Figura 4.1 – Diagramma di flusso dei programmi *alarm*, *pwm170_Test* e *reset_ALARM*.
Si osserva che quest'ultimo script *customizzato* svolge la funzione di porre a zero gli allarmi ritenuti poco significativi nella fase di avvio del *OS* su *eC2*

C.2 – BUSSOLA* – FIG. 4.2

Residente su *eC1*, questo programma scritto in *C* ha il compito di interrogare la bussola digitale di bordo (*cfr.* **fig. 1.2**, *[COMPASS]* in [**4**]), potendone ricavare informazioni sulla versione del *firmware*, leggendone i dati ed avviandone la calibrazione su esplicito comando invocabile a terminale.

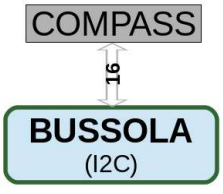


Figura 4.2 – Diagramma di flusso del programma *BUSSOLA*

A valle dell'*inizializzazione*, in fase di avvio dell'*OS* sul bus di comunicazione a cui è connessa, nel programma è lasciato all'utente facoltà di decidere come acquisire i dati, essendo previsto sul componente due possibili modalità di lettura (a 8 o a 16 bit); scegliendo la seconda opzione si sfrutta appieno la risoluzione pari a 12 bit, tale da garantire letture dell'orientamento rispetto al Nord con precisioni prossime al decimo di grado. In condizioni di marcia normale del veicolo, tuttavia, l'incertezza di misura varia tra ±0,5° e ±1° e questo si riscontra sperimentalmente essere legato all'influenza del campo magnetico rotante generato dal propulsore *brushless* a magneti permanenti quando sotto carico.

C.3 – PRESSOSTATO* – FIG. 4.3

Residente su *eC1*, questo programma scritto in *C* è deputato a leggere ed a memorizzare le rilevazioni del profondimetro (*cfr.* **tab. 1.1**, *[P0]* in [**4**]) nel momento in cui viene richiesta la sua prima esecuzione. Il programma effettua, con una certa cadenza temporale, acquisizioni di 100 campioni, restituendo al suo interno una media semplice. Il primo valore medio ricavato dai campioni rilevati nel momento del lancio del programma, costituisce il riferimento (corrispondente alla cosiddetta quota zero) per tutte le acquisizioni successive.

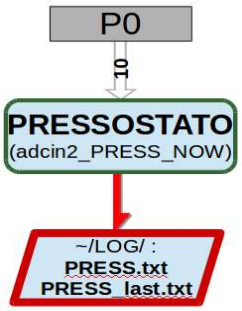


Figura 4.3 – Diagramma di flusso del programma *PRESSOSTATO*

Da questa prima lettura, compiuta quando il veicolo giace sul pelo libero dell'acqua, il programma calcola, per differenza, la variazione di pressione media (sempre su 100 campioni) letta alle iterazioni successive, deducendo, con opportune formule di conversione, la profondità percepita dal veicolo in quel dato momento. La dispersione dei 100 campioni rispetto alla loro media è di circa ±15cm.

Per consentire una profilatura puntuale dell'andamento della profondità col trascorrere del tempo, è previsto un salvataggio su file, sia dell'ultimo dato letto (per condividerlo eventualmente con altri programmi), sia del cronologico di tutte le misure effettuate.

C.4 – IMU* – FIG. 4.4

Residenti su *eC1*, questi programmi scritti in *C++* svolgono la funzione di interagire con l'unità IMU (*cfr.* **fig. 1.2**, *[IMU]* in [**4**]), verificandone l'efficacia del collegamento, ricavandone informazione sulla versione del *firmware* caricato sul dispositivo, avviandone una auto-calibrazione spaziale e leggendone le nove grandezze fisiche transitanti sulla porta USB.

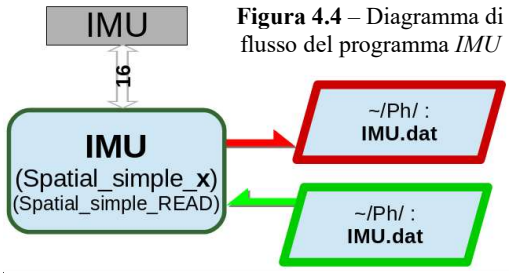


Figura 4.4 – Diagramma di flusso del programma *IMU*

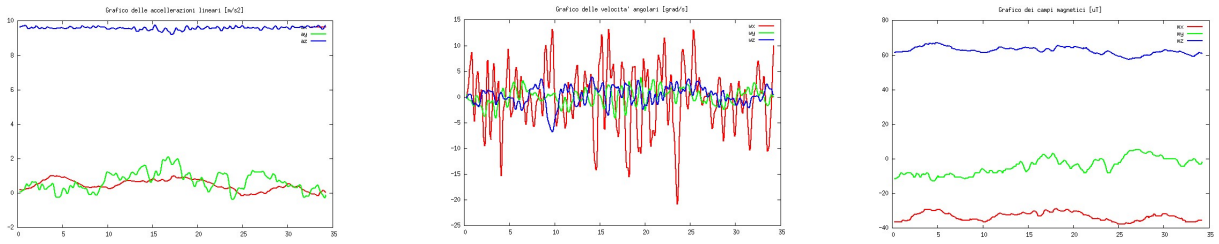
I dati sono inviati dall'unità secondo la frequenza impostata.

I due programmi (denominati *Spatial_Simple_GOOD* e *Spatial_Simple_LIGHT*) svolgono funzioni diverse, che sono:

- il primo, una funzione preliminare di test, restituendo a schermo tutte le grandezze misurate, compreso il calcolo e la visualizzazione *real-time* degli angoli di *roll*, *pitch* e *yaw*;
- il secondo, invece, registra i *timestamp* e tutte le grandezze misurate su un file (denominato *IMU.dat*), lasciando la possibilità all'utente di scegliere, già in fase di prima esecuzione del programma, con quale intervallo (impostabile da 8 a 64ms) campionare ed ogni quanto tempo scrivere sul file il gruppo di misure effettuate.

Entrambi i programmi si basano in modo prevalente su quanto già sviluppato dal produttore della IMU (che ne fornisce anche apposite librerie) al solo scopo di esempio applicativo [**8**] e test, e comunque nei limiti di licenza all'uso stabiliti in [**9**].

Un terzo programma, *Spatial_simple_READ*, rilegge i dati registrati dal secondo programma e ne traccia, su distinte finestre, i tre grafici corrispondenti ad accelerazioni, velocità angolari e campi magnetici, ognuno con gli andamenti nel tempo lungo le tre coordinate spaziali *x (traccia rossa)*, *y (traccia verde)* e *z (traccia blu)*. In aggiunta, questo programma calcola gli angoli di *roll*, *pitch* e *yaw* e ne visualizza il grafico corrispondente, basandosi sempre sul file salvato dal secondo programma (**fig. 4.5**).



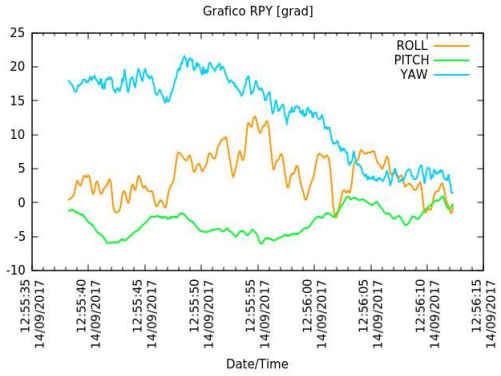


Figura 4.5
Visualizzazioni grafiche generate dal programma "*Spatial_simple_READ*";
in alto, da sinistra a destra, le schermate relative a: accelerometri, velocità angolari e magnetometri.

Sulla destra, la visualizzazione relativa al risultato del calcolo per:
ROLLIO – BECCHEGGIO – IMBARDATA

C.5 – MOTORE* – FIG. 4.6

Residente su *eC1*, questo programma scritto in *C* consente alla scheda elettronica *[S-MOT]* (*cfr.* [**4**]) con cui si interfaccia in via esclusiva il propulsore, di impostare i parametri legati al governo del *thruster* quali il verso di rotazione, l'inserimento del freno motore e la velocità di rotazione. Le funzioni di protezione motore per sovracorrente o stallo, *under-voltage lockout*, lettura errata dai sensori ad *effetto Hall*, disabilitazione *driver* di gestione motore e *thermal shutdown* sono gestite a bordo della *[S-MOT]* da opportuno integrato che si occupa anche del controllo di velocità ad anello chiuso, inseguendo un assegnato *set-point* di velocità fornito da un integrato dedicato ad 8 bit su bus *I2C* ed erogando più o meno corrente (con tecnica *Pulse Width Modulation – PWM*) sui tre avvolgimenti del motore per garantire costanza nella velocità di rotazione (*closed loop motor speed control*).

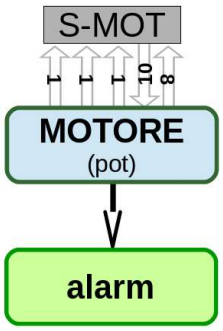


Figura 4.6 –
Diagramma di flusso del programma *MOT*

Gli allarmi eventualmente generati da *[S-MOT]* e gestiti in autonomia dalla scheda medesima, sono restituiti da un *circuito sommatore* al programma che ne può leggere in ogni istante il malfunzionamento e segnalarlo al programma *alarm*.

C.6 – TIMONI* – FIG. 4.7

Residente su *eC1*, è una coppia di programmi di test scritti in *C* che si prende in carico la gestione delle pinne, interfacciandosi tramite seriale col *SERVO Controller* citato in **fig. 1.2**. Tramite esso è possibile azionare i singoli *step-motor*, realizzando l'attuazione delle pinne citate in **fig. 1.2** e **tab. 1.1** (secondo due possibili risoluzioni, ad 8 o a 16 bit), sia singolarmente che in combinazione. In quest'ultimo caso, il programma consente, con un solo comando, di impostare specifici movimenti globali sul veicolo, quali virate, immersioni ed emersioni, azioni di frenata ed azzeramento delle pinne.

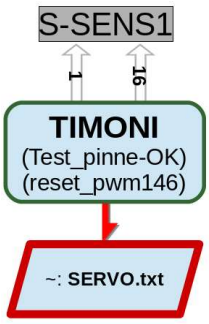


Figura 4.7
Diagramma di flusso del programma *TIMONI*

Il secondo si occupa di resettare il *SERVO Controller* nel caso si registrino dei malfunzionamenti.

E' previsto il salvataggio su file dei dati grezzi mandati dal computer al servo.

C.7 – FARI* – FIG. 4.8

Residente su *eC2*, questo programma scritto in *C* gestisce l'accensione e lo spegnimento dei fari (*cfr.* **fig. 1.2**), lasciando all'utente la possibilità di scegliere se azionarli in modalità continua o modulata in *PWM*. In questo secondo caso si è scelto di impostarne, una volta per tutte, il valore di modulazione ad un *duty-cycle* del 50% e con una frequenza di impulso a 100Hz. É facilmente dimostrabile che il valore del primo parametro è stato fissato quale miglior compromesso tra la potenza elettrica assorbita e l'efficienza luminosa complessivamente ottenuta, mentre il secondo è esclusivamente legato alla stabilità di funzionamento della serie di led pilotabili dall'integrato impiegato nel circuito elettronico, così come raccomandato dalla casa produttrice.

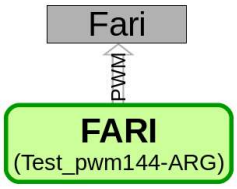


Figura 4.8 –
Diagramma di flusso del programma *FARI*

C.8 – VIDEO* (FFMPEG / MPLAYER) – FIG. 4.9

Residenti su *eC2*, costituiscono un gruppo di script che consentono chiamate dirette ad applicativi specifici preinstallati nell'*OS* che consentono di interagire con le *webcams* di bordo al fine di effettuare sequenze di registrazioni video o di istantanee e, ove la connessione di rete lo consenta, visualizzare in diretta (con una leggera latenza dovuta al processamento ed alla remotizzazione dell'immagine) ciò che vede il VENUS.

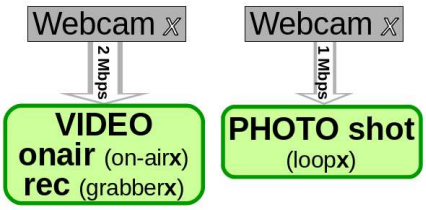


Figura 4.9 – Diagrammi di flusso dei programmi attinenti il flusso video

Per sequenze di registrazioni si intende la possibilità di selezionare da programma dei parametri quali il numero e la durata delle singole acquisizioni, nonché il tempo che deve intercorrere tra un'acquisizione e l'altra. L'installazione di specifici *tools* (accennati alla fine del capitolo precedente), inoltre, consente anche di impostare nel dettaglio, ove fosse necessario, tutti i parametri delle telecamere quali: fuoco, nitidezza,

contrasto, ecc… La chiamata ai programmi specifici che seguono è subordinata all'installazione di eventuali pacchetti software e *codecs*.

In una prima fase, tra le istruzioni all'interno dello script, si è usato il programma *mplayer* [**10**], individuato in rete per la sua sintassi semplificata e per l'immediatezza nei comandi. In una seconda fase, date le aumentate richieste di performance alle telecamere e la contestuale aumentata potenza di calcolo del sistema, si è deciso di potenziare l'hardware mediante un *SBC* (*cfr.* **tab. 2.1**) a bordo, espressamente dedicato all'elaborazione video, in aggiunta alle *eCs* preesistenti. Si è potuti così passare dal generico *mplayer* alla potente e versatile piattaforma *ffmpeg* [**11**].

In questo modo si è raggiunta una ottima qualità video e buon fattore di compressione del flusso dati: in Laboratorio si è ottenuto un flusso video di *1920x1080@15fps* ad un *baudrate* prossimo ai 2MB/s su immagini in movimento, indipendentemente dalla lunghezza della registrazione, di qualità decisamente superiore alla precedente. Con particolare riferimento al salvataggio di file per video e foto effettuati dagli script, questi conservano nel proprio nome (per ogni frammento di sequenza) data e ora di acquisizione ed una sigla (*L* o *R*) a seconda se il flusso video trae origine dalla telecamera sinistra o destra, venendo poi archiviati in cartelle dedicate.

C.9 – OTTICO* – FIG. 4.10

Residente su *eC2*, questo programma scritto in *C*, è dedicato alla futura gestione del modem ottico (*Underwater Optical Communication System – UWOC* – attualmente, ancora in fase di sviluppo presso il Laboratorio). É dedicato a gestire il settaggio dei parametri, la modalità di funzionamento e la ricezione/invio di file dal/al dispositivo mediante porta seriale su standard USB e si interfaccia con il *firmware* di Arduino Due [**12**] che controlla il modem. Per evitare fastidiose ed indesiderate accensioni dei potenti led che caratterizzano i circuiti di trasmissione del dispositivo a valle di Arduino, in parallelo al cavo seriale su standard USB, è stato previsto un canale di sicurezza aggiuntivo che gestisce un relè, incaricato di attivare/disattivare la tensione principale che alimenta i led.

Eventuali file trasmessi da un altro veicolo mediante il sistema di comunicazione, vengono salvati su un file generico, mentre eventuali aggiornamenti al *firmware* di Arduino Due possono essere effettuati localmente tramite *tool* specifico.

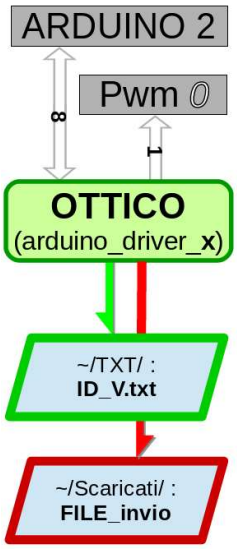


Figura 4.10
Diagramma di flusso del programma *OTTICO*

C.10 – GPS – FIG. 4.11

Residente su *eC2*, è un programma scritto in *C* che rende disponibili al sistema i dati di posizionamento globale forniti dai satelliti geostazionari all'antenna GPS (**fig. 1.2**); il segnale, opportunamente ricondizionato dall'elettronica di bordo,

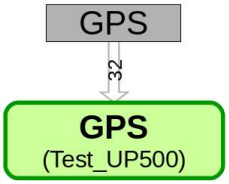


Figura 4.11
Diagramma di flusso del programma *GPS*

giunge al computer tramite un collegamento seriale su standard USB che processa il dato secondo protocollo *NMEA* (*National Marine Electronics Association*).

Grazie a questo programma è possibile verificare la funzionalità hardware del dispositivo e visualizzare tutti i dati di posizionamento globale secondo il protocollo menzionato, aggiungendo la possibilità, in presenza di segnale valido e con opportuno programma aggiuntivo, di sincronizzare gli orologi di sistema all'ora internazionale (*UTC – Coordinated Universal Time*). E' stata predisposta la possibilità di scrittura su file per la condivisione di queste informazioni con altri programmi.

C.11 – SONAR – FIG. 4.12

Residenti su *eC2*, questa coppia di programmi scritti in *C++* svolgono la funzione di interagire col sonar panoramico (*cfr.* **fig. 1.2**, *[S0]* in [**4**]), mediante specifico protocollo fornito dal produttore del dispositivo acustico, *inizializzandolo* alla prima connessione**.** Settandone correttamente i parametri di configurazione (settore circolare da spazzolare, numero di campioni da acquisire per ogni fascio acustico, guadagno, risoluzione angolare e range di misura) con una risoluzione impostabile a 4 o a 8 bit, opportunamente correlati ai corrispondenti istanti di acquisizione (*timestamp*) che il dispositivo sonar genera di volta in volta.

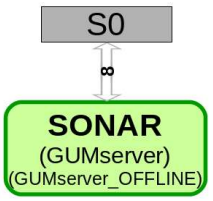


Figura 4.12 – Diagramma di flusso del programma *SONAR*

I programmi corrispondenti a questa funzione (denominati *GUMserver* e *GUMserver_OFFLINE*) svolgono, di concerto con l'omologo programma *client*, residente sul PC remoto (**figg. 1.1** e **1.2**), due funzioni diverse, che sono:

- il primo, la funzione di server, visualizzando, in presenza della connessione WiFi esistente (*LVN*), sullo schermo dell'operatore remoto, una traccia acustica che varia secondo i parametri impostati;
- il secondo, la funzione di registrare tutti gli echi restituiti dal sonar in corrispondenza dei relativi *timestamp* su un file (denominato, ad es. *logG20Range2Passo5BINS200.bin*), secondo i parametri impostati.

I programmi descritti, così come il programma *client* che viene lanciato sul PC remoto, sono basati sul lavoro di tesi di M. Martinelli. Nel suo esaustivo lavoro [**13**] si può approfondire questo argomento.

A questa terna di programmi, infine, se ne aggiunge un quarto (*client_READ*, variante del programma *client*) che risiede anch'esso sul PC remoto, attraverso il quale è possibile visualizzare a schermo (*off-line*) i dati registrati sul file binario generato dal secondo programma (**fig. 4.13**).

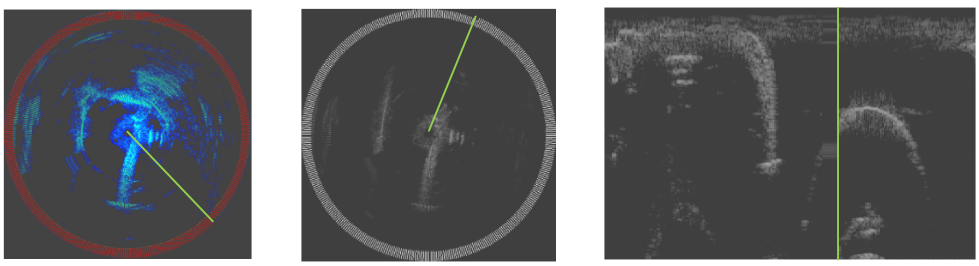


Figura 4.13 – Varie scansioni acustiche (echi) visualizzati sulla *Stazione di Controllo* dell'operatore esterno ottenuti indistintamente dal programma "*client*" o "*client_READ*"; nell'ultima scansione a destra è distinguibile il bordo ovale della piscina

C.12 – TEMP – FIG. 4.14

Residente su *eC1*, questo programma scritto in *C* legge dall'elettronica interconnessa alla sonda di temperatura ed umidità di bordo (*cfr.* **tab. 1.1**, *[H0]* in [**4**]) i valori di interesse, registrando, mediante scrittura su file, tutte le misure effettuate a partire dal momento in cui il programma viene lanciato e fino a quando viene lasciato attivo il processo corrispondente. Al verificarsi di determinate situazioni ritenute critiche, esso modifica un *flag* sul file *FANon.txt*, per fare in modo che il demone *FAN* avvii la ventola di raffreddamento. Parte del codice è stato reimpiegato nel demone *Safamon* citato al capitolo precedente.

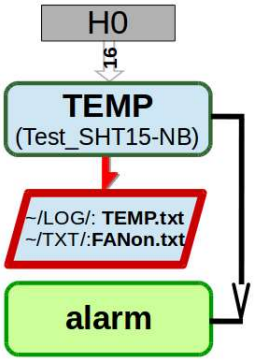


Figura 4.14
Diagramma di flusso del programma *TEMP*

C.13 – ARRESTO / RIAVVIO

Residenti su ogni computer di bordo, questi script effettuano l'arresto/riavvio dell'*OS* locale dopo aver inviato i segnali di arresto/riavvio a tutti gli altri computer di bordo, consentendo un arresto/riavvio sicuro e coordinato del sistema nel suo complesso.

Si osserva che tra loro questi tre script condividono gli stessi comandi base, ma sono tutti diversi, realizzando ciascuno una diversa sequenza di arresto/riavvio, a seconda del computer da cui viene lanciato.

D – HMI

L'interfaccia uomo-macchina (*Human-Machine Interface*) è lo strumento che consente all'operatore di interagire con la macchina. Essa è basata sulle librerie *open source* di *QT* ed *OpenCV* ed è installata su un PC portatile (*Stazione di Controllo*) dotato di link WiFi. L'interfaccia si presenta suddivisa in più sezioni, ciascuna visualizzata in una propria finestra (*dialog box*).

Questo particolare applicazione, in presenza di un robusto canale di comunicazione, consente di:

- passare al robot comandi elementari (**MW** e **FINS** - **fig. 5.1** e **5.5**);
- impostare i principali parametri del robot (**SET** e **SONAR** – **figg. 5.2** e **5.3**);
- visualizzare le letture dei sensori di bordo (**SET** e **PAR** – **figg. 5.2** e **5.4**);
- programmare e lanciare le missioni (**PROG** e **CMP** – **figg. 5.8** e **5.9**);
- stabilire le modalità di acquisizione di foto/video ed effettuare il download delle registrazioni (**MW**, **VIDEO** e **PHOTO** – **figg. 5.1**, **5.6** e **5.7**).

Questa interfaccia è completata da un programma, *wifi_VenusX*, sempre residente sul PC portatile, che consiste in una serie di script, ognuno associato alla configurazione WiFi di un determinato veicolo. Tale script contiene istruzioni elementari ed essenziali che consentono di configurare correttamente la connessione alla macchina con la quale si è scelto di comunicare (la ***X*** identifica l'*ID* del VENUS selezionato – *cfr.* **fig. 1.1**). Si noti come il fallimento nell'instaurare il collegamento radio associato alla chiamata di questo script si traduca nell'espressione "*NOT CONNECTED*" visualizzata sotto il *menu a tendina* collocato al centro della finestra principale sotto raffigurata (**MW** – particolare di **Fig. 5.1**).

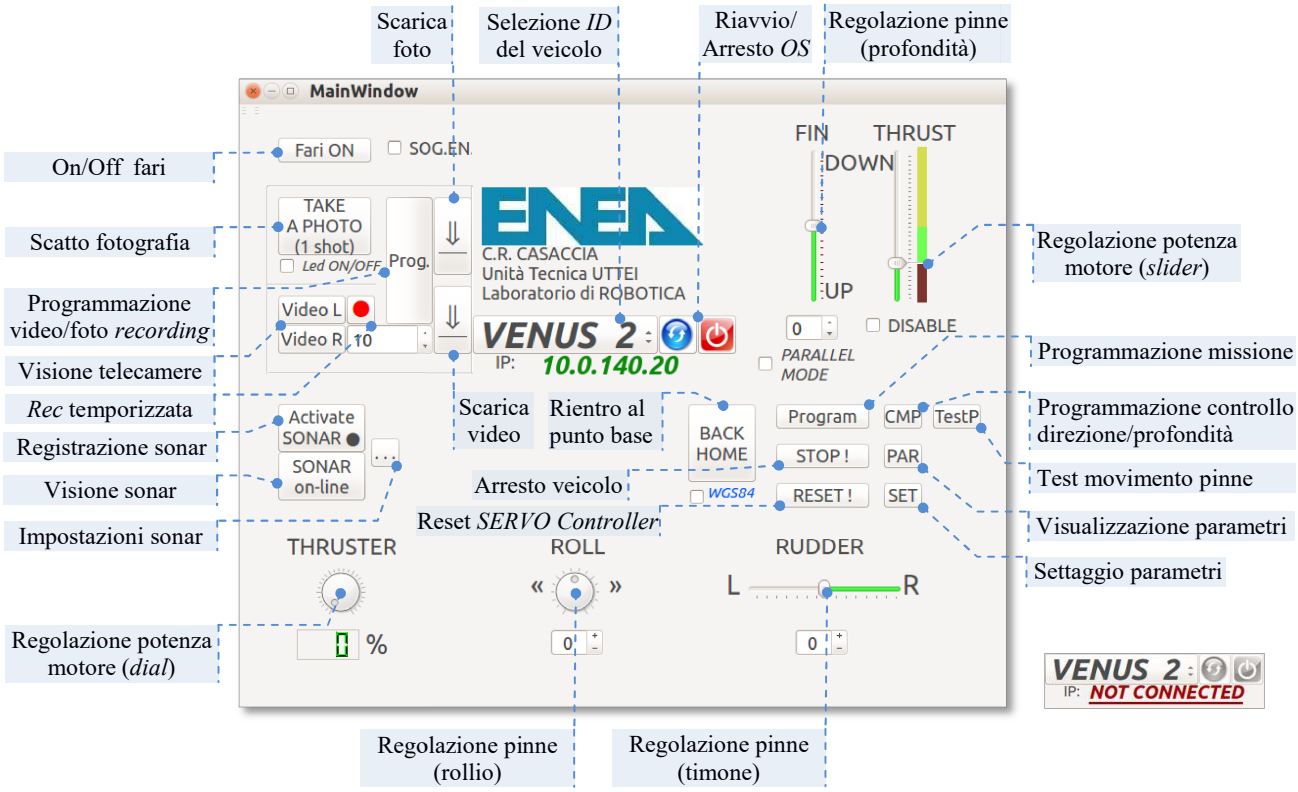
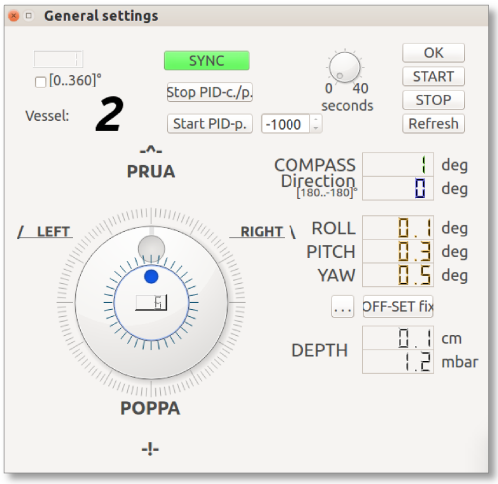


Figura 5.1 (MW) – Finestra di dialogo principale dell'interfaccia uomo-macchina (*HMI*) del VENUS, pienamente operativa sul PC remoto dell'operatore, in presenza di segnale WiFi ottimale. Nel particolare in basso a destra è illustrato il risultato di una connessione WiFi fallita nel momento in cui si sceglie l'*ID* del veicolo col quale comunicare



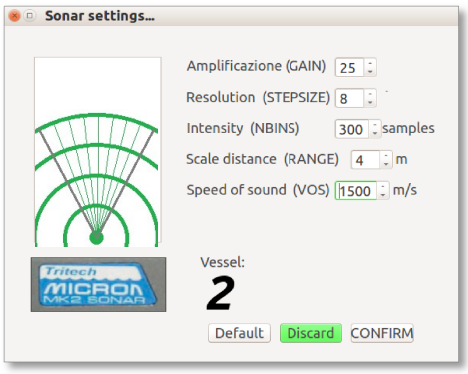
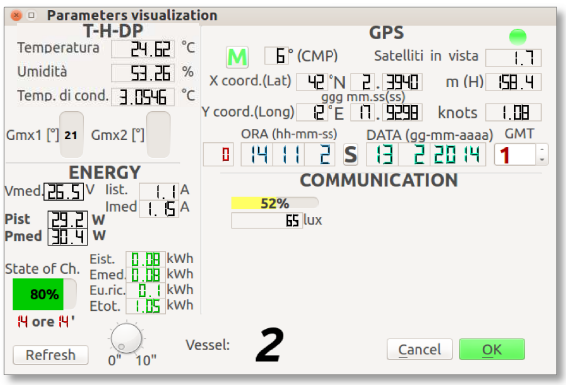


Figura 5.3 (SONAR) – Impostazioni parametri sonar

<<< **Figura 5.2 (SET)** – Settaggio parametri generali



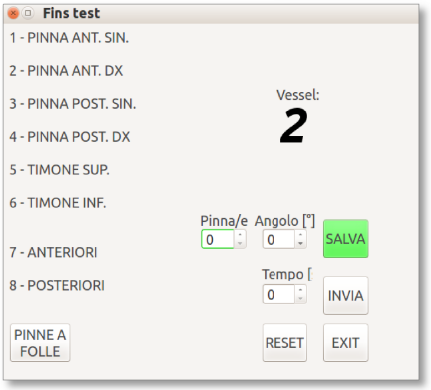


Figura 5.4 (PAR) – Visualizzazione parametri generali

Figura 5.5 (FINS) – Test movimento singole pinne

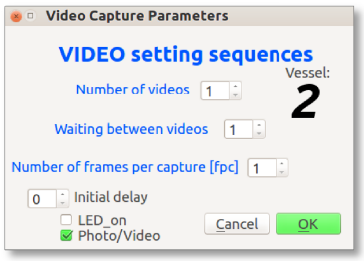


Figura 5.6 (VIDEO) – Programmazione delle sequenze di registrazioni video

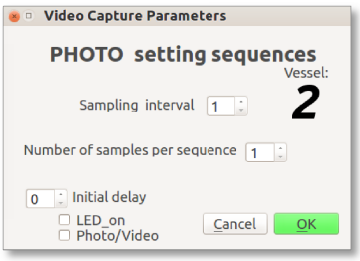


Figura 5.7 (PHOTO) – Programmazione delle sequenze di istantanee

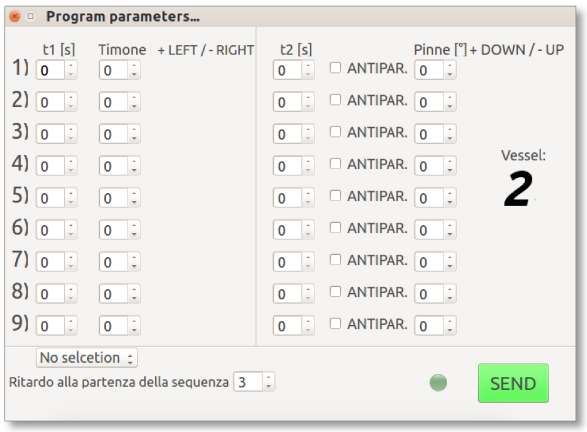


Figura 5.8 (PROG) – Programmazione delle traiettorie *senza* Sistema di controllo (*open loop*)

Figura 5.9 (CMP) – Programmazione delle traiettorie con Sistema di controllo minimale ad obiettivi (direzione *o* profondità)

Si precisa che questa interfaccia può essere vista come un insieme di programmi residenti nello strato più vicino all'*[UTENTE]* di **fig. 3.1a** ed è suddivisibile in due ulteriori sotto-categorie:

- un programma a finestre di dialogo (*object-oriented*), scritto in *C++*, che realizza la parte grafica dell'interfaccia vista precedentemente (*GUI*) e che rende fruibili le finestre (**figg. 5.1÷5.9**) sullo schermo della *Stazione di Controllo* attraverso cui l'operatore può interagire col veicolo in modo piuttosto semplice ed intuitivo;
- un insieme di programmi (*robot-oriented*), residenti sui rispettivi computer, tipicamente scritti in *C*, finalizzati esclusivamente all'esecuzione di *tasks* associati all'interazione dell'operatore con l'interfaccia grafica (*GUI*). I sorgenti di questo set di programmi rappresentano una estensione o "combinazione" dei programmi di origine afferenti entrambi al sotto-raggruppamento ***C***.

Riguardo al secondo punto, tra questi programmi, se ne passano in rassegna i più significativi:

D.1) *REBOOT/SHUTDOWN* (**MW – Fig. 5.1**), residente su *eC1*, richiama uno degli script visti in ***C.13*** per realizzare il riavvio/arresto di tutti gli *OSs* di bordo;

D.2) *loop/loop1* (**PHOTO – Fig. 5.7**), residente su *eC2* è lo script a cui vengono passati gli argomenti che definiscono le sequenze di istantanee viste in ***C.8***;

D.3) *grabber/grabber1* (**VIDEO – Fig. 5.6**), residente su *eC2* è lo script a cui vengono passati gli argomenti che definiscono le sequenze di acquisizione video viste in ***C.8***;

D.4) *datetime-QT* (**SET** – **Fig. 5.2**), residente su *eC2*, programma scritto in *C* che rende disponibile data e ora aggiornate del PC remoto all'*NTP* server, per la sincronizzazione di tutti i computer di bordo;

D.5) *delay-QT* e *GetFiles* (**PROG** – **Fig. 5.8**), residenti su *eC2*, è una coppia di programmi scritti in *C* con i quali viene gestita la fase di programmazione della sequenza di traiettorie da far compiere al veicolo, mediante memorizzazione locale delle singole tratte da eseguire. Trascorso un certo ritardo dal lancio del primo programma (corrispondente alla pressione del tasto "SEND" sulla finestra di **fig. 5.8**), viene eseguito il secondo programma che pone a carico del veicolo l'esecuzione delle traiettorie specificate al passo precedente;

D.6) *pinne-QT*, *motore-QT*, *roll-QT* e *inchioda-QT* (**MW / PROG** – **Figg. 5.1** e **5.8**), residenti su *eC1*, sono tutti programmi scritti in *C* che rendono operativa la fase di programmazione delle traiettorie in *open loop*, ovvero senza implementare il Sistema di controllo del veicolo trattato in [3];

D.7) *TestPinne-QT* (**FINS** – **Fig. 5.5**), residente su *eC1*, è un programma scritto in *C*, estensione del programma *TIMONI* trattato in ***C.6***, che consente le operazioni specificate sulla relativa finestra, passando al programma le posizioni angolari che le pinne devono assumere, tramite file denominato *TestPinne.txt*;

D.8) *BackHome* (**MW** – **Fig. 5.1**), residente su *eC2*, è un programma scritto in *C* che legge le coordinate di riferimento salvate su un file (denominato *HOME*) e porta il veicolo al punto base stabilito, alla pressione del corrispondente pulsante, posto sulla finestra di dialogo principale (**fig. 5.1**);

D.9) *PID-comp* (**SET / CMP** – **Figg. 5.2** e **5.9**), residente su *eC1*, è un programma scritto in *C* che legge i dati della bussola e, sulla base della differenza tra direzione impostata e letta, in presenza di segnale radio valido, corregge in tempo reale i timoni in modo da raggiungere il *target* di direzione impostato per ciascun tratto della sequenza;

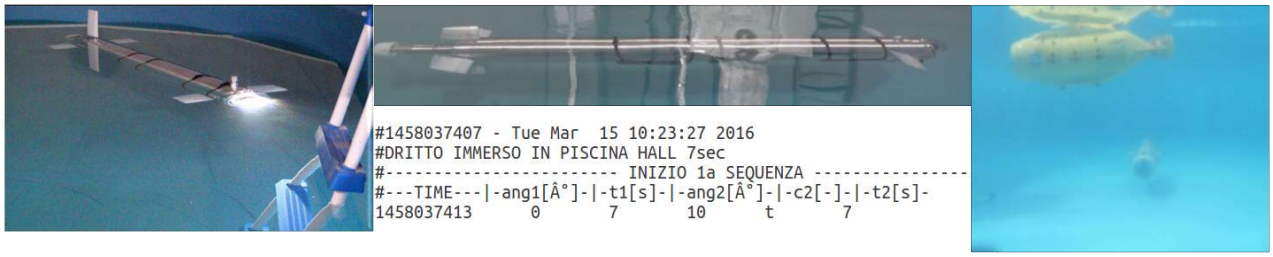
D.10) *PID-press* (**CMP** – **Fig. 5.9**) , residente su *eC1*, è un programma scritto in *C* che legge i dati dal profondimetro e, sulla base della differenza tra quota impostata e letta, corregge in tempo reale le pinne di profondità in modo da avvicinarsi al *target* di immersione impostato per ciascun tratto della sequenza.

6. Test sperimentali in Laboratorio

Il Laboratorio di Robotica è dotato di una piscina lunga 7m, larga 2,4m e profonda circa 1m, già impiegata per effettuare dei primi test funzionali e per equilibrare con opportune zavorre la prima versione prototipale di VENUS [**4**].

Nonostante le relativamente ristrette dimensioni che la caratterizzano, appena sufficienti ad offrire limitata libertà di movimento al VENUS, si è comunque potuto condurre le prime prove tese a dimostrare la possibilità del veicolo di essere interamente governato dalla *GUI* sin qui illustrata, allo scopo di organizzare in seguito delle uscite al lago. Da questa prima fase si sono potute valutare le performance complessive del veicolo, nell'ottica di migliorarne le sue capacità funzionali ed apportare eventuali modifiche all'interfaccia ed a tutto il software di base ad essa correlato, per ridurne le criticità.

Con le prime prove in piscina si sono potute tuttavia effettuare solo quelle operazioni di test dell'interfaccia compatibili con le problematiche precedentemente esposte, come ad esempio: comandi di accensione e spegnimento fari (**fig. 6.1a**), acquisizione dalle telecamere mediante programmazione di una sequenza video (**fig. 6.1c**) e, infine, un lancio di prova programmato (**fig. 6.1b**) mediante l'interfaccia riportata in **fig. 5.8**. Riguardo a quest'ultima prova, si riporta un estratto testuale che sintetizza i comandi passati al veicolo per consentirgli di compiere una corsa completa in piscina, in immersione, della durata di 7s, con fase di arresto finale (**fig. 6.1b**).



(a) ***(b)*** ***(c)***

Figura 6.1 – Istantanee relative all'uso della *GUI* ***a)*** accensione fari ***b)*** lancio di prova nella piscina del laboratorio e ***c)*** foto di una delle due telecamere di bordo durante una immersione, estratta da una registrazione video programmata

Già in questa primissima fase di test, sono emerse due importanti problematiche:

- l'imprevisto continuo slittamento dell'azionamento rispetto al giunto di accoppiamento con l'alberino della pinna, che è stato risolto rinforzando l'innesto tra giunto ed azionamento. Questo ha richiesto la ritaratura della cosiddetta posizione neutra (0°) della pinna per via software, da cui è derivata la necessità di creare una *dialog-box* specifica sulla *GUI* per individuare l'*off-set* necessario a ripristinare la posizione neutra di ciascuna pinna (*cfr.* **fig. 5.5**);
- la scarsa qualità del video acquisibile tramite *webcam* (**fig. 1.2**) che, contrariamente alle foto, raggiungeva una risoluzione massima piuttosto bassa, di soli *320x240@15fps*, imputabile alle limitate risorse di sistema disponibili sulla *eC2* adottata. Essa, infatti, veniva investita dall'intero flusso video, in mancanza di una scheda grafica (*cfr.* **Tab 2.1**). Anche durante le registrazioni video

si sono osservate le stesse limitazioni. Successivamente, prove mirate hanno consentito di escludere la possibilità che la velocità di scrittura sull'unico supporto interno di memoria disponibile non fosse sufficiente allo scopo. Da qui l'idea di adottare un terzo computer *SBC* che aumentasse anche la capacità di calcolo a bordo è stata risolutiva.

Valutata la possibilità di una prima uscita al lago, un ruolo essenziale, come già in [**4**], ha rivestito l'uso della piscina per la delicata operazione di zavorramento del VENUS che è stato poi oggetto di prove in acque libere, ovvero il VENUS 2.

Prima di eseguire le prove del VENUS al lago di Bracciano, individuate le giuste zavorre (opportunamente collocate all'interno del tubo di pressurizzazione), è stato messo a punto un programmino di monitoraggio scritto in *C* che consentisse di registrare i parametri letti e scaricabili a fine missione, rendendoli disponibili su un file denominato *D_D.txt*.

Questo file si è aggiunto ad un altro denominato *TRAJ.txt*, di cui in **fig. 6.1b** si riporta un estratto, allo scopo di registrare le traiettorie, già previsto nella fase di esecuzione del programma connesso all'interazione con l'interfaccia di programmazione (*cfr*. ***D.5***).

7. Test sperimentali in acque libere

Il Laboratorio di Robotica ha attiva una collaborazione con la Base dell'Aeronautica Militare "Luigi Bourlot" in Vigna di Valle che mette gentilmente a disposizione uno specchio d'acqua per la conduzione delle prove. Questo è vero soprattutto per la seconda fase di *prove sperimentali*, ovvero quel set di test ampiamente trattate in [**5**], così identificate perché avvenute a valle di una serie più cospicua di modifiche sul veicolo (*cfr.* **8**).

Vengono qui descritte alcune *prove* sperimentali *preliminari* legate al test del software di base ed all'affinamento dei sistemi hardware di bordo, per aumentare l'affidabilità dell'intero veicolo e stabilizzare il suo comportamento nel corso dei successivi test.

Le prove a cui ci si riferisce in questo documento sono state effettuate nel corso di due specifiche campagne di misura al lago di Bracciano:

- la prima, condotta in prossimità di Trevignano Romano, col supporto della locale Hydra Ricerche [**14**] che ha fornito barche di supporto ed un sub costantemente immerso col veicolo;
- la seconda, condotta presso la Base Militare di Vigna di Valle.

Il metodo prescelto per l'esecuzione di una singola missione (detta anche sequenza) prevede la suddivisione del relativo percorso in un numero finito di tratti elementari (definiti semplicemente tratti), caratterizzati ognuno da singole azioni da far compiere al robot all'interno di un tempo prefissato.

Alla fine di ogni tratto viene aggiunta una riga sul file *TRAJ.txt* all'interno della quale viene registrato il *timestamp* ed i parametri caratteristici del tratto successivo.

Alla fine di ogni missione, precedentemente salvata o impostata all'occorrenza ed inviata a bordo, viene prevista una fase di arresto (ben visibile nella **fig. 7.2a**) che consiste nel posizionare le quattro pinne di poppa a 90° per qualche secondo, tenendo frenato il motore fino alla partenza di una nuova missione.

PRIMA CAMPAGNA

In questa prima serie di test per prima cosa si è provveduto a predisporre la registrazione dell'unico dato disponibile durante la navigazione, ovvero quello della bussola.

All'epoca, infatti, pur essendo disponibili i dati sul profondimetro *[P0]*, non si erano osservati valori numerici sufficientemente coerenti e stabili e non si è quindi ritenuto opportuno registrarli. Non erano ancora disponibili, inoltre, neanche i dati dell'*[IMU]*, che sono stati invece acquisiti a partire dalla campagna successiva, a valle di importanti modifiche sulla struttura esterna ed interna del VENUS 2.

Posizionato il veicolo in acqua (**fig. 7.1**), l'assetto è risultato piuttosto appruato. Nell'istante in cui il motore è partito, con le pinne di profondità in posizione neutra rispetto alla direzione di marcia, il veicolo ha cominciato a compiere una lieve immersione. Questo comportamento è stato compensato dal cilindretto porta-antenne (*cfr. [SOG]* in **tab. 1.1**) posto sulla sua sommità (normalmente affiorante in superficie a veicolo fermo), che ne ha variato l'assetto idrodinamico complessivo una volta immerso. A velocità di regime, infatti, si è approssimativamente ripristinato l'assetto di marcia neutro (PITCH≈0°). In questa prima campagna di prove si sono svolti più lanci, impostando una serie di sequenze predefinite consistenti rispettivamente in:

- una prima traiettoria rettilinea percorsa solo in superficie; durata: 30s;
- un cerchio stretto (sempre in superficie); durata: 103s;
- un mezzo cerchio in superficie ed un primo breve tratto in immersione; durata: 45s.

Concluse con successo queste sequenze preliminari, si è proceduto ad effettuare una serie di test in immersione, di cui si riportano le due più significative.

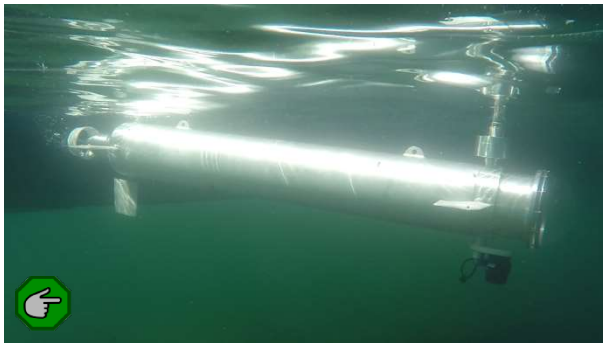


Figura 7.1 – Istantanea relativa al primo lancio del 4 aprile 2016: è ben visibile la cavitazione provocata dall'elica all'atto della partenza

Di seguito sono raffigurate le istantanee connesse all'esecuzione della sequenza denominata *GIRO IMMERSO LENTO* (**figg. 7.2a** e **7.2b**).

Dal file delle traiettorie (*TRAJ.txt*) registrato sul veicolo si può vedere come, a parte un primo tratto di soli 5s con le pinne di profondità neutre (0°), per 58s viene impostato il timone a 18° per far virare a destra e consentirne l'immersione con angolo impostato a 10° sulle pinne di profondità, effettuando così un percorso a spirale; il VENUS ha continuato a ruotare alla quota raggiunta (cerchio immerso) per altri 58s, per finire con la fase di emersione (pinne di profondità a -10°) per la restante durata di 30s.

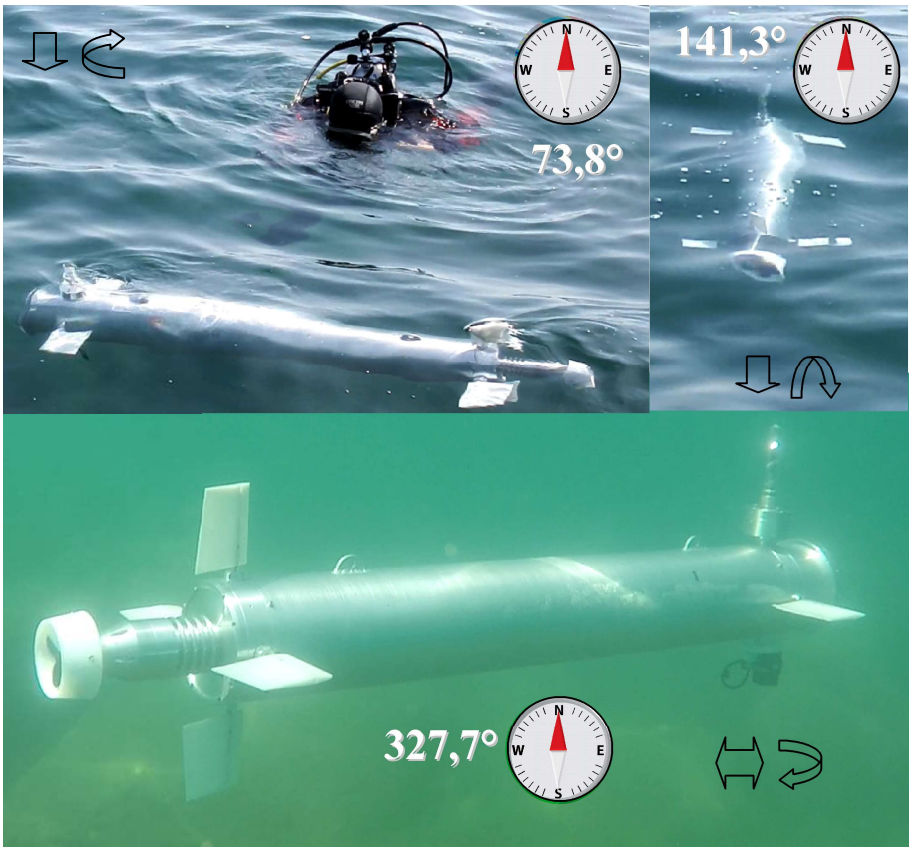
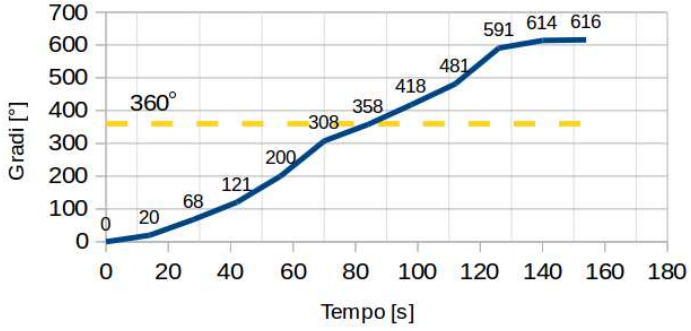


Figura 7.2a – Istantanee della sequenza denominata *GIRO LENTO IMMERSO* (4 tratti); dall'alto, in senso orario: esecuzione del 2° e del 3° tratto. Si riporta anche la direzione indicativa risultante dal file "*D_D.txt*" riportato in **fig. 7.2b**

```
#1459780875 - Mon Apr  4 14:41:15 2016
#GIRO LENTO IMMERSO - DSCF3021.MOV+DSCV3022.MOV+GOPR9475.MP4
#------------------------ INIZIO 1a SEQUENZA -----------------------
#---TIME---|-ang1[°]-|-t1[s]-|-ang2[°]-|-c2[-]-|-t2[s]-
1459780880      0       5       0       t       5
1459780938      -18     58      10      t       58
1459780996      -18     58      0       t       58
1459781026      0       30      -10     t       30
```

"*TRAJ.txt*"




```
#1459778628 - Mon Apr  4 14:03:48 2016
#GIRO LENTO IMMERSO
#---TIME---|-Dir[°]-
1459780870      73.80
1459780884      93.60
1459780898      141.30
1459780912      194.50
1459780926      274.10
1459780940      21.30
1459780954      71.40
1459780968      132.00
1459780982      195.10
1459780996      304.50
1459781010      327.70
1459781024      329.80
```

"*D_D.txt*"

Figura 7.2b – Estratto numerico delle traiettorie impostate in **fig. 7.2a** delle rilevazione della bussola e del corrispondente grafico della direzione rilevata, traslata a 0° rispetto al valore iniziale di 73,8°. Dal grafico è visibile come il veicolo, nel corso della missione così impostata, durante i 196s occorsi per l'esecuzione del tratto in rotazione, compia quasi due giri completi (circa 600° su 2x360°)

Nelle figure **7.3a** e **7.3b**, sono rappresentate le immagini relative alla seconda sequenza impostata, denominata *DISCESA [DRITTO]*. Nel file delle traiettorie si può vedere come, a parte un primo tratto di soli 5s con le pinne di profondità posizionate per restare in superficie (-10°), per 20s vengono portate di 10° in basso, per poi procedere in posizione neutra per 10s e poi a riemergere (pinne a -10°) per altri 20s.

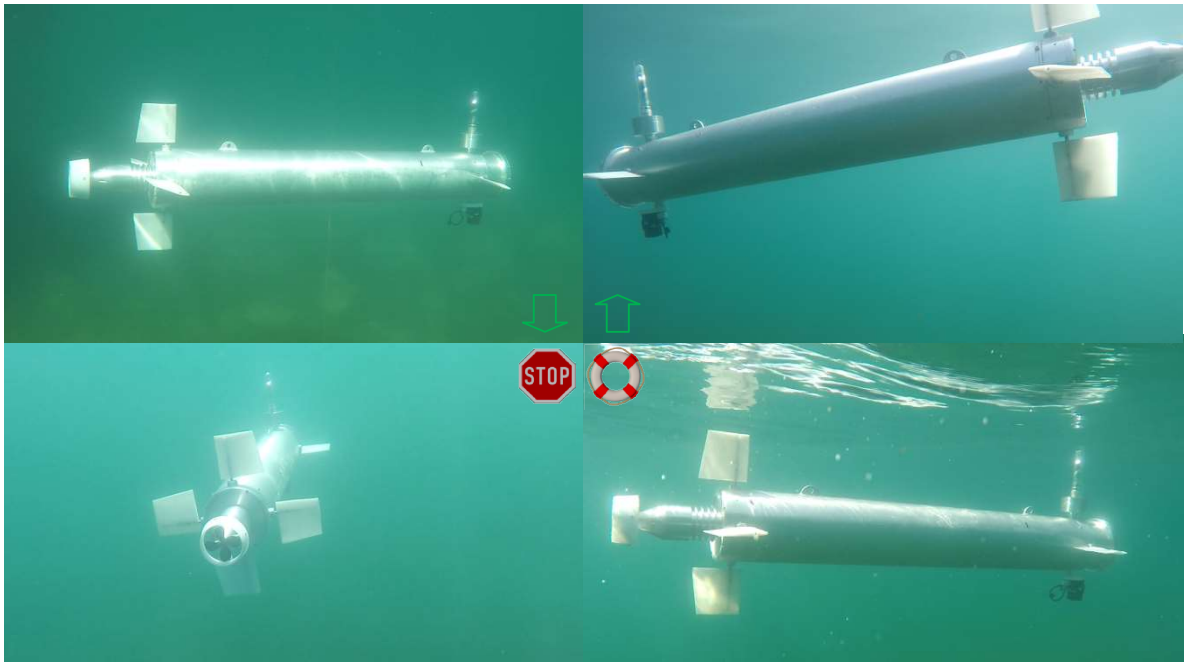


Figura 7.3a – Istantanee della sequenza denominata *DISCESA [DRITTO]* (4 tratti); in ordine cronologico, dall'alto in basso e da sinistra a destra: esecuzione del 2° (immersione) e del 4° tratto (emersione), fase di arresto alla fine della sequenza (arresto veicolo) e riaffioramento naturale (per spinta idrostatica)

```
#1459781980 - Mon Apr  4 14:59:40 2016
#1/2 DISCESA [DRITTO] - (GOPR9479.MP4)
#----------------------- INIZIO 1a SEQUENZA -------------
#---TIME---|-ang1[°]-|-t1[s]-|-ang2[°]-|-c2[-]-|-t2[s]-
1459781986      0       5       -10     i       5
1459782006      0       20      10      t       20
1459782016      0       10      0       t       10
1459782036      0       20      -10     t       10
```

"*TRAJ.txt*"

```
#1459778628 - Mon Apr  4 14:03:48 2016
#1/2 DISCESA [DRITTO]
#---TIME---|-Dir[°]-
1459781979      102.40
1459781993      108.80
1459782007      117.40
1459782021      106.90
1459782035      113.90
```

"*D_D.txt*"

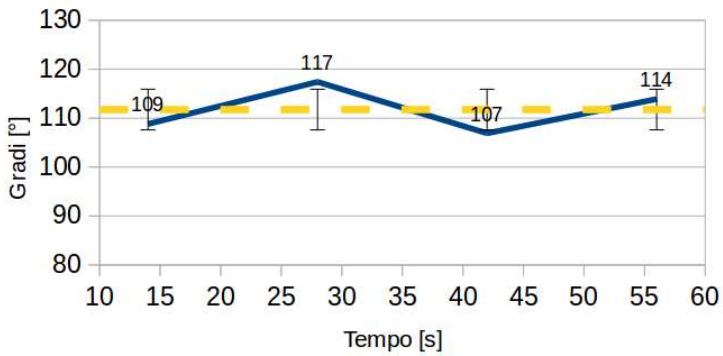


Figura 7.3b – Estratto numerico delle traiettorie impostate in **fig. 7.3a**, delle rilevazione della bussola e del corrispondente grafico della direzione rilevata. Dal grafico è visibile come il veicolo, nel corso della missione così impostata, durante i 55s occorsi per l'esecuzione totale delle tratte, mantenga una traiettoria rettilinea nell'intorno di ±4°

Si osserva che in questa seconda sequenza la durata complessiva di 55s non ha consentito al veicolo di riemergere a *thruster* acceso. Tuttavia questa circostanza ha consentito di verificare l'efficacia della lieve positività (*buoyancy*) a cui si accenna in [**4**], dimostrando il grado di sicurezza del veicolo rispetto all'azione di anti-affondamento. Questo aspetto offre ulteriori garanzie nel caso intervengano gli script caratterizzati da azioni di *halt* o condizioni di eventuale anomalia o blocco del motore.

SECONDA CAMPAGNA

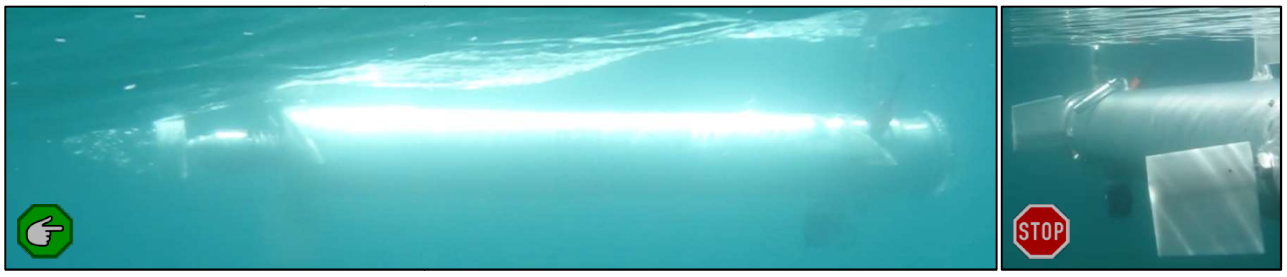
A valle di significative modifiche migliorative sia all'interno che all'esterno del *vessel*, inclusa l'aggiunta del modulo *[IMU]*, si è ripreso a collaudare il sistema al lago, avviando una serie più numerosa di campagne di acquisizione.

La prima campagna di questa serie (**fig. 7.4**) è stata condotta presso la Base Militare di Vigna di Valle. Si è voluto testare una prima versione di controllo di profondità, basata sulla vecchia architettura a strati di **fig. 3.1a**.

La sequenza è stata assegnata impostando una quota di navigazione ad una profondità (*target*) di 2m per 80s e 5s di tratto rettilineo superficiale, sia dopo l'avvio che prima dell'arresto del motore.

Le condizioni iniziali da evidenziare, con veicolo a riposo, sono state le seguenti (*cfr.* **fig. 7.5**):

- PITCH a circa -8°, determinato dallo zavorramento interno a prua, effettuato in Laboratorio, per agevolarne l'immersione ma soprattutto per compensare l'effetto idrodinamico provocato dal cilindretto porta-antenne posto sulla sommità anteriore del veicolo una volta immerso;
- ROLL a circa -4°, determinato dallo zavorramento laterale (verso sinistra) attuato dall'esterno, appendendo un peso all'occhiello anteriore (**fig. 7.4b**) per compensare la resistenza idrodinamica opposta dal connettore di carica delle batterie posto invece sulla destra (**fig. 7.4**);
- YAW (direzione di posa in acqua del *vessel*) a Sud (circa 175°), allo scopo di dirigere il veicolo, a vantaggio di sicurezza, verso la riva del lago, una volta lanciata la missione;
- QUOTA (profondità), registrata a qualche decina di centimetri sotto il pelo libero dell'acqua, legata alla posizione ribassata del profondimetro *[P0]* rispetto alla struttura del veicolo.



(a) ***(b)***

Figura 7.4 – Istantanee a T_{start} e T_{stop} relative al lancio del 28 luglio 2017: ***a)*** è ben visibile la cavitazione provocata dall'elica nel tentativo di immersione con pinne "eccessivamente" inclinate; ***b)*** stop finale – verso la prua è possibile notare il contrappeso esterno

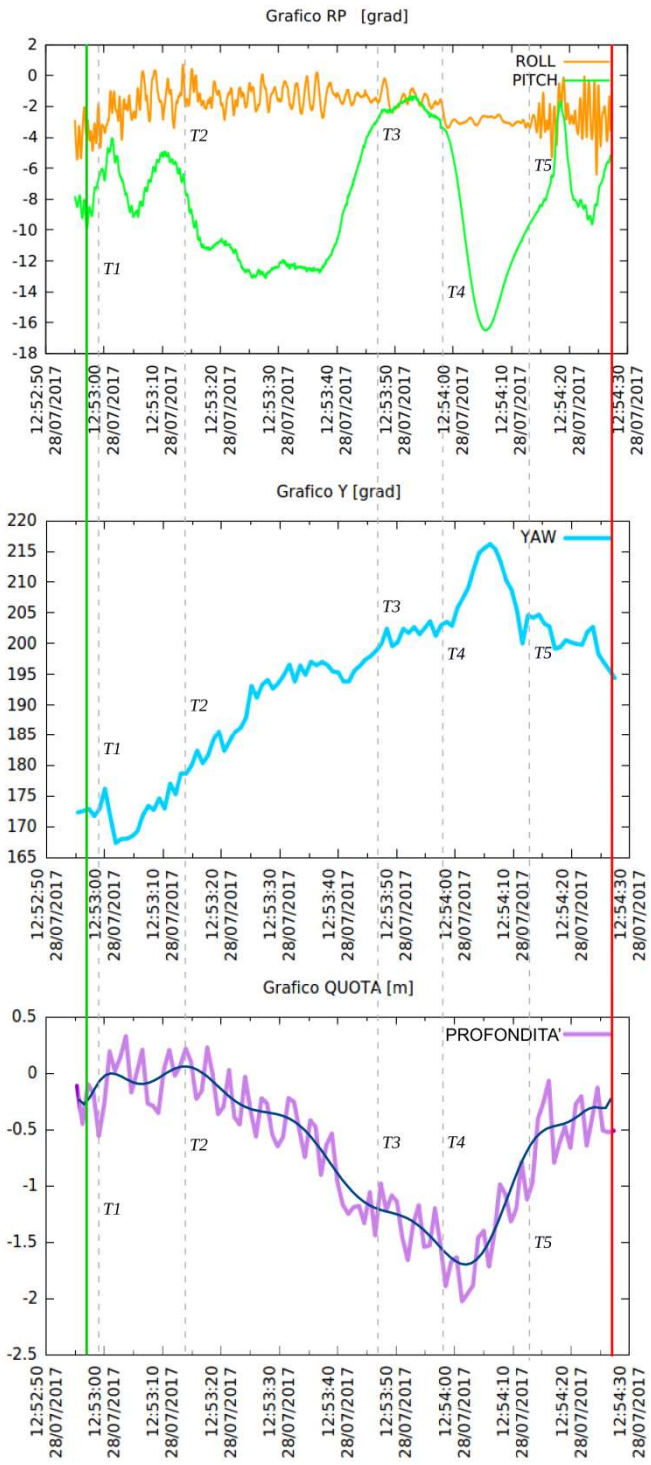


Figura 7.5 – Andamenti, dall'alto al basso:
del rollio/beccheggio (ROLL/PITCH),
dell'imbardata (YAW) e della quota (PROFONDITA')
Registrazione video associata: *MVI_0818.MOV*

*Tstart=**12:52:57** – Tstop=**12:54:27** – **Fig. 7.5***

Nei primi 3s dalla partenza (fino all'istante *T1=**12:53:00***) sull'elica si verifica della *cavitazione* (**fig. 7.4a**) causata dall'accentuato PITCH che poi si riduce assieme al ROLL, semplicemente per il fatto che il veicolo comincia a prendere velocità quando è in completa immersione.

Nei 14s successivi (fino all'istante *T2=**12:53:14***), dopo una pendolazione iniziale di beccheggio (PITCH), il veicolo raggiunge una posizione di equilibrio almeno sul ROLL (-2°) e comincia ad immergersi lentamente, anche se fortemente appruato (PITCH prossimo ai -10°); si noti come nel lasso temporale che va da *T2* a *T3* l'orientamento (YAW) vari leggermente proprio a causa di una idrodinamica non ottimale legata a questo accentuato appruamento ed al leggero rollio, che tende a "tirarlo" verso destra.

Prova ne sono gli 11s successivi (da *T3=**12:53:47*** a *T4=**12:53:58***) in cui il veicolo si posiziona pressoché orizzontale e prosegue tranquillo la sua discesa verso il *set-point* dei 2m di profondità **(**mai realmente raggiunti per la durata impostata nella missione**)**, facendo però registrare una accettabile stabilizzazione dei valori di rollio, beccheggio ed imbardata.

Dall'istante *T4* comincia la fase di riemersione di poppa (non prevista dal tratto impostato), riaccentuando per i 15s successivi l'alterazione dei valori di rollio e beccheggio e falsando palesemente quello di imbardata, che all'istante *T5* ritorna magicamente allo stesso valore a cui si trovava all'istante *T4*, contrariamente da quanto si riscontra nella registrazione video.

All'istante *T5=**12:54:13***, infine, la situazione si normalizza nuovamente, stabilizzando la quota intorno a 0,5m fino ai 14s successivi, ma con delle accentuate pendolazioni sul rollio e con un riaumento del beccheggio rispetto al valore iniziale della missione.

Per completezza, si riporta il grafico che mostra l'inclinazione assunta dalle pinne durante questa missione (**fig. 7.6**). Si osserva che, in coerenza con i dati commentati a partire dalla **fig. 7.5**, man mano che il veicolo si avvicina al *set-point* di profondità (*T3÷T4*) le pinne riducono la propria inclinazione, mentre superato l'istante *T5*, esse aumentano nuovamente l'angolo, nel tentativo di recuperare la quota approcciata agli istanti precedenti.

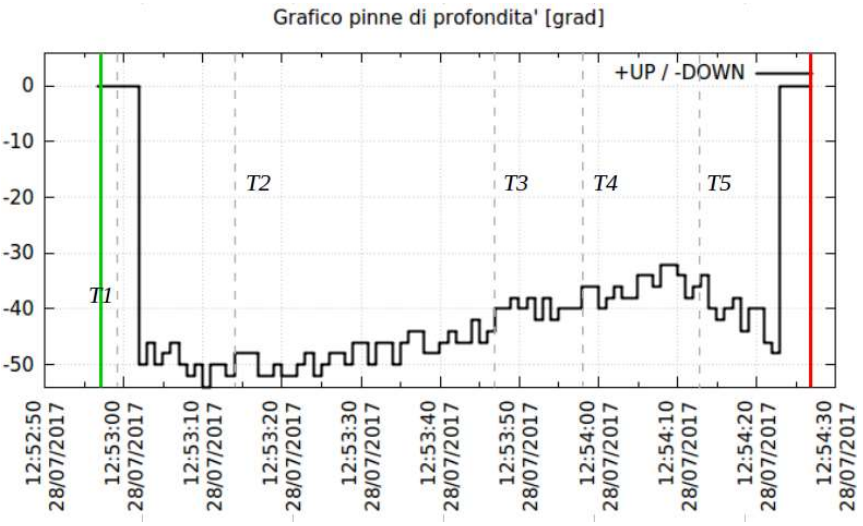


Figura 7.6 – Andamento dell'angolo delle pinne di profondità nel corso della missione illustrata in **fig. 7.5**.
Controllo eseguito con un elevato valore del coefficiente sulla componente proporzionale del regolatore in *PID-press* (***D.10***)

8. Conclusioni

Con le due campagne di acquisizione, i cui risultati sono stati descritti nelle pagine precedenti, si sono condotte una serie di prove mirate sul veicolo che hanno permesso di migliorarne le prestazioni complessive.

A valle delle *prove preliminari* qui illustrate sono stati pianificati vari interventi mirati a risolvere diversi aspetti tecnici ancora suscettibili di studio ed approfondimento. Ad esempio, apportando modifiche all'hardware (motore più potente, sostituzione dell'elica, revisione dei cablaggi interni – per citarne solo alcune) ed al software (uso di piattaforme precostituite più performanti), questo lavoro ha consentito di portare il prototipo VENUS ad una maturazione tecnologica sempre maggiore, sia pure in presenza di molte criticità. In questo modo sono state validate le caratteristiche tecniche, è stato affinato il software di controllo e sono state valutate le sue prestazioni complessive e le reali potenzialità di utilizzo.

Trattandosi di un prototipo sperimentale, a conclusione di questo rapporto, si vogliono evidenziare gli aspetti critici che ancora permangono.

Il primo aspetto da segnalare è legato alle problematiche di *Safety* connesse al suo utilizzo da parte di un operatore esterno: la mancanza di una politica di *obstacle avoidance* [**5**] ne rende al momento impraticabile l'impiego in spazi ristretti o con ostacoli posti lungo il suo tragitto.

Sempre con riferimento alla *Safety*, bisogna osservare che, per i limiti riscontrati sul canale radio WiFi già evidenziati in [**4**], non è possibile prevedere, in condizioni ambientali sfavorevoli, un pilotaggio affidabile del veicolo in teleguida o la gestione delle missioni da lunga distanza (>30m) a causa di "campo debole". Questo problema sarebbe tuttavia risolvibile ripensando l'hardware costituito da tutte le antenne collocate nel veicolo.

Si osserva a tal proposito che, allo stato attuale, l'antenna WiFi (avente *radiation pattern* cilindrico), soprattutto se impiegata in *saturazione di banda* (ad esempio, richiesta streaming video 'live' a veicolo emerso), interferisce con la sovrastante antenna GPS (avente *radiation pattern* semisferico) provocando la cancellazione pressoché totale di ogni informazione proveniente dai satelliti di posizionamento globale.

Il secondo aspetto è legato alle possibilità di governo del veicolo: già in questo lavoro si è tentato di implementare una minima funzione di controllo per obiettivi (*cfr.* ***D.9*** e ***D.10*** – **Fig. 5.9**), sia pure con qualche limitazione per:

- la crescente complessità richiesta nella programmazione della *GUI,* inizialmente concepita per svolgere funzioni essenziali;
- i limiti connessi ad una architettura a strati come quella di **fig. 3.1a**, rivelatasi sempre più complessa. Ciò si è concretizzato in un aumentato numero di file da condividere sull'*OS* ed in un numero sempre maggiore di programmi specifici a bordo, per poter assolvere le crescenti funzioni richieste.

Il software di base si è rivelato affidabile ed ha permesso la messa a punto dei singoli dispositivi di bordo e la conduzione delle *prove preliminari* illustrate in questo documento. L'insieme di programmi e l'interfaccia grafica (*cfr.* **5.*D***) sono stati anche impiegati a scopi dimostrativi in occasione di eventi aperti al pubblico, organizzati presso la piscina del Laboratorio.

Entrambe le limitazioni descritte, inoltre, sono state in gran parte superate dall'adozione del *middleware ROS* (*cfr.* **fig. 3.1b**).

Di fondamentale importanza si sono rivelati i demoni (*cfr.* **5.*B***) soprattutto nel corso delle prove in acque libere, in quanto hanno consentito all'operatore di tenere sempre sotto controllo lo stato generale del veicolo ad ogni riemersione, offrendo al contempo garanzie aggiuntive sulla sicurezza di navigazione durante l'esecuzione delle singole missioni.

Il terzo aspetto è legato alla meccanica: durante ogni campagna svolta, si sono rivelati delicati gli assetti individuati nella distribuzione delle zavorre.

Il quarto aspetto è costituito dalla precisione di misura raggiungibile con i sensori di bordo, con particolare riferimento ai moduli di bussola ed IMU. Entrambi i sensori sono lievemente influenzati dal campo magnetico rotante prodotto dal motore e questo affligge la misura di un valore tuttavia ritenuto accettabile per gli scopi, visti i movimenti lenti e le modeste accelerazioni che caratterizzano il veicolo in navigazione.

L'ulteriore aspetto da porre in evidenza è la perfezionabilità del software, con particolare riferimento ad alcuni programmi appartenenti alle classi dei demoni (*cfr.* ***B.1***÷***B.4***) e dei programmi specifici, *alarm* (*cfr.* ***C.1***) ed *ottico* (*cfr.* ***C.9***), che, più di altri, costituiscono software in continuo aggiornamento e sono tuttora oggetto di studio e di miglioramento. Essi sono sottoposti a continua revisione per far sì che i processi ad essi associati svolgano al meglio la loro funzione ed interagiscano tra loro e con l'*OS* ospitante nel modo più semplice e diretto possibile.

Per tutti gli aspetti riguardanti la descrizione elettro-meccanica e la prosecuzione dei test funzionali del robot si rimanda ai rapporti tecnici [**4**] e [**5**], dove vengono illustrati rispettivamente la struttura dello scafo con i suoi sistemi di bordo e lo sviluppo del Sistema di Controllo. In tali documenti vengono inoltre descritte le *prove sperimentali* di funzionamento in acque libere che mostrano la capacità di VENUS di soddisfare agli obiettivi progettuali.

Bibliografia

[1] 2017, https://wsense.it/

[2] Z. Feng, G. Shang, L. Lian "A low-cost testbed of underwater mobile sensing network", *Proc.* IEEE OCEANS Sydney AU, Maggio 2010, pp. 1–5

[3] R. Zanasi "Dispense di fondamenti di Controlli Automatici", UniMO, 2011

[4] G. Cupertino, P.A. Fichera "VENUS - Veicolo per Navigazione Subacquea e Sorveglianza – Struttura dello Scafo e Sistemi di Bordo – Prove Preliminari"

[5] P.A. Fichera "VENUS – Veicolo per Navigazione Subacquea e Sorveglianza – Sistema di Controllo – Prove Sperimentali", ENEA, RT/2020/9/ENEA

[6] http://gumstix.8.x6.nabble.com/

[7] https://www.ros.org/

[8] https://www.phidgets.com/docs/OS_-_Linux

[9] https://creativecommons.org/licenses/by/2.5/ca/

[10] http://mplayerhq.hu/design7/info.html

[11] https://ffmpeg.org/about.html

[12] https://www.arduino.cc/en/Guide/ArduinoDue

[13] M. Martinelli: "Integrazione di sensori sonar per underwater SLAM" – Tesi di Laurea, Università 'LA SAPIENZA' - ROMA, 2012

[14] https://www.hydraricerche.it/chi-siamo/

ENEA
Servizio Promozione e Comunicazione
www.enea.it

Stampa: Laboratorio Tecnografico ENEA - C.R. Frascati
luglio 2020