POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Specialistica in Ingegneria Aeronautica

Implementazione dell'infrastruttura di una Ground Station per prove di volo di velivoli ultraleggeri

Relatore: prof. Alberto ROLANDO

Tesi di laurea di:
Palamidessi Fabio
matricola 739863

Anno Accademico 2012 / 2013

Sommario

Il presente lavoro di tesi si propone di esporre la definizione dell’architettura, Hardware e Software, di una Ground Station (GS) per un sistema di acquisizione dati per prove di volo comunemente denominato FTI (Flight Test Instrumentation).
La GS è stata sviluppata come parte integrante del sistema MNEMOSINE, già sviluppato dal DSTA del Politecnico di Milano, e che ha la peculiarità di essere orientato all'utilizzo in flight test su velivoli appartenenti alla classe ULM (Ultra Light Machine).
Inizialmente, si è proceduto a verificare quali requisiti fossero richiesti alla GS, per poi procedere allo studio di una architettura generale del Sistema GS, fino a definire le sue due componenti principali: l'unità di calcolo primaria e l'unità di puntamento antenna.
Il primo elemento è preposto all'estrapolazione dei dati dai datagrammi ricevuti dal FTI on-board, alla loro successiva ingegnerizzazione e invio attraverso protocollo TCP/IP ad altri terminali.
La seconda unità, è chiamata a garantire il costante puntamento dell'antenna verso il velivolo, al fine di ottenere un adeguato datalink fra la GS e il sistema FTI on-board, per svolgere questo compito è stato necessario implementare un modulo HW che fosse in grado di rilevare posizione GPS della GS e pressione statica a terra.

Indice

Indice delle Figure

1. Introduzione

1.1 Flight Test

Le normative aeronautiche emesse da EASA o FAA prevedono, per la maggior parte dei velivoli, l'obbligo di certificare l'idoneità al volo dei prototipi di interi velivoli e/o di singole parti.

La procedura di certificazione consta oltre a test statici e verifiche documentali, anche di numerose prove da svolgere in volo, dette “flight test”, per verificare le reali prestazioni, ed i comportamenti del velivolo durante il volo.

Negli ultimi anni si è andata a evidenziare la tendenza a voler svolgere dei “flight test” anche su velivoli che non richiedano la certificazione, innanzitutto per poter fornire all'acquirente la certezza delle qualità della macchina, ma anche per poter raccogliere dati utili per migliorare il velivolo.

Nello svolgimento dei “flight test” vi sono due filosofie per l'immagazzinamento delle informazioni ottenute durante le prove:

- la sola registrazione dei dati attraverso il sistema FTI on-board
- l'affiancamento alla tecnica precedente di un trasferimento dati in tempo reale dal velivolo verso una stazione di terra detta “Ground Station”.

Il secondo metodo permette una immediata verifica del comportamento di massima del velivolo e la possibilità di annullare/modificare prove che non stiano seguendo l'andamento voluto a causa di errori di pilotaggio o di comportamenti non previsti del velivolo, al fine di evitare pericoli per il pilota e/o sprechi di tempo e risorse nel caso in cui sia necessario ripetere più volte la prova.

La Ground Station (GS) ha quindi il compito di ingegnerizzare i dati ricevuti dal collegamento con il velivolo, farne copia e distribuirli in tempo reale ad altri terminali.

Detto ciò si può definire come Ground Station il sistema composto da una o più unità di calcolo, da tutte le periferiche per l'interfaccia con l'utente, con il velivolo e con altri terminali, e nel caso in esame, anche dal meccanismo di puntamento dell'antenna direzionale utilizzata per il data-link.

Da questa breve introduzione, si possono desumere quali considerazioni abbiano portato all'idea di ampliare il sistema di rilevazione dei dati di volo MNEMOSINE, creato per la sola acquisizione e salvataggio dei dati a bordo del velivolo, affiancandogli un sistema Ground Station, la cui definizione e realizzazione sarà l'oggetto della trattazione delle pagine seguenti.

1.2 Requisiti operativi e obbiettivo

La Ground Station (in seguito GS), collaborerà con MNEMOSINE, un sistema FTI per la rilevazione dei dati di volo per velivoli leggeri, realizzato dal Dipartimento di Scienze e Tecnologie Aerospaziali del Politecnico di Milano (DTSA), già operativo, ma tuttora in continua fase di sviluppo.

Ciò detto si necessita di un sistema trasportabile, affidabile, facilmente manutenibile, e che, soprattutto, si presti ad essere modificato in hardware e software per supportare le future modifiche e ampliamenti del FTI MNEMOSINE.

Il sistema attraverso l'unità principale dovrà essere in grado di:

- interfacciarsi con la strumentazione on-board, o di leggere i dati da un file di backup creato dal Nodo CAN_logger del sistema MNEMOSINE,
- ingegnerizzare i dati così ottenuti con curve di calibrazione facilmente modificabili dall'operatore
- trasmettere le informazioni così ottenute via TCP/IP a terminali remoti e/o memorizzarle su Hard disk.

Ed attraverso l'unità di “puntamento d'antenna” e l'HW dedicato di:

- calcolare la posizione GPS del sistema GS
- rilevare la pressione statica a terra
- calcolare il corretto alzo e azimut dell'antenna
- spedire via TCP/IP i dati di posizione GPS e pressione rilevata dalla GS

Completano la GS un controller per motori DC Brushed e l'attuatore tilt-pan, che permetteranno di direzionare sempre in modo adeguato l'antenna, garantendo un costante data-link con il sistema FTI on-board.

Si stima la necessità di analizzare, ingegnerizzare e spedire fra i 1100 e i 1500 dati/secondo, quindi la programmazione dovrà porsi nell'ottica di garantire una adeguata velocità di esecuzione, e data la mole di dati da distribuire si necessita di ottimizzare il flusso di dati attraverso rete.

Nella fase di progettazione dell'unità per il direzionamento dell'antenna vi è da aggiungere l'ulteriore requisito di garantire il puntamento del velivolo con un cono di tolleranza di ±2,5°, per sfruttare al meglio la direttività del ricevitore.

1.3 Architettura e schemi a blocchi

Per assolvere ai compiti richiesti è necessario implementare un sistema complesso, che nelle prime fasi di progettazione può essere schematizzato come da 1.

Tutte le misurazioni saranno trasmesse dal velivolo in radiofrequenza, il segnale verrà captato dall'antenna direzionale della GS che le de-modulerà. I dati così ottenuti verranno classificati secondo specifici criteri e in seguito ingegnerizzati e trasmessi tramite protocollo TCP/IP ad altri terminali e ad eventuali unità di memorizzazione.

Il corpo principale del sistema, l'unità di calcolo primaria, dovrà essere sviluppata per garantire l'accessibilità da terminale remoto, oltre che al consueto accesso locale, ed essere facilmente trasportabile e manutenibile, garantendo contemporaneamente adeguati standard di affidabilità software e hardware.

La seconda unità di calcolo (che potrebbe coincidere con la prima) si occuperà contemporaneamente dell'analisi dei dati GPS del velivolo, ricevuti via TCP/IP, confrontandoli con quelli della posizione dalla GS calcolando azimuth e alzo per il puntamento dell'antenna direzionale verso il velivolo, così da garantire sempre una adeguata ricezione del segnale.

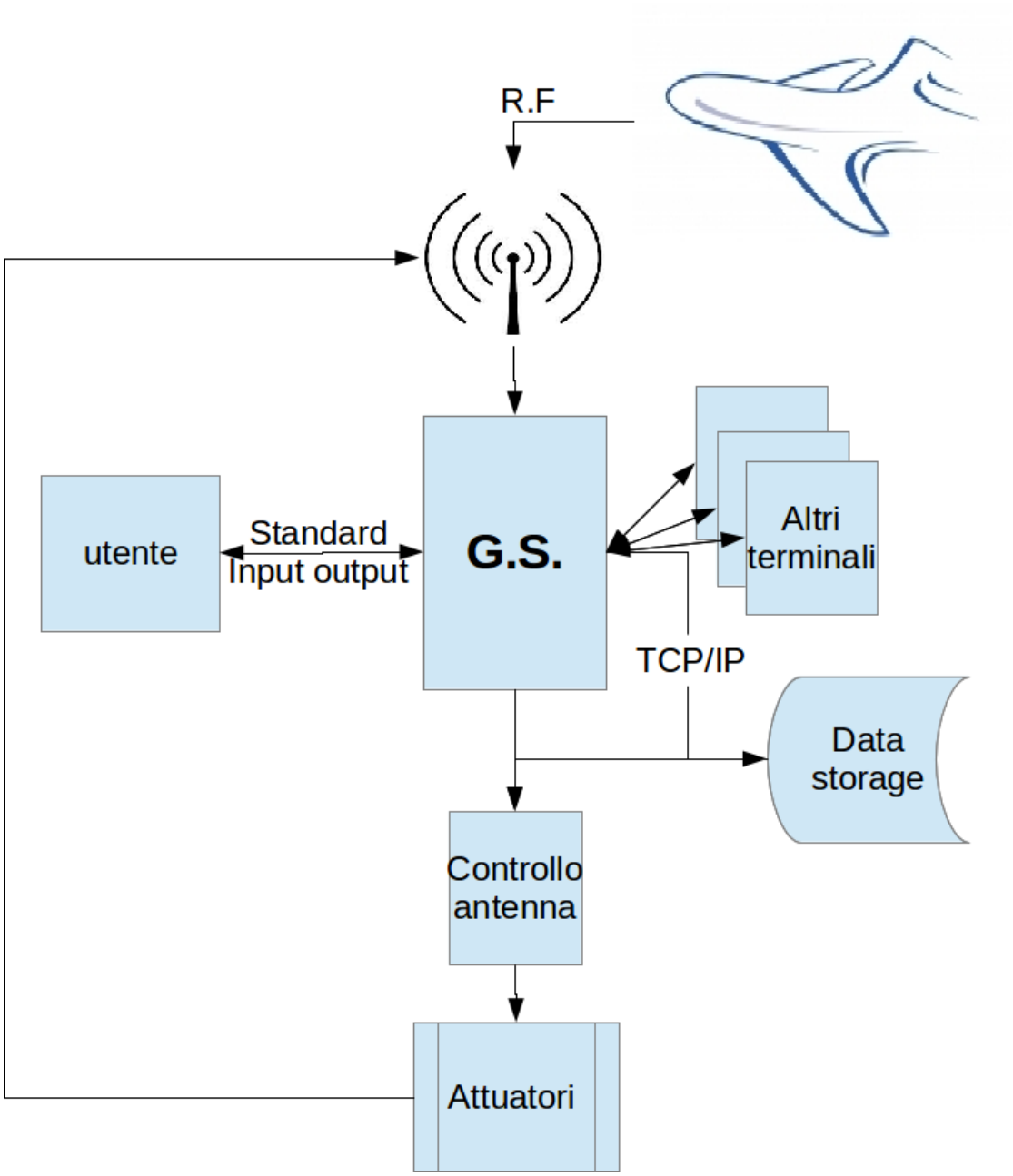

Figura 1: schema sistema Ground Station

2 Preparazione dell'unità principale

I primi passi intrapresi per effettuare il design del sistema sono stati l'identificazione dei requisiti minimi per i singoli componenti, tali da poter soddisfare gli obbiettivi prefissi e garantire contemporaneamente la compatibilità fra i diversi elementi.

Definiti i requisiti minimi come da paragrafo 1.2, si sono valutate le specifiche tecniche dei componenti già disponibili, e si è provveduto alla formulazione di un preventivo per la scelta dei componenti mancanti, necessari allo sviluppo del sistema GS:

- una scheda mini ATX da utilizzare come unità secondaria di calcolo per il puntamento dell'antenna,
- un motor drive con porta seriale e/o CAN che sia in grado di supportare i 2 motori già presenti sul attuatore tilt pan QPT 20,
- uno o più dischi SSD (Solid State Disk) da installare nell'unità di calcolo primaria (o nell'unità di memorizzazione) per garantire una sufficiente velocità e affidabilità nella registrazione / lettura dei dati.

Definiti tutti i componenti utili per la realizzazione del sistema GS si è proceduto a studiare come fosse necessario configurare un ipotetico PC di uso comune, per meglio adattarsi ai requisiti richiesti all'unità principale della GS.

Di seguito verranno presentati brevemente i concetti utilizzati per la definizione dell' architettura di memorizzazione e la scelta del sistema operativo da utilizzare nell'unità principale di calcolo.

2.1 Architetture di memoria

Allo scopo di garantire massime prestazioni all'unità principale di calcolo, per la memorizzazione dei dati si è valutato di utilizzare una architettura complessa per la connessione dei dischi rigidi, così da migliorare velocità e affidabilità nella memorizzazione.

Nelle prime fasi di progetto si è pensato anche di utilizzare dischi SSD (Solid State Disk) cioè degli Hard Disk che non presentino parti mobili, e che utilizzino memoria flash simile a quella delle normali pendrive USB, avvantaggiandosi inoltre del fatto che questo tipo di tecnologia presenti il particolare pregio di essere immune alle vibrazioni e agli shock fino a oltre 500 G rendendo così il sistema molto più affidabile durante i trasporti.

Per massimizzare l'affidabilità e le prestazioni del sistema vengono analizzati due diversi tipi di architetture per la connessione dei dischi nell'unità primaria

2.1.1 Architettura RAID 0 + 1

Questa architettura sfrutta il seguente schema:

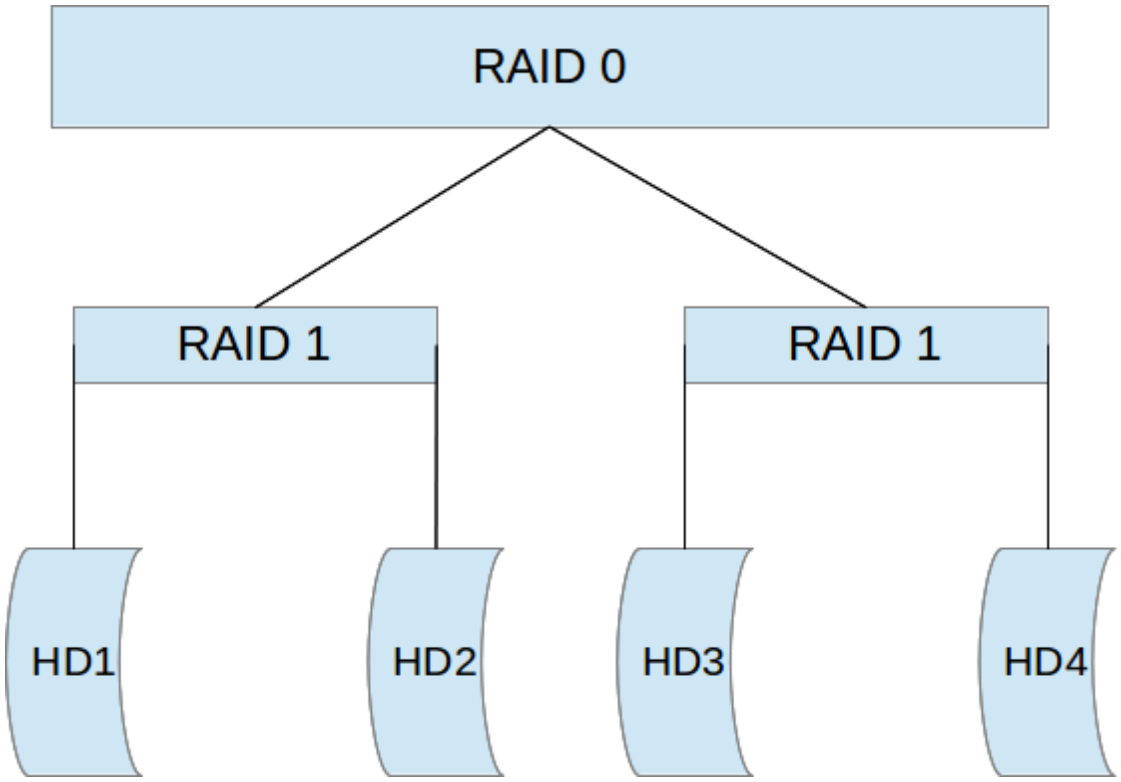

Figura 2: RAID 0 + 1

Si tratta di una configurazione che permette di ottenere ridondanza dei dati attraverso il metodo Raid1 (Mirroring) coniugato alla maggior velocità di accesso / memorizzazione ottenibile dal Raid 0. (Striping).

Questa architettura quindi, risulta “fault tollerant” per la distruzione di un disco e ha il 25% di possibilità di sopravvivere al malfunzionamento di 2 dischi.

Questo aumento di affidabilità ha il costo di una diminuzione dello spazio di memorizzazione totale del sistema, infatti la capacità dei dischi in RAID 1 è

Capacità totale= capacità del singolo disco con memoria minore

mentre per RAID 0 si ottiene

Capacità totale = somma delle capacità dei dischi

Il nostro sistema finale avrà dunque la capacità di memorizzazione pari alla metà di quella totale dei dischi utilizzati (avendo assunto l'ipotesi di utilizzare dischi della stessa capacità e tempi di scrittura comparabili).

2.1.2 Architettura RAID 5

Si tratta di una architettura di connessione che si basa su un solo layer (come si può ben vedere da 3.) in cui tutti i dischi svolgono sia il ruolo di immagazzinamento primario dei blocchi di dati sia quello di salvataggio nella memoria dedicata (detta stripe) delle informazioni utili per la ricostruzione dei dati che ipoteticamente possano essere persi a causa della rottura di un disco.

Qualora un disco (o una parte di esso) divenisse inutilizzabile il sistema avvertirebbe solo un degradamento della velocità di scrittura e lettura.

In caso di rottura di due dischi, a differenza del RAID 1+0, il recupero dati risulta assolutamente impossibile, ma, vi è una maggiore capacità di memorizzazione totale rispetto all'architettura RAID 1+0, infatti si ottiene che

Capacità totale = capacità disco singolo (numero dischi -1)

Svantaggi del Raid 5 sono la necessità di calcolare i blocchi di parità e procedere alla loro memorizzazione rallentando il sistema, ma in caso di utilizzo di Solid State Disk e di un hardware con firmware dedicato i tempi di scrittura risultano esigui quindi questa architettura viene considerata preferibile al RAID 1+0.

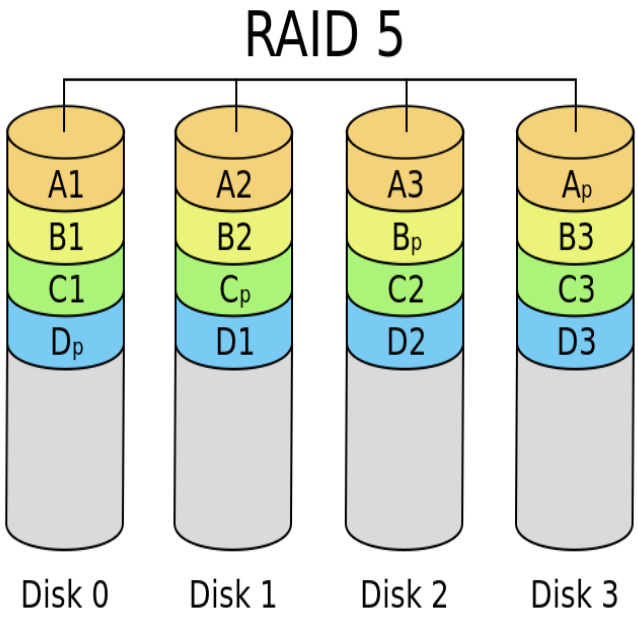

Figura 3: RAID 5

2.2 Studio fattibilità per la conversione di una distro Linux per supporto dischi SSD

Si è deciso di implementare tutte le applicazioni in ambiente Linux, soprattutto per la disponibilità di codice sorgente di pubblico dominio e l'esistenza di grandi comunità di sviluppo in grado di fornire una illimitata letteratura su molti aspetti della programmazione in tale Sistema Operativo (SO / OS); il che permette una vasta possibilità di sviluppo, quasi priva di limitazioni.

Fra le molteplici distribuzioni disponibili si è concentrata l'attenzione su quelle definite Lightweight, ovvero quei S.O. che utilizzando degli ambienti grafici e servizi accessori minimali permettono ottime prestazioni a scapito di una interfaccia grafica spartana.

A tale proposito si sono analizzate principalmente le distirbuzioni con ambiente grafico LXDE e XFCE a 64 bit che garantiscono il funzionamento con soli 128 MB di RAMe processore a core singolo ma che possono supportare anche processori multicore di ultima generazione con architettura a 64 bit e non hanno limitazione sulla dimensione massima di RAMe HD tipica dei sistemi a 32 bit.

Infine è stata vagliata anche la possibilità di escludere completamente l'utilizzo di una interfaccia grafica usando una distribuzione Debian, e utilizzando un accesso remoto via Secure SHell (SSH) [1].

Si sono già introdotti i pregi dei dischi con tecnologia SSD che garantiscono un tempo esiguo per l'accesso ai dati, circa 1/50 di quello necessario ai normali Hard Disk, una fault-ratio inferiore al 1%, rispetto al 8-9% dei comuni HD a metà della vita utile dichiarata, e una tolleranza agli shock che spesso supera i 500G come Maximum Rating.

Il difetto di questa tecnologia è la limitata possibilità di riscrittura di un singolo blocco di memoria che dai produttori viene garantita dalle 100 000 alle 800 000 riscritture, ma test indipendenti indicano un numero utile di scritture dimezzato rispetto alle previsioni fornite dal costruttore 4.

La soglia di un migliaio TB come quantità massima di dati scrivibili su un singolo device sembrerebbe comunque un parametro di affidabilità più che accettabile, ma si preferisce approfondire il problema considerando l'effetto delle continue scritture effettuate per le necessità sistema operativo.

Per verificare quali siano i reali accessi in scrittura effettuati dal sistema operativo, si è voluto effettuare una prova empirica, installando una delle distro candidate su un terminale dotato di HD “classico” e verificando tramite il programma iotop il numero di scritture avvenute in 5 ore di inattività. Va notato che il numero di scritture su HD dipende dal kernel e dalle impostazioni del sistema operativo ed è pressoché indipendente dal tipo di ambiente grafico installato, a patto che quest'ultimo sia abbastanza “leggero” da consentire l'utilizzo esclusivo della RAM ovvero senza ricorrere alla memoria Swap.

I dati ottenuti pur non essendo affatto rigorosi, danno una stima di quanto sia l'ammontare degli accessi in scrittura.

In caso di sistema a riposo è possibile verificare come la maggioranza degli accessi sia dovuta al processo di Juornaling (jbd2) e del server SQL preinstallato nella distro e del log di SQL (rsyslog).

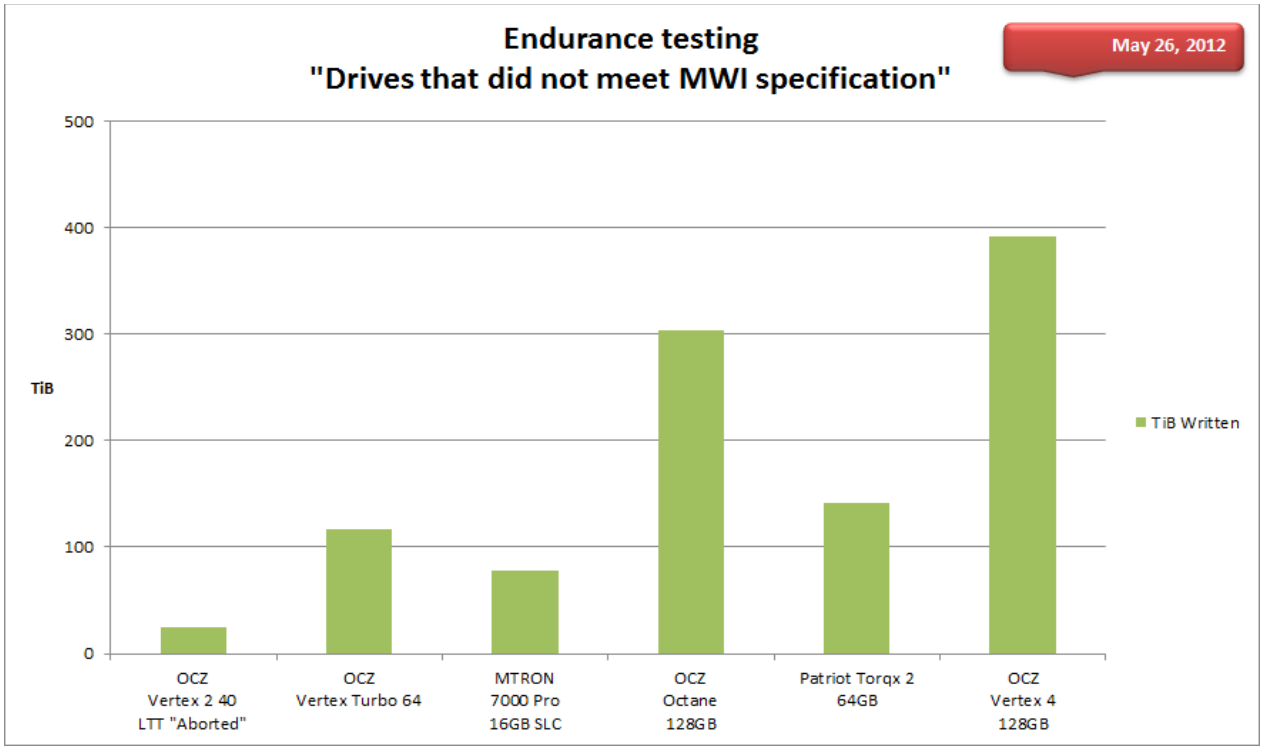

Figura 4: Test indipendente sul numero massimo di scritture su SSD il dichiarato per questi dischi era fra 800 e 1000 TB da www.ssdaddict.com

Al fine di limitare il numero di scritture senza dover intervenire con la modifica del kernel si sono vagliate le seguenti ipotesi:

- disinstallare il server sql non necessario alle nostre esigenze
- disabilitare il journaling della partizione ovvero usando una formattazione Ext4 senza journaling,
- disabilitare l'opzione “atime” del journaling

Fra gli ultimi due punti si è scelto la strada di disabilitare l'opzione di scrittura del time stamp di accesso ai file (detta appunto atime) piuttosto che disabilitare l'intero journaling, infatti in caso di crash del sistema o di caduta di tensione il journaling garantisce che il file system resti coerente e che la perdita di dati sia minimizzata coinvolgendo solo le informazioni in fase di memorizzazione.

Si ritiene utile far notare che un sistema con dischi SSD deve essere assolutamente fornito di una quantità di RAM maggiore di un terminale dotato di HD tradizionali, in quanto l'utilizzo della memoria Swap va limitato a poche situazioni di emergenza al fine di evitare un rapido degrado del disco.

Ultimo accorgimento previsto per l'utilizzo dei dischi di tipo SSD è la non formattazione (dunque la non scrivibilità) di circa il 10%-15% dello spazio totale di memoria per dare la possibilità al controller interno del disco SSD, di utilizzare questa zona per inserire i dati destinati a settori usurati, senza inficiare la velocità di lettura scrittura e l'integrità delle informazioni.

3 Software dell'Unità Principale

Il sistema è stato pensato come composto da più elementi connessi fra loro con diversi tipi di bus.

L' unità principale è stata concepita per essere l'elemento computazionalmente più prestante che ha lo scopo di:

- decodificare i messaggi in arrivo dal velivolo (i dati sono codificati attraverso il protocollo Caffe)
- ingegnerizzare i dati
- classificare e stoccare i dati in strutture
- verificare il sopraggiungere delle condizioni per l'invio delle strutture dati alle utenze ed eventualmente provvedere all'inoltro.

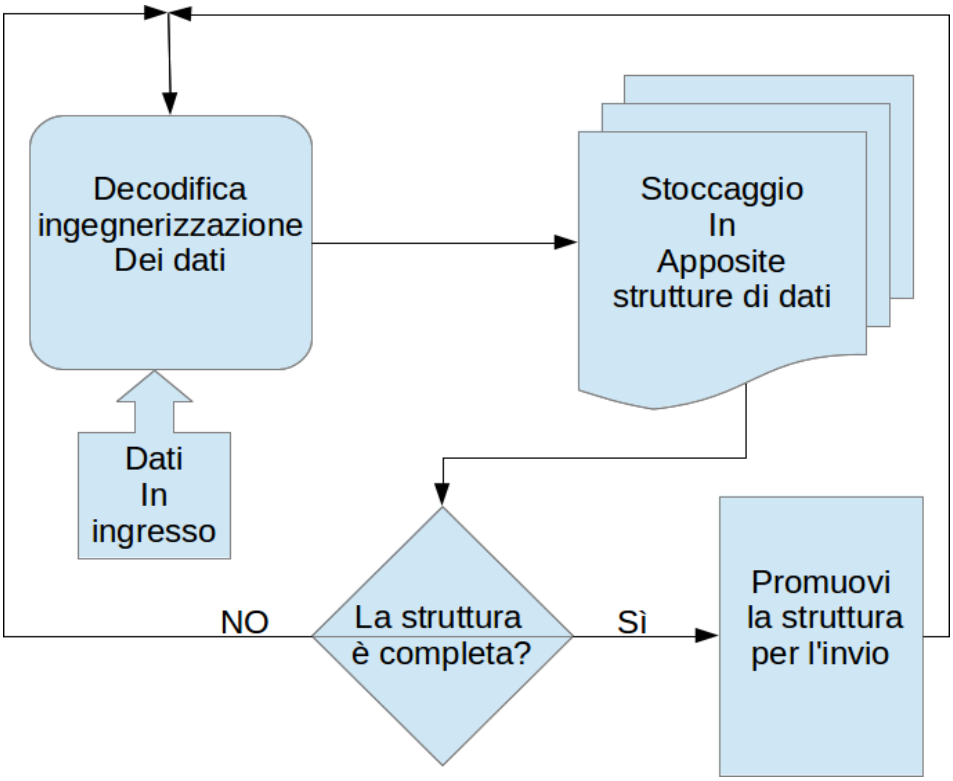

Figura 5: schema indicativo delle funzioni svolte dal Caffe_server

Il linguaggio di programmazione prescelto per l'implementazione è l'ANSI C.

Lo sviluppo del software per l'unità di calcolo principale richiede la conoscenza del sistema di registrazione dei dati di volo e di tutti i protocolli che intervengono nelle varie fasi, perciò nei prossimi paragrafi si darà una breve rassegna di alcuni dei concetti fondamentali utilizzati per l'implementazione di CaffeNet_server, ovvero il programma che verrà implementato sull'unità principale di calcolo.

3.1 MNEMOSINE e il protocollo Caffe

MNEMOSINE è il sistema di raccolta dati per prove in volo sviluppato dal DSTA del Politecnico di Milano al fine di eseguire Flight test su velivoli ultraleggeri. Il sistema di acquisizione dati installato sul velivolo consta attualmente di 7 nodi collegati da un Can-bus ed è in grado di registrare: dati inerziali, posizione delle superfici di comando, forza esercitata sui comandi dal pilota, posizione e velocità GPS, dati aria e dati motore, con una frequenza di campionamento che varia da parametro a parametro fra i 2 e i 60 Hz come illustrato in Tabella 3.1.

ID	Nome Parametro	NODO	Frequenza	
1	X_acceleration	Terpsicore	60	Hz
2	Y_acceleration	Terpsicore	60	Hz
3	Z_acceleration	Terpsicore	60	Hz
4	Pitch_angle	Terpsicore	60	Hz
5	Roll_angle	Terpsicore	60	Hz
6	Yaw_angle	Terpsicore	60	Hz
7	Pitch_rate	Terpsicore	60	Hz
8	Roll_rate	Terpsicore	60	Hz
9	Yaw_rate	Terpsicore	60	Hz
10	Elevator_control_position	Eutherpe	10	Hz
11	Aileron_control_position	Eutherpe	10	Hz
12	Left_pedal_position	Eutherpe	10	Hz
13	Flap_position	Eutherpe	10	Hz
14	Rudder_force	Calliope	10	Hz
15	Aileron_force	Calliope	10	Hz
16	GPS_time_of week	Polimnia	4	Hz
17	GPS_week_no.	Polimnia	4	Hz
18	GPS_X_position (ECEF)	Polimnia	4	Hz
19	GPS_Y_Position (ECEF)	Polimnia	4	Hz
20	GPS_Z_Position (ECEF)	Polimnia	4	Hz
21	GPS_X_velocity (ECEF)	Polimnia	4	Hz
22	GPS_Y_velocity (ECEF)	Polimnia	4	Hz
23	GPS_Z_velocity (ECEF)	Polimnia	4	Hz
24	GPS_fix_quality	Polimnia	4	Hz
25	GPS_tracked_satellites	Polimnia	4	Hz
26	GPS_dilution_Of precision	Polimnia	4	Hz
27	Propeller_rpm	Talia	2	Hz
28	Total_pressure	Urania	10	Hz
29	Static_pressure	Urania	10	Hz
30	Angle_of attack	Urania	10	Hz
31	Angle_of sideslip	Urania	10	Hz

Tabella 3.1: Parametri Mnemosine e frequenza di aggiornamento

La modularità del sistema e l'attuale basso carico di utilizzo del CAN-bus, prospetta la possibilità di un forte ampliamento del numero di nodi utilizzati nel FTI e conseguentemente la capacità di aumentare notevolmente la quantità di parametri che sarà possibile registrare durante le prove.

I nodi che compongono il sistema MNEMOSINE a bordo del velivolo, utilizzano per la comunicazione un particolare protocollo proprietario denominato Caffe, che si basa su tecnologia CAN per l'invio di dati codificati in pacchetti composti da 40 bit di header e fino a 64 bit di payload.

- L' **header** contiene informazioni utili per il tracciamento del pacchetto, viene per comodità incapsulato in una struttura dati così formata

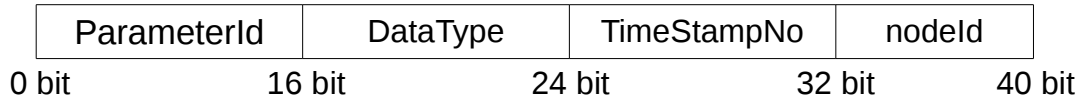

Figura 6: Header del pacchetto Can

- *ParameterId* è l'identificativo numerico che identifica quale parametro sia contenuto nel payload
- D*ataType* specifica come sia formattato il dato contenuto nel payload
- *TimeStamp* è un identificativo del decimillisecondo in cui viene spedito il messaggio
- *nodeId* è l'identificativo del nodo che ha inviato il messaggio

- Il **payload** di 64 bit contiene il dato o i dati formattati come indicato dal parametro *dataType presente nell'header.* Il payload è dunque l'informazione o le informazioni trasportate dal pacchetto.

I principali formati di dati utilizzati come payload sono presentati in Tabella 3.2

Formato	Byte	Range
Double	64	$-2^{52}x10^{1023} < x < 2^{52}x10^{1023}$
Float	32	$-2^{23}x10^{8} < x < 2^{23}x10^{8}$
uint8	8	$0 < x < 2^{8}$
uint16	16	$0 < x < 2^{16}$
uint32	32	$0 < x < 2^{32}$
uint64	64	$0 < x < 2^{64}$

Tabella 3.2: tipi di dati previsti come payload nel protocollo Caffe

3.2 CAN Controller Area Network

Il Controller Area Network (CAN) è un protocollo di comunicazione di tipo seriale, introdotto negli anni ottanta dalla Robert Bosch GmbH, in grado di gestire sistemi real-time, con un elevato livello di integrità dei dati trasmessi ovvero progettato per funzionare senza problemi anche in ambienti fortemente disturbati. Una rete CAN nella sua versione più tradizionale è composta da un bus lineare realizzato con una coppia intrecciata di fili (schermati o meno) e da un numero teoricamente infinito di interfacce (nodi) collegate fra loro attraverso questo mezzo trasmissivo. Nella sua versione più comune si tratta di un sistema di trasmissione con tempo non deterministico, ma nel nostro caso viene implementato un timestamp che permette la corretta ricostruzione della precisa sequenza temporale dei messaggi inviati.

Il bus non viene occupato secondo uno schema predefinito bensì tutti i componenti collegati hanno gli stessi diritti, ascoltano in "parallelo" tutti i messaggi (multicast), e sono sempre pronti a ricevere e, quando necessario, cominciano a trasmettere se nessun altro componente sta inviando pacchetti sul bus; questa è la comunicazione event-driven, ossia la trasmissione viene inizializzata solo quando essenziale.

Se due moduli iniziassero la trasmissione nello stesso momento, questi rileveranno una "collisione" e si comporteranno secondo un insieme di regole definite Collision Detection, che identificano nodi recessivi e dominanti.

Il protocollo CAN per decidere le priorità di acceso al bus adotta un metodo di arbitraggio a singolo bit non distruttivo, ovvero il messaggio inviato dal nodo dominante in caso di collisione non viene alterato e quindi non è necessario procedere alla ritrasmissione.

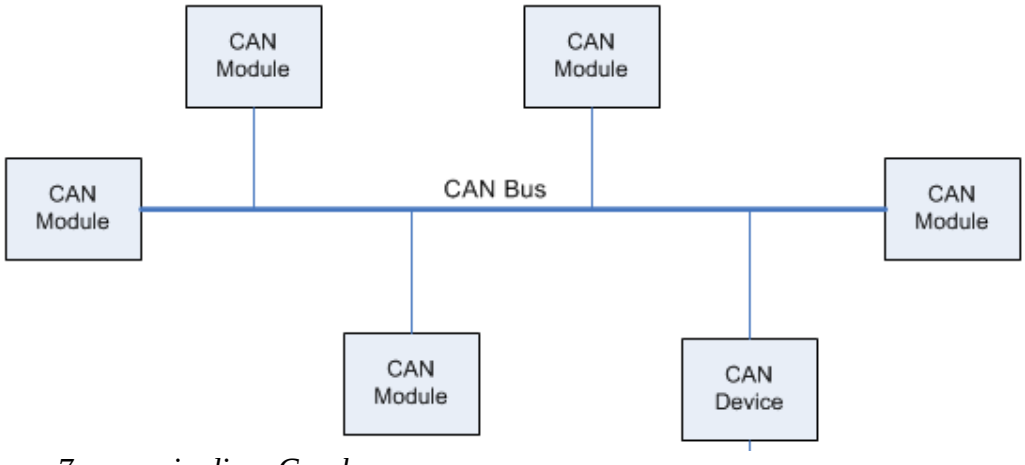

Figura 7: esempio di un Can-bus

3.3 UDP TCP IP User Datagram Protocol e Transmission Control Protocol (TCP) e Internet Protocol

Transmission Control Protocol (TCP), User Data Protocol (UDP) e l'Internet Protocol (IP) sono i componenti principali della Suite di protocolli Internet.

Tale suite è descrivibile dal modello OSI [2], che definisce l'intero processo di invio ricezione dati come una pila di protocolli. Nella pila di protocolli OSI ogni livello svolge una serie di azioni che riguardano la trasmissione di dati e fornisce un ben definito servizio ai livelli più alti. I livelli più alti sono logicamente più vicini all'utente e funzionano con dati più astratti lasciando ai livelli più bassi il compito di tradurre i dati in forme più “basse” mediante le quali possono essere fisicamente manipolati bit a bit e trasmessi infine sul canale di comunicazione.

La suite TCP/IP è stato prodotto come una soluzione ad un ben preciso problema ingegneristico \ pratico, cioè ottenere una connessione efficiente ed affidabile per lo scambio di dati fra terminali, e pur essendo raffrontabile con il modello OSI, si differenzia da esso per una minor connotazione teorico-deduttiva, accorpando alcuni livelli OSI in un singolo layer del protocollo TCP/IP.

Come si nota da 8 a differenza del modello OSI nel protocollo TCP/IP le funzioni sono condensate in soli quattro livelli, al posto dei 7 del modello OSI, partendo dall'alto, cioè dal layer più vicino all'interfaccia utente:

- Livello **applicazione:** raggruppa i livelli di: applicazione, presentazione, sessione del protocollo OSI; rappresenta la piattaforma sulla quali si basano le applicazioni che hanno necessità di operare sulla rete
- Livello di **trasporto**: ha corrispondenza diretta con l'omonimo layer del modello OSI, fornisce una comunicazione da un capo all'altro tra due host, utilizzando degli identificativi numerici (a 16 bit) detti porte, che permettono il corretto smistamento dei dati in connessioni multiple, è su questo livello che si evidenzia la differenza fra protocollo User Data Protocol e Transmission Control Protocol
- Livello **Internet:** è assimilabile al layer Rete del Modello OSI, qui viene adattato il payload all'Internet Protocol, questo livello si occupa dell'indirizzamento e della suddivisione dei dati da inviare in pacchetti compatibili con il Maximum Trasmission Unit (MTU)
- Livello **Rete**: equivalente al layer data-link e fisico del Modello OSI. Le interfacce di comunicazione dei nodi adiacenti saranno individuate per mezzo di un indirizzo univoco, usualmente denominato MAC address. Il livello fisico, trasmette il messaggio sul canale di comunicazione usualmente sotto forma di segnale elettrici o elettromagnetici, leggasi reti cablate o wireless.

Come già accennato si hanno a disposizione due protocolli per il trasferimento di informazioni: TCP e UDP; questi differiscono molto fra loro per le peculiarità della comunicazione instaurata fra i terminali connessi.

Nelle prossime pagine si presenterà una breve panoramica sulle caratteristiche dei due protocolli che hanno portato alla scelta di UDP piuttosto che TCP per le necessità della sistema GS.

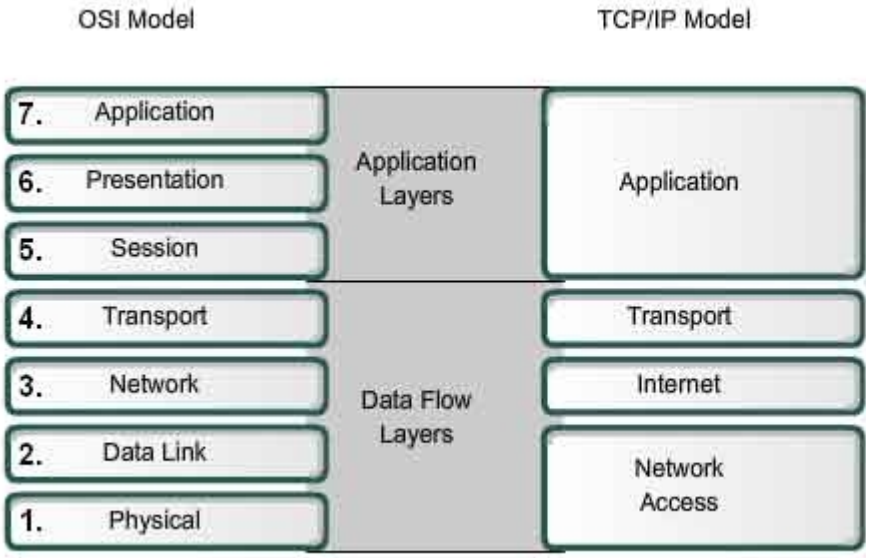

Figura 8: raffronto TCP/IP contro OSI

3.3.1 TCP

Il protocollo TCP si definisce “orientato alla connessione” ovvero necessita che venga negoziata (e accettata) la connessione prima che vengano spediti dei dati, il collegamento rimane aperto, anche in mancanza di un flusso di pacchetti fino a che uno dei due terminali coinvolti non ne richieda la chiusura attraverso l'apposito messaggio di servizio. TCP inoltre, offre un flusso di dati bidirezionale per cui i pacchetti possono viaggiare quasi contemporaneamente verso entrambi i terminali senza soluzione di continuità.

I blocchi di dati nel protocollo TCP vengono suddivisi, solo se necessario, dal livello inferiore (Internet Protocol) per adeguarsi ai limiti imposti dal MTU.

L'unità minima del messaggio nel TCP è detta segmento e il protocollo stesso garantisce che questi siano ricevuti in ordine e ri-invia un pacchetto fino a che non viene ricevuto un segnale di ACKnowledge (ACK) dal terminale ricevente, che certifica l'avvenuta ricezione del blocco di dati, si noti che l'ACK viene inoltrato solo dopo aver verificato l'integrità del segmento confrontando il Checksum calcolato con quello contenuto nel header.

Ciascun segmento viene normalmente incapsulato in un pacchetto IP, che è costituito dall'header TCP e dal payload, ovvero i dati del livello superiore.

Le informazioni contenute nel Header costituiscono i dati essenziali per il funzionamento corretto del canale di comunicazione tra le due entità TCP, e viene anche utilizzato per realizzare le funzionalità dello strato di trasporto

Bit offset	**Bits 0–3**	**4–7**	**8–15**	**16–31**
0	Porta di partenza			Porta destinazione
32	Numero sequenziale del segmento			
64	Numero di Acknowledgment			
96	Data offset	riservato	Flag per utilità TCP	
128	Checksum			Puntatore per dati urgenti
160	Dati opzionali			
160/192+	Data / Payload			

Tabella 3.3: Header TCP

Analizzando per semplicità i pacchetti al livello trasporto, e tralasciando gli incapsulamenti successivi, si nota come, indipendentemente dalla dimensione del payload contenuto un segmento TCP non potrà mai essere di formato inferiore a 160 byte, il che per messaggi composti da un solo float (32 bit) l'header sarebbe oltre il 500% del payload.

3.3.2 UDP

UDP è un protocollo di tipo connectionless, ovvero non necessita di negoziare una connessione fra due terminali per inviare o ricevere i dati ma è sufficiente, che il client si connetta all'ip del server sulla porta adibita alla trasmissione.

UDP non garantisce una rigida sequenzialità dei pacchetti, né la ritrasmissione di quelli persi, ed è perciò generalmente considerato di minore affidabilità. Si tratta di un protocollo che consente di stabilire connessioni con bassa latenza, ovvero trasmissioni veloci, limitando al minimo i pacchetti di sistema spediti e dunque ottimizzando la banda disponibile per l'invio dati; va inoltre notato che anche UDP ha la possibilità di abilitare la funzione di Checksum per la reiezione dei pacchetti corrotti che non verranno rispediti dal mittente ma scartati dal client in attesa del datagram successivo.

+	**Bit 0-15**	**16-31**
0	Porta di partenza	Porta destinazione
32	Lunghezza payload	Checksum opzionale
64 +	Data	

Tabella 3.4: Header UDP

Notiamo che l' header UDP, sempre considerando per semplicità il datagramma al solo livello di trasporto e tralasciando dunque i successivi incapsulamenti, è di soli 32 bit e che, in caso dell'invio come payload di un singolo float (32 bit), l'header sarebbe “solo” equivalente al payload.

3.4 Endianess Big-endian o Little-endian

Con endianess si definisce la differenza fra i due metodi usati dai calcolatori per immagazzinare in memoria dati di dimensione superiore al byte, nel caso di comunicazione fra più terminali o su reti è sempre bene ricordare la possibilità che i diversi elementi adottino una endianess differente.

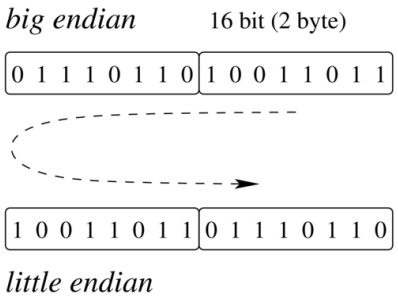

Figura 9: Big endian e little endian

- Con **big endian** si definisce la forma di memorizzazione dei dati digitali, nella quale il dato meno significativo si trova alla fine della sequenza di byte ovvero memorizzare un dato come una sequenza di bytes a partire da quello più significativo finendo con quello meno significativo. Tipicamente adottato dai sistemi Intel
- Con **little endian** si definisce la forma di memorizzazione dei dati digitali, nella quale il dato più significativo si trova alla fine della sequenza di byte ovvero memorizzare un dato come una sequenza di bytes a partire da quello meno significativo finendo con quello più significativo. Tipicamente adottato dai sistemi Motorola.

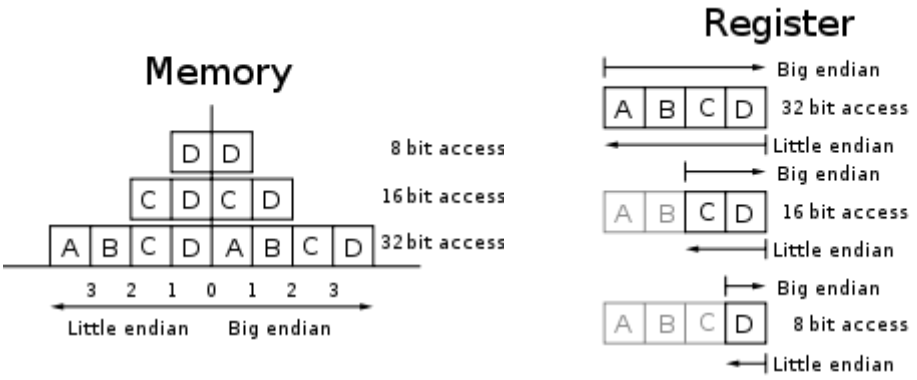

Figura 10: big endian e little endian nella memoria e nei registri

E' importante notare che big-endian, è stato scelto come ordine standard in molti protocolli Internet e di rete, viene perciò anche chiamato *network bytes order*. Per contro viene chiamato *host bytes order* l'ordine nativo dell'host indipendentemente se questo sia little o big endian. Quindi è necessario provvedere una funzione che vada a convertire le endianess dei dati prima e dopo l'invio, in modo da rendere insensibile il processo alle possibili differenze di endianess fra gli host. Queste funzioni sono già sviluppate per i dati più comuni e inserite nella libreria *<arpa/inet.h>* con i nomi htonXY e ntohXY rispettivamente Host to Network e Network to Host dove XY variano per indicare il tipo di dato accettato dalla funzione.

3.5 Sviluppo del software

Nei seguenti paragrafi si svilupperanno i concetti che hanno portato all'elaborazione di Caffenet_server, cioè il programma creato allo scopo di interfacciarsi con il sistema FTI on board durante il volo o di leggere file di backup salvati del sistema MNEMOSINE, ingegnerizzare i dati così ottenuti e trasmetterli via TCP/IP ad altri utenti.

3.6 Necessità strutture dati e loro creazione accorgimenti tecnici usati

I datagrammi in arrivo dal velivolo sono molteplici e non tutti adatti ai fine dell'analisi del volo e dei sistemi. Insieme ai molti dati utili per lo scopo prefissato, vi sono messaggi di “servizio” che hanno per payload informazioni sullo stato del sistema FTI stesso, che risultano inutili per la ricostruzione istante per istante delle condizioni rilevate.

Questo fatto, unito al notevole peso che un header UDP in TCP/IP ha su un piccolo payload, pregiudicano l'ottimizzazione della banda di trasmissione dal server in progettazione, e ai fini pratici un qualsiasi client a valle del sistema GS non è interessato a ricevere i dati alla maggior frequenza possibile, ma è realmente interessato a ottenerli con una latenza tale da garantirne l'utilizzabilità.

Chiarendo il concetto con l'uso di un esempio non esaustivo, è inutile ai fini della ricostruzione delle dinamiche del velivolo, che vengano fornite le misure di pressione totale con una frequenza di 16Hz qualora la pressione statica venisse campionata a 8Hz, in quanto i parametri di volo più utili necessitano di essere calcolati con entrambe le grandezze, e dunque è possibile considerare più che accettabile l'invio delle due misure contemporaneamente, facendo tardare o scartando la misura campionata con frequenza maggiore, considerando così trascurabile la perdita di informazioni dovute a questo rallentamento nel flusso di dati.

Le osservazioni sopra esposte hanno portato alla creazione di strutture che raggruppino efficacemente i dati come segue:

- *Ine_data*

  struttura contenente tutte le informazioni provenienti dalla piattaforma inerziale del velivolo ovvero tutti i dati per la ricostruzione delle forze di inerzia agenti sul velivolo.

- *Sur_data*

  Struttura dati contenente tutte le posizioni delle superfici di comando, rispetto, alle attuali necessità sono stati inseriti anche la posizione degli aerofreni, la posizione del trim e la forza a cui sono soggette le superfici, parametri richiesti dall'agenzia EASA per la Certificazione dei velivoli di maggiori dimensioni

- *Cont_data*

  struttura dati contenente la posizione dei comandi e la forza esercitata dal pilota sugli stessi, anche per questa struttura sono stati inseriti degli ingressi attualmente non in uso, per coprire in avvenire tutte le possibili configurazioni

- *Gps_data*

  Struttura contenente tutti i dati GPS quali: posizione, velocità, prua, nonché i dati sul numero di satelliti in vista.

- `Air_data`

  Struttura contenente i tutti i dati aria raccolti, nonché tutte le misure utili ricavabili da essi

- `ENG_data`

  Struttura contenente i parametri proveniente dal/dai motori, è l'unica struttura non lineare ma annidata per poter ospitare i dati provenienti fino ad un massimo di 4 motori

- `Sys_data`

  struttura contenente tutti i dati provenienti dai sistemi del velivolo, ad esempio, e non esaustivamente, pressione dell'impianto frenante per ciascuna ruota, posizione di ogni carrello, tensione nell'impianto AC e DC, corrente in circolazione nei due impianti, e altri parametri ancora

- `S_g_data`

  struttura contenente i parametri per valutare la qualità della ricezione del segnale GPS

- `S_m_data;`

  struttura contenente tutti i dati sullo stato dei nodi MNEMOSINE

Durante lo sviluppo del codice sorgente si è notato che le impostazioni del compilatore Gcc, non permettevano di avere una perfetta corrispondenza fra la dimensione teorica e reale della struttura, infatti in fase di ottimizzazione il compilatore disponeva i byte delle strutture per ottimizzarne la velocità di accesso in lettura/scrittura.

Questo sarebbe potenzialmente fonte di errori per la ricostruzione delle strutture nei client, dato che questi ultimi potrebbero essere tratti in inganno durante lo spacchettamento della struttura dati per la presenza di “bytes di ottimizzazione” introdotti dal compilatore e non previsti nella variabile strutturata originale. È bene ricordare come la disposizione dei bytes di ottimizzazione sia legata a moltissimi fattori come ad esempio l'architettura del calcolatore, le impostazioni e la versione del compilatore ed è facilmente prevedibile che la posizione di questi nella struttura dati possa variare per compilazioni effettuate su terminali differenti, favorendo così l'insorgere di errori fatali di comunicazione.

Si è individuato nell'attributo packed ([3]) la soluzione a questo problema; questa direttiva forza il mantenimento della disposizione voluta dei byte e di conseguenza la dimensione della struttura originaria, garantendo contemporaneamente minimo uso della memoria.

3.7 Le Librerie come elementi generali riutilizzabili

Nello sviluppo del software si è sempre data molta importanza al principio per cui il sistema GS dovesse essere pensato al fine di favorire futuri ampliamenti e sviluppi, sia della Ground Station stessa che del FTI MNEMOSINE fra i quali, è bene ricordare, esiste un legame simbiotico.

Non sarebbe sensato dover editare e ricompilare il codice sorgente completo tutte le volte che sia necessario abilitare o modificare dei parametri legati ad un singolo nodo di MNEMOSINE, pare dunque una migliore soluzione che tutte le funzioni più comuni collegate a parametri propri del sistema FTI, vengano codificate su librerie diverse dal programma principale, e dove possibile, si utilizzino metodi che permettano modifiche che non richiedano affatto la ricompilazione del codice sorgente.

Si ricorda che un programma vede una libreria come una “black box” alla quale fornire un certo numero di dati con un preciso formato e dalla quale si aspetta di ricevere un certo numero di risultati correttamente formattati.

Questo fa si che il programma principale sia insensibile a variazioni operate sulle librerie ad esso linkate, fintanto che non vengano alterati il numero e il formato dei dati in ingresso e uscita.

Altra particolarità delle librerie è che queste “porzioni di codice oggetto” proprio per la connotazione “black box” possono essere utilizzate da qualsiasi programma, si favorisce così un veloce sviluppo di software con compiti diversi che condividano le sole funzionalità delle librerie incluse.

Come si noterà nella sezione “3.8.6“ l'utilizzo esteso delle librerie ha permesso di creare, in pochi giorni, un nuovo software chiamato Mk Prove volo 2012, veloce, solido, e affidabile con compiti diversi dal sistema GS mentre, se si fosse dovuto procedere alla realizzazione ex-novo del programma, sarebbe stato necessario un lasso di tempo maggiore.

Si esamineranno ora le librerie principali create appositamente per questo progetto.

3.7.1 CaffeNet

La libreria principale e base dello sviluppo dell'intero software, è CaffeNet. Questa contiene la definizione delle strutture dati, degli header e delle union utilizzate per la comunicazione via TCP/IP.

Si è già accennato al fatto che tutti i parametri vengano raggruppati in strutture, per ragioni di ottimizzazione delle prestazioni della rete, ma l'utilizzo di queste variabili composite rende impossibile la ricostruzione dei dati dopo l'invio, a meno di utilizzare particolari tecniche, in quanto il Client non sarebbe in grado di discernere quale tipo di struttura sia quello appena ricevuto, con la conseguente incapacità di interpretazione dei bytes ivi contenuti.

Sovviene, dunque, la necessità di strutturare il pacchetto dati da inviare in due parti fondamentali, un Header (intestazione), e una parte contenente il payload cioè le informazioni utili per studiare lo stato del velivolo.

Il payload contiene la struttura dati di cui si è già disquisito ampiamente, la parte “header” servirà al terminale ricevente per avere tutte le informazioni utili per la decodifica dei dati contenuti nel payload.

L'Header viene sviluppato con la forma di una struttura contenente tutti gli elementi utili per la ricostruzione dei dati lato client, i parametri necessari inseriti sono i seguenti:

HEADER	
type	*parametro*
uint8 t	*version*
uint8_t	*PL_type*
uint32_t	*Tx_seq*
uint32_t	*PL_sex*
char	*Top status*
int	*Top_cnt*
uint8 t	*Flag*

Tabella 3.5: header messaggio CaffeNet

- *uint8_t version;*

rappresenta la versione del software Caffenet, evita possibili errori dovuti all'utilizzo di programmi di diverse versioni non compatibili fra loro.

- *uint8_t PL_type;*

Indica quale tipo di struttura è contenuta nel payload, permette la corretta decodifica dei dati contenuti nel payload

- *uint32_t Tx_seq;*

Numero sequenziale di trasmissione viene incrementato per ogni struttura inviata, permettendo così di identificare eventuali pacchetti dispersi a causa della non garantita affidabilità del protocollo UDP

- `uint32_t PL_seq;`

Numero sequenziale di payload viene incrementato per ogni struttura dello stesso tipo inviata, permettendo così di identificare eventuali pacchetti dispersi precedentemente.

- `uint32_t PL_len;`

contiene la dimensione del payload in Kbyte,

- `char top_status;`

indica lo stato del top_counter, se viene premuto il tasto del Top counter questo parametro viene immediatamente settato a 1 e trasmesso con le prime 3 strutture inviate.

Il tasto Top Counter a disposizione del FTE sul velivolo in fase di test, serve ad annotare una circostanza di volo rilevante, ad esempio l'inizio di una manovra, un comportamento istantaneo ritenuto non idoneo da analizzare nel debreefing post missione, quindi è necessario che la pressione del Top Counter sia notificata il più velocemente possibile.

- `int top_cnt;`

è il contatore che viene incrementato ogni volta che viene premuto il Top Counter, permette la ridondanza per il segnale Top Counter, e di tenere traccia di quale fase di volo sia collegata alla struttura dati allegata

- `uint8_t Flag;`

Un parametro per ora non utilizzato introdotto per futuri sviluppi, permette la trasmissione di 256 diversi messaggi pre-concordati fra server e terminali client, uno per ogni invio, oppure fino a 8 messaggi contemporaneamente.

La struttura attuale del pacchetto dati è rappresentabile dal seguente schema

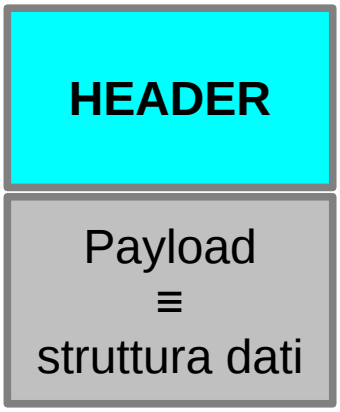

Figura 11 Idea generale struttura pacchetto dati UDP

Questa struttura è una pura astrazione teorica, in quanto, non risulta essere realizzabile praticamente infatti, le strutture dati hanno dimensioni diverse a seconda del tipo e di conseguenza il Client non saprebbe quanti byte debba ricevere, inoltre è necessario concatenare i due elementi da inserire in un unico pacchetto.

Diventa quindi utile includere la struttura dati con header in una union.

Questo premette anche di sfruttare la peculiarità delle union, di poter accogliere uno degli elementi contenuti, e di allocare sempre la stessa quantità di memoria equivalente alla dimensione dell'elemento maggiore.

L'header, fino a questo punto della trattazione si configura come una mera entità senza una reale implementazione, si va ora a sopperire a questa mancanza procedendo a configurare una struttura che ospiti gli elementi dell'header e la union contenente la struttura.

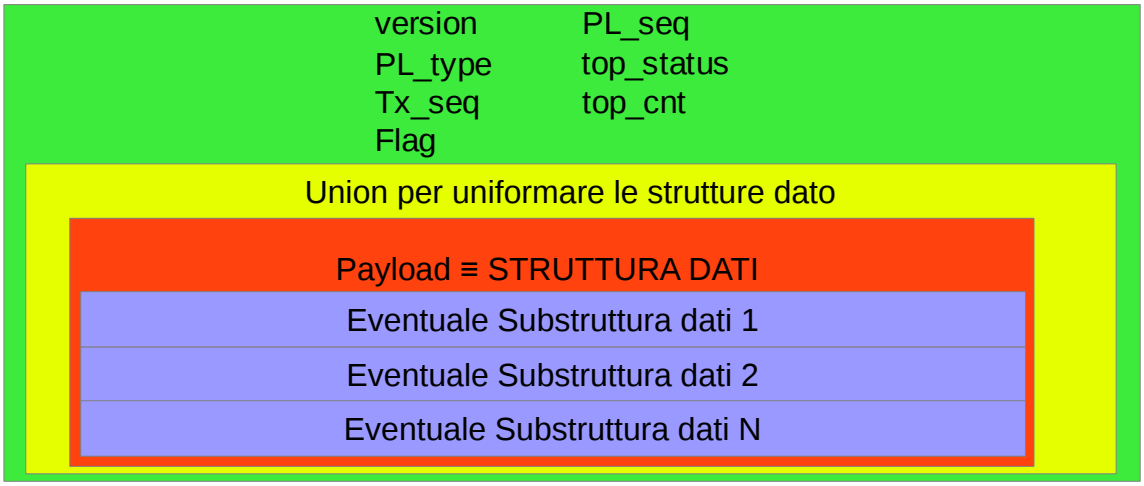

Figura 12: rappresentazione della strutture dell'header

Infine si inserisce quest'ultima struttura in una nuova union, per sfruttare un'ulteriore proprietà di questo tipo di variabili composite, ovvero la confondibilità fra le diverse rappresentazioni dei dati ivi contenuti.

Così facendo l'union contenente header e payload, può essere immediatamente convertita in un array di caratteri della dimensione esattamente coincidente alla suddetta variabile, questo permette di inviarla comodamente con protocollo UDP come vedremo in seguito.

La configurazione finale annidata delle union/struct è mostrata in 13.

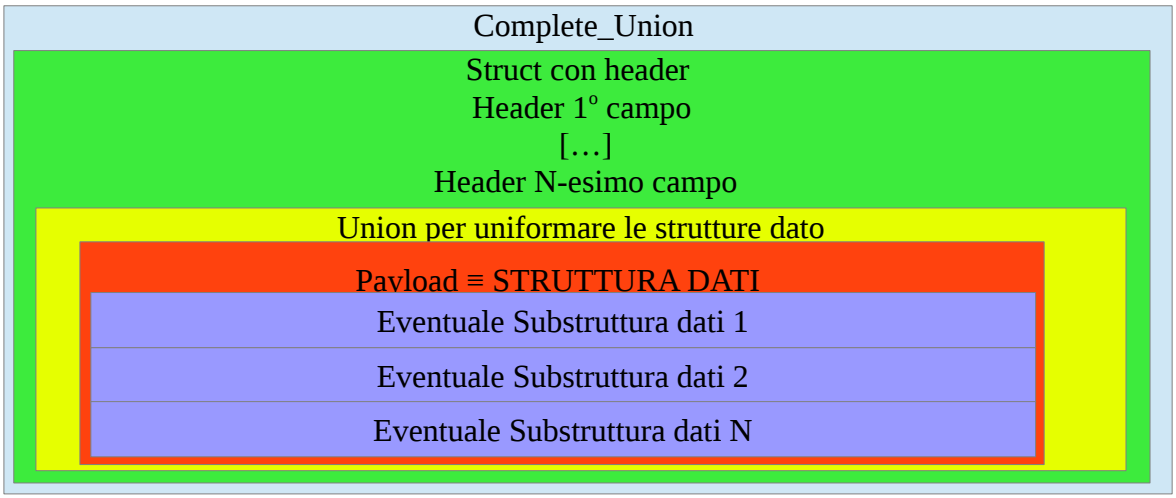

Figura 13: esploso della struttura di un pacchetto UDP CaffeNet

Nella libreria CaffeNet vengono inserite inoltre alcune funzioni:

- `int caffeNet_init (int type_str, t_u_complete* t_str);`

funzione che accetta in ingresso un integrale che specifica il tipo di payload e provvede a pre-inizializzare la struttura puntata da *t_str*, a tutti i parametri float o double viene assegnato il valore SNAN “Signaling Not a Number” [4], mentre si pongono al massimo valore tutti i dati per cui non esistano definizioni di NaN.

La funzione restituisce 0 se correttamente eseguita,

- `int caffeNet_test (t_u_complete * t_str, int m);`

funzione che accetta in ingresso un integrale *int m* che specifica il tipo di payload e provvede a inizializzare la struttura puntata da *t_str*, immettendo dati noti, credibili e costanti per ogni variabile.
Questa funzione è studiata per testare il corretto funzionamento dei programmi e consentirne una facile verifica anche in assenza di una sorgente di dati

La funzione restituisce 0 se correttamente eseguita,

- `int htonCaffenet(t_u_complete *p_union);`

funzione che accetta in ingresso un puntatore a una union **p_union*, provvede a identificare il tipo di payload contenuto, e a convertire l'endianess di tutti i livelli dei dati (header, payload), da host endianess a network endianess. *HtonCaffenet* provvede a richiamare una funzione di conversione diversa per ogni tipo diverso di payload, ognuna di queste, si riconduce alla conversione di ogni singolo dato contenuto nella struttura invocando htonXY nella forma più appropriata. La funzione restituisce 0 se correttamente eseguita.

- `int ntohCaffenet(t_u_complete *p_union);`

funzione che accetta in ingresso un puntatore a una union **p_union* e provvede a identificare il tipo di payload contenuto e a convertire il byte order di tutti i livelli dei dati (header, payload), da network a host endianess.

Anche in questo caso viene richiamata una funzione di conversione diversa per ogni struttura dati; ognuna di queste, si riconduce alla conversione di ogni singolo dato contenuto nella struttura invocando ntohXY nella forma più appropriata. La funzione restituisce 0 se correttamente eseguita.

3.7.2 CaffeEng

Con il nome CaffeEng si è designata la libreria che raggruppa le funzioni per l'ingegnerizzazione dei dati. Come già accennato i dati in arrivo alla Ground Station devono essere spacchettati e prima di essere inseriti nella struttura di appartenenza, si deve provvedere alla conversione dai valori in grandezze con unità di misura del Sistema Internazionale.

Questo implica che sia nota la curva di calibrazione di ogni sensore utilizzato.

Nella fase che ha preceduto la progettazione di questa libreria si è verificato quale fosse il grado delle curve di taratura degli strumenti in uso, ottenendo che queste fossero caratterizzabili da un comportamento al massimo cubico; per il principio di voler agevolare successivi sviluppi si è deciso di sviluppare la libreria in modo da poter implementare curve di taratura fino al 4° grado.

E' impossibile richiedere che i coefficienti di calibrazione vengano inseriti manualmente dall'utente a ogni utilizzo, altrettanto limitante sarebbe la scelta di inserire i valori di taratura nel codice sorgente, infatti, quest'ultima opzione comporterebbe la ricompilazione della libreria a ogni nuova calibrazione.

Si trova una soluzione adatta a tale problema, con l'utilizzo della libreria libconfig [5] la quale permette di leggere, scrivere e manipolare file di configurazione strutturati, questa libreria ha una lettura a codice completamente rientrante, ovvero permette accessi multipli e simultanei al file di configurazione e a tutte le risorse condivise, supporta C e C++.

Un file di configurazione è un documento di testo composto da caratteri ASCII con una rigida formattazione che permette il salvataggio su disco fisso di variabili (semplici o strutturate) e la ri-acquisizione di queste in un momento successivo, anche dopo il riavvio del terminale.

Si procede, all'implementazione di un semplicissimo programma che sfrutti le funzioni di libconfig, per la creazione del file originario di configurazione contenente tutti i parametri utili per l'ingegnerizzazione, inizializzando tutti i valori a dei coefficienti a 0.

Si procede inoltre alla definizione di una struttura chiamata *polinomial* che raggruppi i 118 array di 5 Double che descriveranno la curva di taratura di ogni parametro.

La Libreria CaffeEng consta di sue sole funzioni: la prima chiamata *eng_config* per il “parse” del file di configurazione, ovvero per il processo di lettura e memorizzazione nella struttura polinomial dei coefficienti, la seconda chiamata *pol_eng* utilizzata per la ingegnerizzazione del dato passato in argomento.

- ***Funzione int eng_config(polinomial *pol)***

questa funzione richiede come unico argomento il puntatore alla struttura che memorizzerà i coefficienti dei polinomi di taratura in array di Double, e in sequenza provvede a:

- cercare il file *CAFFE_Calibration.cfg* nella cartella contenente il file in esecuzione e ricorsivamente nelle sottocartelle

- allocare tutta la memoria necessaria per l'operazione di parse

- effettuare il parse di tutti i 5 coefficienti per ognuno dei 118 parametri,

- ***Funzione double pol_eng(double val, int type, polinomial *pol)***

questa funzione restituisce un double contenente il valore della grandezza ingegnerizzata, richiede il valore del counter *val*, il tipo di dato *int type*, e il puntatore alla struttura *polinomial *pol* contenente i coefficienti della curva di calibrazione.

La funzione è strutturata come un switch-case su *type e* una volta identificato il tipo di parametro esegue l'ingegnerizzazione del dato con la formula:

$$gf = \sum_{i=0}^{i=4} strut.polinomial \rightarrow parametro[i] \times val^{i} \qquad \textit{Formula 3.1}$$

Dopo aver introdotto la libreria CaffeEng ci si sofferma nel puntualizzare le potenzialità dei file di configurazione applicati a questo sviluppo:

- Possibilità di ri-taratura a caldo degli strumenti

Il file di configurazione può essere cambiato in ogni momento, ad eccezione l'istante in cui il programma vi accede per caricare i coefficienti attraverso la funzione *eng_config.*

Questo comporta che se la chiamata a tale funzione è effettuata automaticamente, solo all'inizio dell'esecuzione, l'operatore può modificare i valori di taratura nel file *CAFFE_Calibration.cfg* in qualsiasi altro momento. Successivamente, prevedendo un apposito comando in caffeNet_server, è possibile effettuare una nuova chiamata a *eng_config,* per aggiornare i coefficienti, apportando così la modifica senza nessuna soluzione di continuità apprezzabile nel flusso dei dati.

- Possibilità di aggiunte hardware nel sistema

L'utilizzo dei file di configurazione permette inoltre l'aggiunta “a caldo” di uno o più sensori, in quanto, al momento tutti i parametri attualmente non utilizzati, possono essere configurati in pochi attimi inserendo i valori appropriati della curva di calibrazione, o meglio, inserendo una costante unitaria come curva di calibrazione, e dai dati ottenuti calcolare la calibrazione dello strumento per poi aggiornare nuovamente il server forzando l'utilizzo dei coefficienti di calibrazione appena calcolati.

3.7.3 CaffeGPS

CaffeGPS è la libreria del sistema GS che provvede ad implementare tutte le funzioni inerenti al Global Positioning System.

Questa libreria è predisposta per elaborare i dati provenienti da due differenti apparati GPS, uno posto a bordo del velivolo (che chiameremo in seguito GPS1) e uno (che chiameremo GPS2) collocato vicino all'antenna direzionale per la telemetria.

Lo scopo principale di questa libreria è la conversione della posizione del velivolo da Earth-*Centered* Earth-Fixed ECEF a latitudine, longitudine, altezza sull'orizzonte e viceversa, inoltre CaffeGPS provvede al calcolo di alzo e azimuth per intercettare con l'antenna direzionale le trasmissioni del velivolo dati i due valori di GPS1 e GPS2.

The Essential GNSS Project

Per lo sviluppo di CaffeGPS si è studiato approfonditamente il progetto The Essential GNSS, che raggruppa il lavoro di diversi programmatori supervisionato da Glenn D. MacGougan [16].

La libreria Essential_GNSS contiene numerosissime funzioni, molte delle quali inutili per lo sviluppo dell'attuale progetto, ma costituisce sicuramente una adeguata base di partenza per sviluppare Caffe_GPS.

I principali pregi della libreria sono:

utilizzo di dati NMEA GNSS provenienti da porta seriale

analisi di dati per sistemi stand alone o differenziali

dead reconing

analisi dei dati Nmea con filtri a minimi quadrati o Kalman estesi.

Come la maggioranza del software libero questa libreria è fornita con clausola “as is” ovvero senza una vera certificazione dei risultati ottenuti, quindi appare ovvia la necessità di rivedere, validare, o riprogettare tutte le funzioni in modo che vi sia certezza della correttezza dei risultati.

Si è proceduto dunque alla progettazione di Caffe_GPS tenendo come linea guida la libreria Essential GNSS, limitando la complessità delle funzioni al fine di semplificarne la manutenzione e ottimizzare l'utilizzo delle risorse di calcolo.

WGS - 84

La prima scelta di progetto affrontata è stata la designazione dell'ellissoide WGS-84 come modello di riferimento, tale definizione è stata supportata dalla grande disponibilità in letteratura di dati riguardanti tale ellissoide, dalla affidabilità di questo modello su scala mondiale e dalla forte standardizzazione associata a questo modello utilizzato ufficialmente come riferimento per la navigazione aerea, e per le orbite di vari satelliti, compresi quelli formanti la costellazione GPS.

Il CTS 84 è il modello matematico che ricostruisce la Terra da un punto di vista geometrico, geodetico e gravitazionale, costruito sulla base delle misure e delle conoscenze scientifiche e tecnologiche disponibili al 1984.

WGS-84, è l'ellissoide collegato al modello CTS 84, e utilizza un sistema di riferimento (SdR) con origine nel baricentro della terra, asse Z, passante per il polo Nord, asse X, passante per il meridiano di Greenwich, asse Y, scelto in modo da definire una terna destrorsa.

Sistemi di riferimento geodetici utilizzati

I sistemi di riferimento utilizzati nella libreria caffeGPS sono i seguenti:

ECEF

Sistema di riferimento con origine nel centro di massa della terra, Z, diretto come l'asse di rotazione del pianeta, X, passante per il meridiano di Greenwich, Y, scelto in modo da definire una terna destrorsa fra gli assi. Questo sistema di riferimento è considerato inerziale trascurando gli errori derivanti da questa approssimazione.

ECEF è il sistema di riferimento prescelto dallo standard NMEA per il calcolo della posizione GPS.

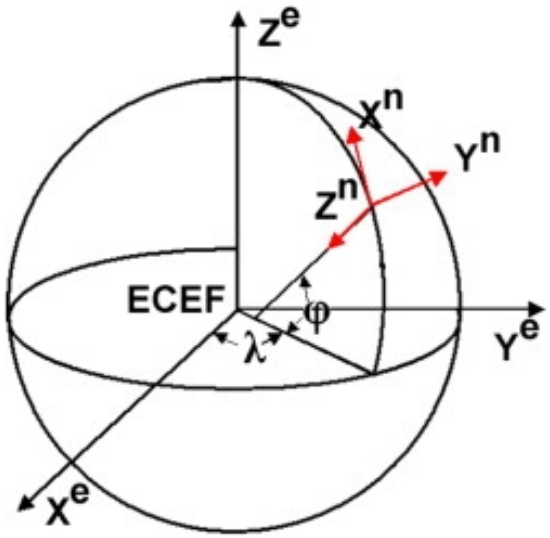

Figura 14: sistemi di riferimento ECEF (Xe,Ye,Ze), LLH (λφh), Nord East Down (Xn,Yn, Zn)

Latitudine longitudine altitudine

E' il sistema di riferimento più classico utilizzato in geodesia, identifica univocamente un punto nello spazio riferendo la posizione al meridiano di Greenwich , all'equatore e all'altezza sul livello del mare.

Il sistema si basa su 2 angoli e una misura lineare diretta perpendicolarmente al piano tangente del geoide.

Gli elementi sono:

Longitudine (λ) angolo, solitamente misurato in gradi sessagesimali, che separa il piano primario generato dal meridiano di riferimento (Greenwich) con quello perpendicolare all'equatore passante per il punto di misura. Può essere designato con valori da 0 a 360° in senso antiorario o da 0 a 180° apponendo la lettera E per misure in senso antiorario o W per misure in senso orario

Latitudine (φ) angolo misurato, solitamente in gradi sessagesimali, sul meridiano passante per la posizione di misura, che separa il piano equatoriale dal punto stesso, Può essere designato con valori da -180° a +180° a seconda che la posizione si trovi sotto o sopra l'equatore oppure può essere apposta la lettera S o N.

Altitudine (h) distanza, solitamente misurata in metri, che congiunge la proiezione della posizione sulla superficie del geoide e il punto di misura, ovviamente secondo la direzione del raggio uscente nel punto di proiezione.

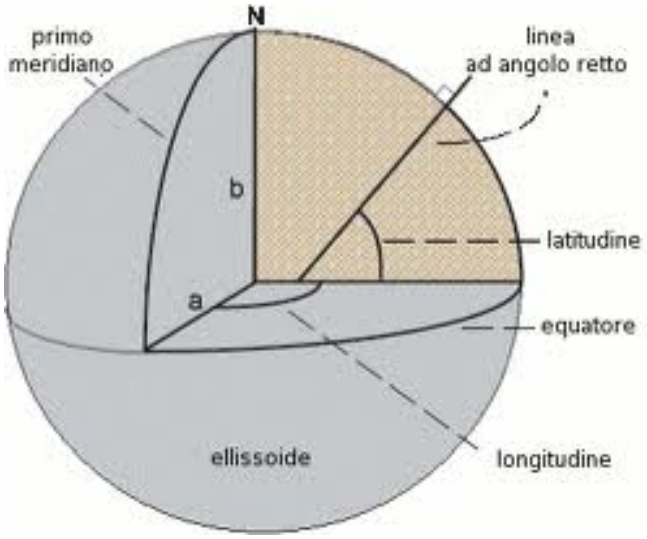

Figura 15: sistema di riferimento Longitudine Latitudine Altitudine

Orizzonte locale

ll sistema di riferimento detto orizzonte locale ha origine O_{NED} nel punto dove si trova l'osservatore con x_{NED} e y_{NED} definiscano il piano tangente al geoide nel punto di origine diretti rispettivamente verso Nord e verso Est, in modo che z_{NED} sia diretto verso il centro di massa del geoide, questo sistema definisce un piano locale sul quale basare il successivo sistema di riferimento

Azimuth, Altezza

Azimuth: scritto a volte anche azimut, angolo tra la proiezione sull'orizzonte locale del punto di misura e il piano di riferimento, cioè il piano definito dalla normale uscente del geoide e la direzione del Nord. La misura viene effettuata in senso orario e assume valori fra 0 e 360°.

Altezza: angolo fra l'orizzonte locale e il punto di misura, assume valori da 0 a 90°, il punto di massima altezza viene definito Zenit.

Questo sistema risulta essere molto intuitivo una volta che l'osservatore a terra posto nell'origine C debba individuare un oggetto in volo conoscendo la sola direzione del Nord Magnetico.

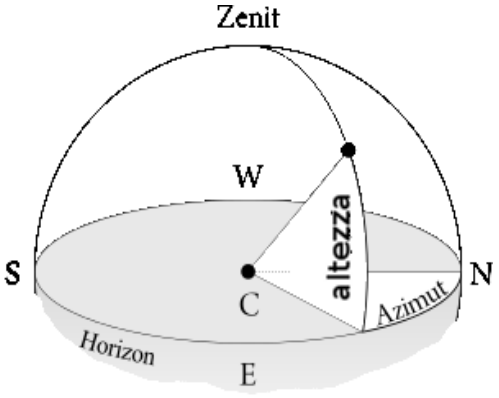

Figura 16: sistema di riferimento Azimuth Altezza

Definizioni e Funzioni implementate in CaffeGPS

Nella libreria CaffeGPS è stato necessario definire un gran numero di costanti al fine di implementare il modello WGS-84, altre sono state inserite allo scopo di memorizzare il valore di π, 2 π, π/2 al massimo delle cifre decimali consentite in un double, per non inficiare la precisione dei calcoli con la continua somma di errori di approssimazione.

double Pi=3.14159265358979238e0;

Inoltre sono state definite le seguenti funzioni:

- ***int Caffe_GPS_ConvertECEFToLatLonH***

  questa funzione accetta ECEF x, y, z come dati in input e da questi calcola i valori di latitudine longitudine e altezza sul livello del mare inserendoli nelle variabili puntate.

- ***int Caffe_GPS_ConvertLatLonHToECEF***

  questa funzione accetta latitudine longitudine e altezza sul livello del mare come dati in input e da questi calcola i valori di ECEF x,y,z inserendoli nelle variabili puntate.

- ***int Caffe_GPS_RotateVectorFromECEFToLocalGeodeticFrame***

  questa funzione calcola le componenti di un vettore in SdR Nord, East, Down partendo da quelle di un vettore ECEF e la posizione di *latitudine e longitudine* di un osservatore a terra.

- ***int Caffe_GPS_ComputeAzimuthAndElevationFromECEF***

  questa funzione calcola azimuth ed elevazione data la posizione dell'osservatore e del punto di misura espresse in sistema di riferimento ECEF.

3.7.4 Libreria CaffeAir

La libreria CaffeAir ha lo scopo di calcolare, dai dati aria rilevati in volo, tutte le grandezze utili.

I dati raccolti sono: la pressione statica e totale; da questi verranno calcolate 13 grandezze correlate, al fine di definire adeguatamente il quadro delle condizioni di volo.

In letteratura sono facilmente reperibili moltissime formule per la determinazione di tutte le grandezze derivate, molte basate su approssimazioni empiriche non adatte alle prestazioni di un velivolo ultraleggero, altre forniscono nel campo di interesse errori grossolani e dunque non sono altresì adeguate allo scopo, mentre alcune formulazioni non sono affiancate da delucidazioni sul campo di validità e applicabilità.

Si sceglie di conseguenza di utilizzare l'approccio proposto da “Introduction to Avionics Systems ” [7]

R.P.G. Collinson limita fortemente le relazioni empiriche, sfruttando invece linearizzazioni di tutti i legami non lineari, e supportando tutte le ipotesi assunte con una estensiva dimostrazione dei limiti di applicazione.

Il diagramma di flusso dei dati proposto con tale approccio è il seguente:

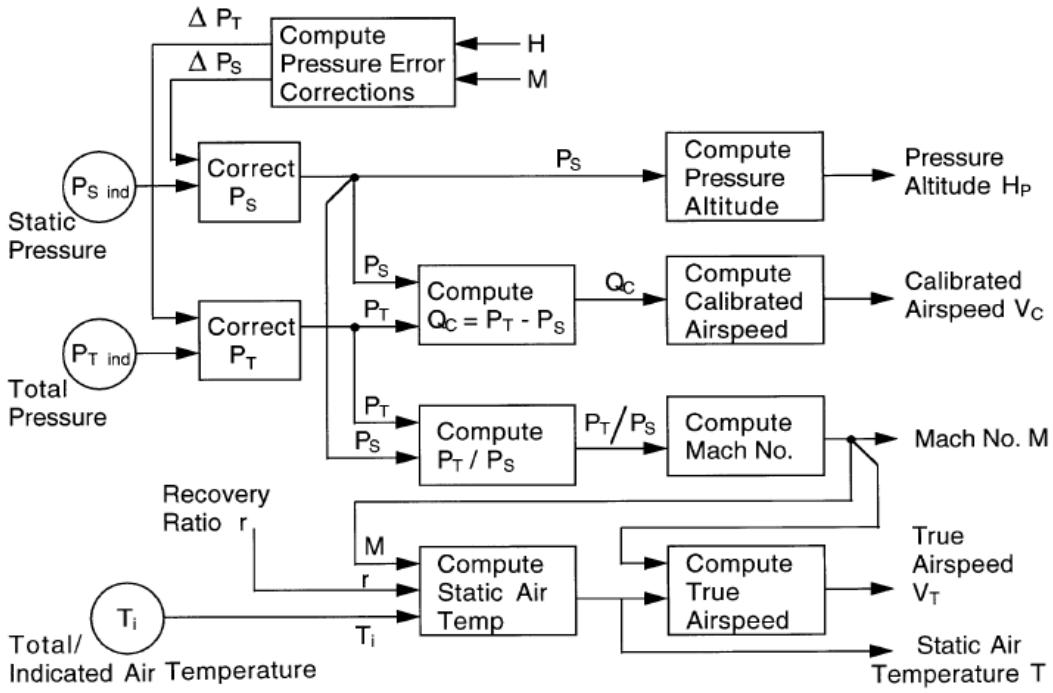

Figura 17: “Introduction to Avionics Systems ” R.P.G. Collinson diagramma di flusso dei dati per il calcolo di tutte le grandezze aerodinamiche utili

Definizioni e Funzioni implementate in CaffeAir

Nella libreria vengono definite come double tutte le costanti necessarie utilizzando la massima precisione consentita dal tipo di dato.

Le funzioni implementate in questa libreria sono:

- ***int Caffe_Air_Altitude(double PS, double * h);***

questa funzione viene utilizzata per il calcolo della quota aria ovvero l'altezza ricavata ponendo come riferimento la pressione standard ISA a 0 m sul livello mare

richiede in input:

double PS [Pa] pressione rilevata dalla presa statica del tubo di Pitot posto sul velivolo

e memorizza il valore calcolato nella variabile puntata da

*double * h* [m] quota aria

per il calcolo della grandezza viene utilizzata la seguente formula

$$h=(TEMP_0/ALPHA)*\left(\left(\frac{PS}{P_0}\right)^{(1/5.255879)}-1\right) \qquad \textit{Formula 3.2}$$

la funzione implementa il limite di validità della formula che sono nel range (0 – 11000 m).

La funzione ritorna 0 in caso di esecuzione corretta, -1 se i dati non fossero compatibili con le ipotesi.

- ***int Caffe_Air_DENS(double h, double * dens);***

questa funzione calcola la densità dell'aria alla quota di volo.

La grandezza viene calcolata utilizzando la densità ISA di quota 0 m ($DENS_0$) e dalla temperatura ISA a quota 0 m ($TEMP_0$).

richiede in input:

double h [m] quota aria

e memorizza il valore calcolato nella variabile puntata da

*double * dens* [Kg/m^3] densità calcolata

per il calcolo della grandezza viene utilizzata la seguente formula

$$dens=DENS_0*\left(1+\left(\frac{ALPHA*h}{TEMP_0}\right)\right)^{4.255879} \qquad \textit{Formula 3.3}$$

La funziona ritorna 0 in caso di esecuzione corretta, -1 se i dati fossero non compatibili con le ipotesi di range compreso fra (0 11000)m.

- ***int Caffe_Air_Mach(double PS, double PD, double * M);***

questa funzione calcola il numero di Mach, ovvero il rapporto fra la velocità di volo e la velocità del suono.
La grandezza viene calcolata utilizzando la densità ISA di quota 0 m ($DENS_0$), e la temperata ISA a quota 0 m ($TEMP_0$).

richiede in input:

double Ps [Pa] pressione statica

double Pd [Pa] pressione totale

e memorizza il valore calcolato nella variabile puntata da

*double * M* [] Numero Mach

per il calcolo del Numero di Mach viene utilizzata la seguente formula (ipotesi di flusso subsonico):

$$M = \sqrt{\frac{\left(\frac{Ps+Pd}{Ps}\right)^{1/3,5} - 1}{0,2}} \qquad \textit{Formula 3.4}$$

La funziona ritorna 0 in caso di esecuzione corretta, -1 se i dati fossero non compatibili con le ipotesi, o se il valore sotto radice fosse negativo.

- ***int Caffe_Air_TS(double M, double TAT, double * TS);***

questa funzione calcola la temperatura di aria statica, ovvero la temperatura che l'aria avrebbe in condizione di immobilità.

richiede in input:

double M [] Numero Mach

double TAT [K] Temperatura indicata

e memorizza il valore calcolato nella variabile puntata da

*double * Ts* [K] Temperatura indicata

La grandezza viene calcolata utilizzando la temperatura misurata (TAT o temperatura indicata), e il numero di Mach.

per il calcolo della temperatura statica viene utilizzata la seguente formula (con ipotesi di flusso subsonico e recovery factor *r*=1)

$$TS = \frac{TAT}{(1 + 0.2\mathrm{rM})} \qquad \textit{Formula 3.5}$$

La funzione ritorna 0 in caso di esecuzione corretta.

- ***int Caffe_Air_CAS(double PD, double * CAS);***

questa funzione calcola la velocità Calibrata (CAS), ovvero la velocità corretta per gli errori dei sensori e di posizionamento ed è di fondamentale importanza per il volo in quanto descrive la pressione dinamica, ergo le forze in gioco, a dispetto delle diverse condizioni di quota.

richiede in input:

double Pd [Pa] pressione dinamica

e memorizza il valore calcolato nella variabile puntata da

*double * CAS* [m/s] Calibrated Air Sped

Prima di tutto viene definito il valore costante del Numero di Mach in condizioni Isa Standard (quota zero SL) così calcolato:

$$M0 = \frac{\sqrt{GAMMA\, P_0}}{DENS_0} \qquad \textit{Formula 3.6}$$

successivamente viene calcolata la velocità calibrata con la seguente formula:

$$CAS = \sqrt{[5 * ((PD/P_0) + 1)^{1/3.5} - 1]} \qquad \textit{Formula 3.7}$$

La funzione ritorna 0 in caso di esecuzione corretta, -1 se i dati fossero non compatibili con le ipotesi, o se il valore sotto radice fosse negativo.

- ***int Caffe_Air_TAS(double TS, double M, double * TAS);***

questa funzione calcola la velocità Vera (TAS), ovvero la reale velocità del velivolo rispetto al flusso d'aria in cui questo vola.

richiede in input:

double Ts [K] temperatura statica

double M [ ] Numero di Mach

e memorizza il valore calcolato nella variabile puntata da

*double * TAS* [m/s] True Air Sped

per il calcolo della velocità Vera viene utilizzata la seguente formula:

$$TAS = \gamma R_R * M * \sqrt{(TS)} \qquad \textit{Formula 3.8}$$

dove

γ *indica rapporto fra ca*lore specifico a pressione e volume costante (γ=1,4)

RR è la costante dei gas per l'aria (RR=*287.0529*)

La funziona ritorna 0 in caso di esecuzione corretta, -1 se il valore sotto radice fosse negativo.

- ***int Caffe_Air_EAS(double TAS, double dens, double * EAS);***

questa funzione calcola la velocità Equivalente (EAS), ovvero la velocità ottenuta riportando la TAS in condizioni di quota ISA 0 in modo da conservare lo stesso valore di pressione dinamica.

richiede in input:

double TAS [m/s] Velocità vera
double dens [Kg/m^3] densità

e memorizza il valore calcolato nella variabile puntata da

*double * EAS* [m/s] Equivalent Air Sped

per il calcolo della velocità Vera viene utilizzata la seguente formula:

$$EAS = TAS \sqrt{(dens / DENS_0)} \qquad \textit{Formula 3.9}$$

Dove $DENS_0$ = 1,225 [Kg/m^3]

La funziona ritorna 0 in caso di esecuzione corretta, -1 se il valore sotto radice fosse negativo.

- ***int Caffe_Air_PS_data_Fill(p_s_data *ps_data, uint64_t time, double RC);***

questa funzione memorizza unicamente i valori di Rate of Climb istantaneo, e il momento di rilevazione del dato negli ultimi elementi dei due vettori;uno di double per il valore di Rateo di salita e uno di uint64_t (interi senza segno a 64 bit) per il tempo espresso in millisecondi.

I due vettori sono parte di una apposita struttura predefinita denominata RC_p_s_data.
Questa funzione viene chiamata solamente durante l'esecuzione di *Caffe_Air_RC.*

Richiede in input:

uint64_t time [millisec] millisecondi da inizio esecuzione programma

double RC [ft/min] rateo di salita

e memorizza i valori nelle variabili punte da

*p_s_data *ps_data* puntatore alla struttura contenente i vettori di tempo e rateo di salita

La funziona ritorna 0 in caso di esecuzione corretta.

- ***int Caffe_Air_RC(p_s_data *RC_ps_data, double * RC);***

questa funzione calcola il rateo di salita (RoC) / discesa del velivol, per l'implementazione si è partiti dalla considerazione che il rateo di salita è la derivata temporale delle variazioni di quota ovvero:

$$Rc = \frac{\partial h}{\partial t} \qquad \textit{Formula 3.10}$$

equazione che è ovviamente valida nel continuo mentre il sistema GS elabora dati digitali ergo discreti, riconducendosi ad una forma digitale si implementa un rapporto incrementale del tipo:

$$Rc_d = \frac{h_{k+1} - h_k}{t_{k+1} - t_k} \qquad \textit{Formula 3.11}$$

utilizzabile solo in teoria,

infatti analizzando il gruppo di dati quota-pressione realmente acquisiti durante un volo rappresentati in 18 si nota una dinamica di incremento lineare nel tempo affetta da un certo rumore di misura, ci si attende quindi la ricostruzione di un rateo di salita quasi costante.

La ricostruzione del Rateo di salita calcolato con la Formula 3.11 fornisce l' andamento descritto da 19 dove appare chiaro che i dati così ottenuti non siano credibili, infatti a fronte di un sviluppo crescente della variabile, il rapporto incrementale assume valori altalenati con salti fra due istanti successivi di quasi 2000 ft/min.

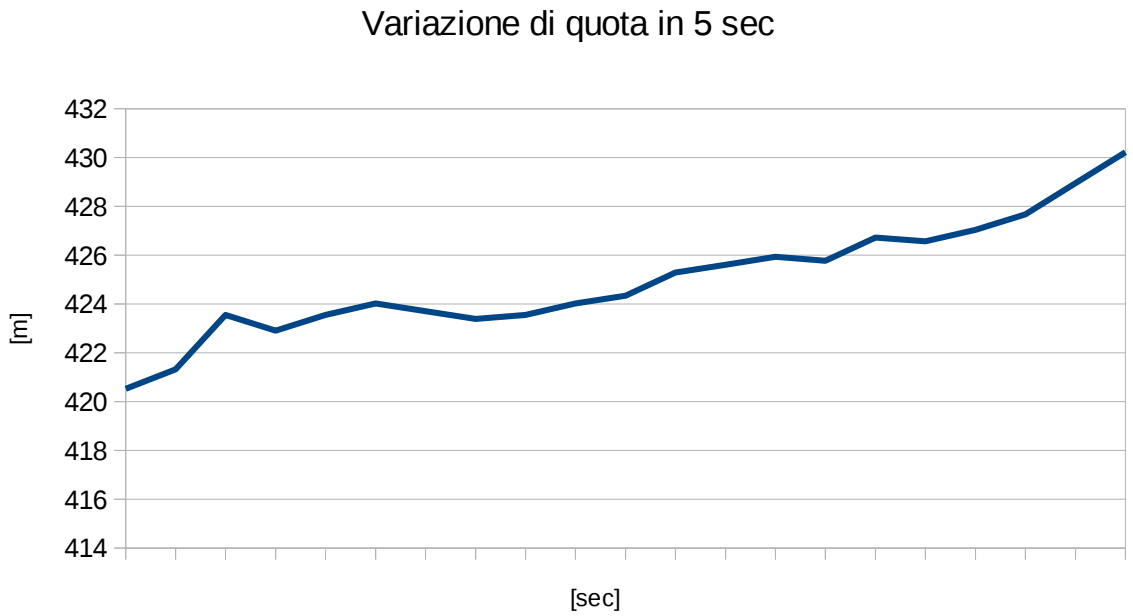

Figura 18: variazione di altitudine in 5 sec

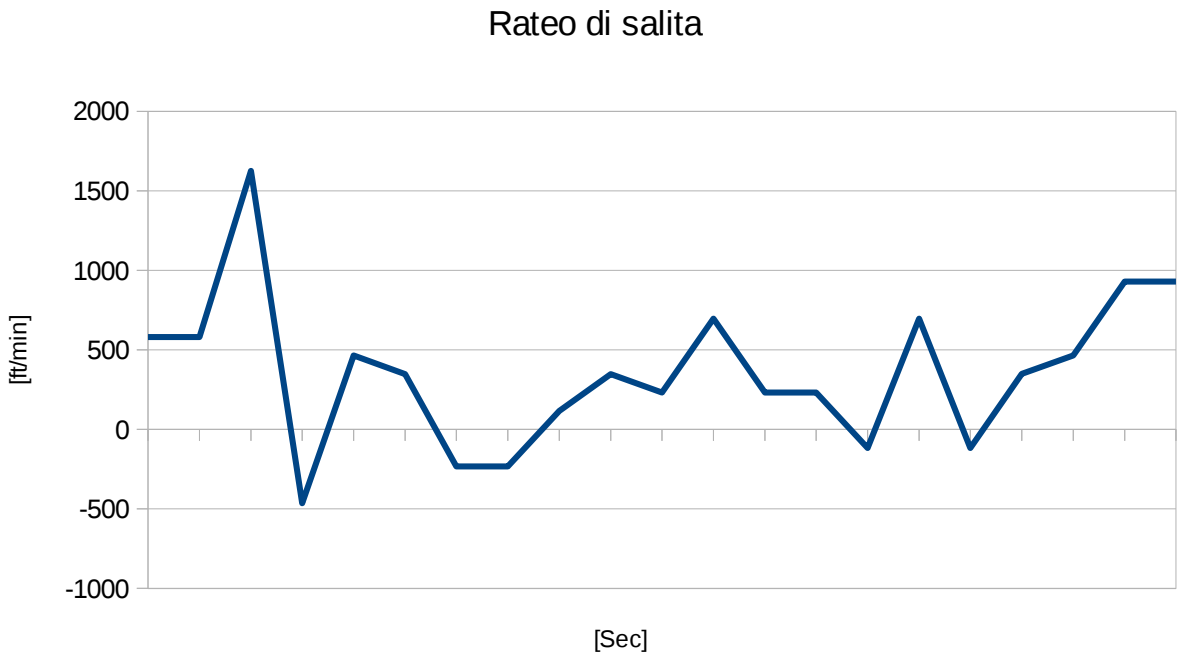

Figura 19: Rateo di salita non filtrato

Si ricorre allora all'utilizzo di una media con finestra mobile come filtro passa basso, al fine di eliminare i rumori di alta frequenza.

Infatti definendo come:

$$h_k = \frac{\sum_{i=0}^{i=N-2} h_i}{N-1}$$
$$h_{k+1} = \frac{\sum_{i=1}^{i=N-1} h_i}{N-1}$$

Formule 3.12

si ha l'effetto di applicare un filtro passa basso con funzione di trasferimento discreta ottenibile da:

$$Y(z) = \frac{\sum_{i=0}^{i=M-1} z^{-i}}{N} X(Z) = \left(\frac{1}{N} \frac{1-z^N}{1-z^{-1}} \right) X(z) = H(z) X(z)$$

Formula 3.13

successivamente utilizzando il software libero Scilab per simulare il comportamento in frequenza di tale filtro si ottiene il seguente grafico:

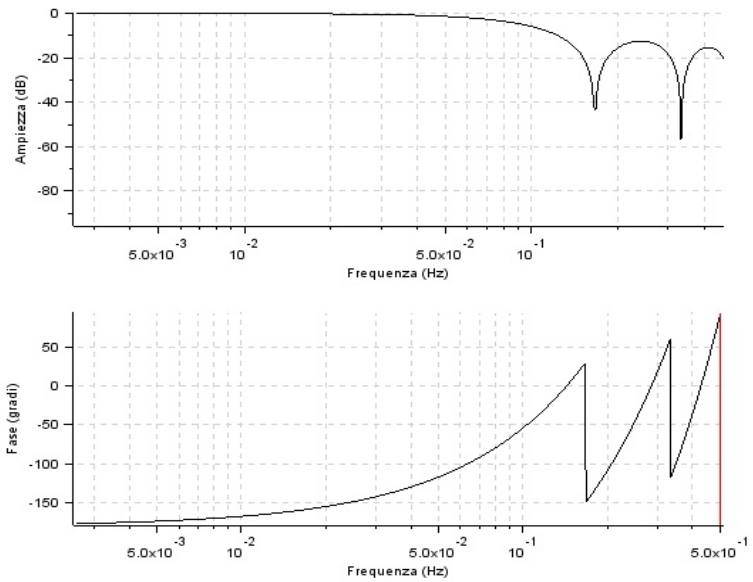

Figura 20: esempio di effetto filtrante di una media su sei campioni

Viene scelto dunque di calcolare RC come:

$$Rc_d = \frac{Rc_{k+1} - Rc_k}{t_{k+1} - t_k}$$ *Formula 3.14*

dove ogni elemento è la media così definita:

$$Rc_k = \frac{h0+h1+h2+h3+h4+h5}{6}$$
$$Rc_{k+1} = \frac{h1+h2+h3+h4+h5+h6}{6}$$
$$t_k = \frac{t0+t1+t2+t3+t4+t5}{6}$$
$$t_{k+1} = \frac{t1+t2+t3+t4+t5+t6}{6}$$ *Formula 3.15*

si ottiene quindi un rateo di salita con l'andamento presentato in 21

che confrontato con il comportamento non filtrato (19) dimostra la bontà del filtraggio introdotto.

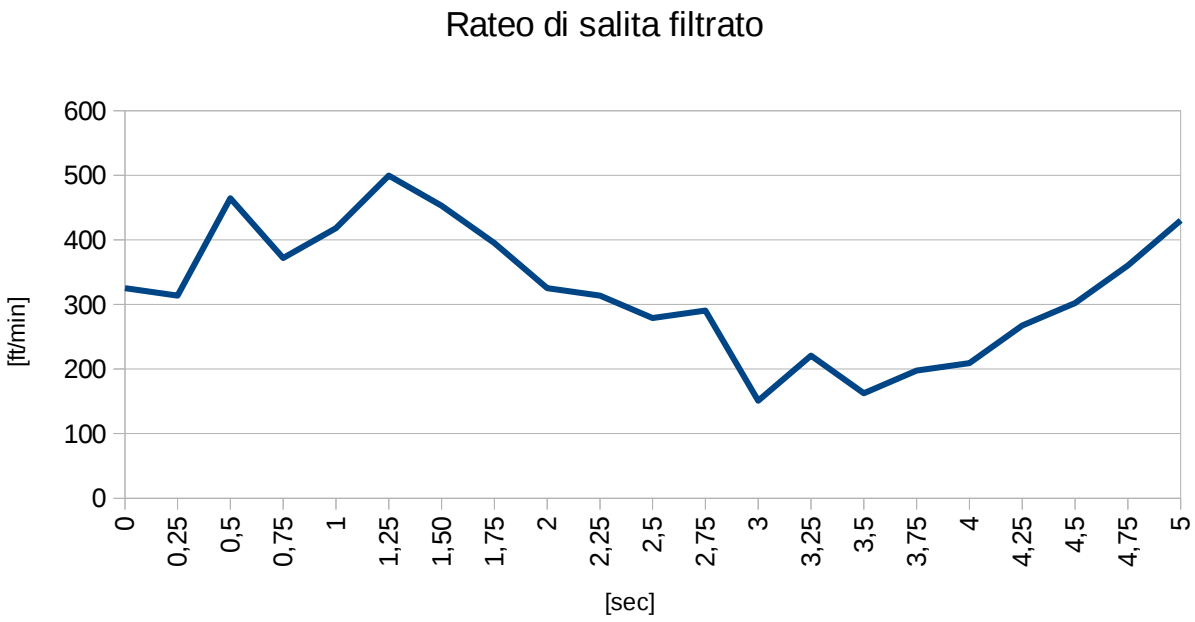

Figura 21: rateo di salita filtrato

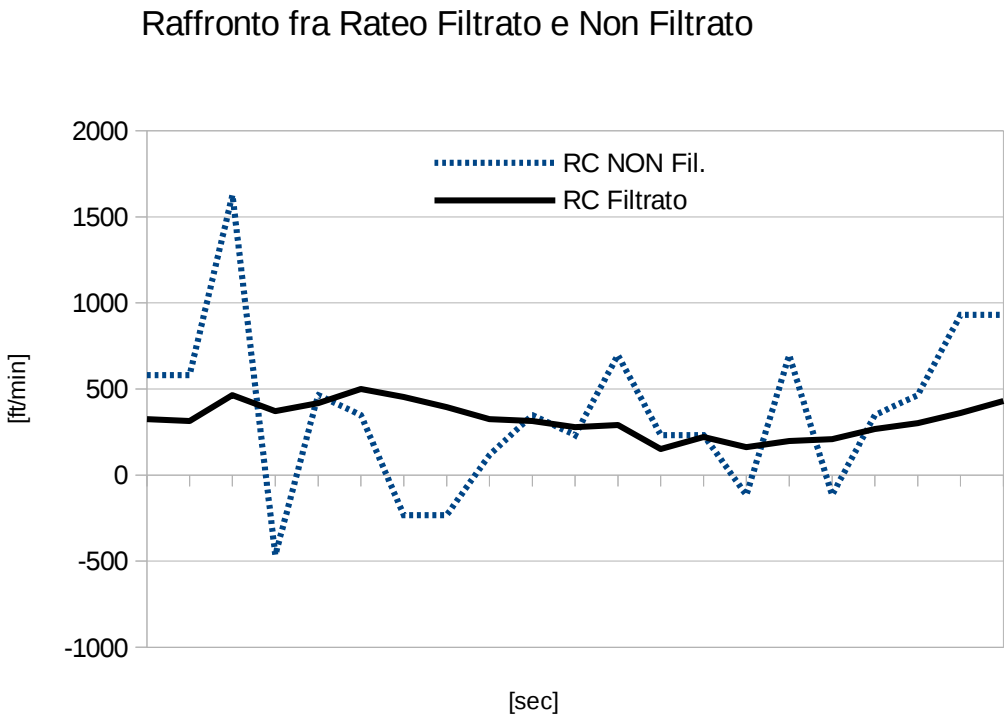

Figura 22: Raffronto fra Rateo Filtrato e non filtrato

E' bene ricordare che il filtro numerico introdotto ha effetto anche sulla prontezza di risposta del sistema, quindi è meglio privilegiare un andamento più discontinuo del rateo di salita al fine di aumentare la velocità di risposta, avendo utilizzato una media su 6 elementi la risposta ad uno scalino giunge a completo regime solo dopo 1,5 secondi; un tempo adeguato alle dinamiche delle comuni fasi di volo, ma leggermente superiore ai tempi tipici degli strumenti analogici utilizzati attualmente in aviazione ultraleggera.

Quote di riferimento e Q code

La determinazione della quota di un velivolo pone sempre il problema di standardizzare un punto di riferimento posto quale origine della misurazione, e mentre per i metodi più moderni, quali il GPS, si è standardizzato l'uso del centro della terra per Misurazioni ECEF o il livello del mare del geoide WGS84 per i riferimenti locali, nelle quote aria vengono definiti tre diversi metodi di azzeramento, la pressione ISA standard a quota 0, la pressione a livello mare e la pressione corrispondente ad una specifica stazione di terra.

Ovviamente ogni riferimento trova applicazione in specifiche problematiche legate alla fase di volo e non è possibile definire uno standard preferibile agli altri.

Storicamente ognuno di questi riferimenti, data la loro importanza, è stato legato ad un Q codes, ovvero quelle codifiche utilizzate sin dall'epoca degli albori dell'aviazione per inviare sinteticamente richieste standardizzate.

Il Q code prevede 676 possibili combinazioni di lettere che spaziano da QAA a QZZ, e quelli riservati per la definizione della quota sono QNE QNH QFE. Queste tre sigle vanno quindi a definire la quota del velivolo secondo i diversi riferimenti utilizzati più comunemente nelle fasi di volo.

- QFE definisce la quota aria misurata rispetto ad una stazione di terra; si ottiene settando come riferimento dell'altimetro la pressione della GS.

- QNH definisce la quota aria misurata rispetto al livello del mare; si ottiene settando come riferimento dell'altimetro la pressione calcolata a livello del mare a partire dalla pressione di una stazione di terra di cui sia nota l'altitudine.

- QNE definisce la quota aria misurata rispetto alla quota 0 a pressione ISA standard, si ottiene settando come riferimento dell'altimetro la pressione ISA standard ovvero 101325 [Pa].

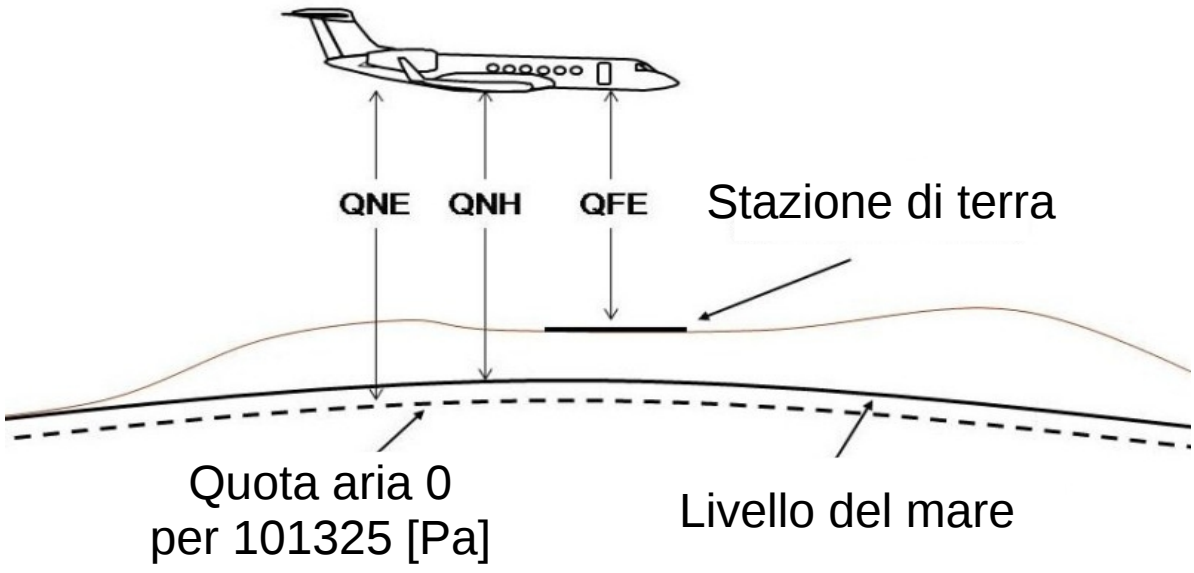

Figura 23 Riferimenti QFE QNH QNE

si procede a definire ora le tre funzioni in grado di calcolare la quota espressa in ft per ognuno dei riferimenti con le rispettive funzioni.

- ***int Caffe_Air_QFE(double PS, double field_press, double * QFE);***

questa funzione viene utilizzata per il calcolo della quota aria misurata rispetto alla base di terra

richiede in input:

double PS [Pa] pressione statica rilevata dalla presa statica del tubo di Pitot posto sul velivolo

double field_press [Pa] pressione statica rilevata dalla stazione di terra

e memorizza il valore calcolato nella variabile puntata da

*double * QNE* [ft] quota aria espressa in [ft] come tuttora in uso in tutto il mondo[1]

per il calcolo della grandezza viene utilizzata la seguente formula:

$$QFE = 0.3048\left(\frac{288,15}{-0,0065}\right)*\left(\left(\frac{PS}{field_{press}}\right)^{(1/5.255879)}-1\right)[ft] \qquad \textit{Formula 3.16}$$

restituisce 0 se correttamente eseguita, -1 se il rapporto fra pressioni risultasse negativo

- ***int Caffe_Air_QNE(double h, double * QNE);***

questa funzione viene utilizzata per il calcolo della quota aria misurata rispetto alla pressione standard ISA di quota 0 ovvero 101325 [Pa]

richiede in input:

double h [m] quota già calcolata espressa in metri

e memorizza i valori calcolati nelle variabili puntata da

*double * QNE* [ft] quota aria espressa in [ft]

questa funzione semplicemente converte la quota pre-calcolata in [ft]
ovvero idealmente viene calcolata come:

$$QNE = 0.3048\left(\frac{288,15}{-0,0065}\right)*\left(\left(\frac{PS}{P_0}\right)^{(1/5.255879)}-1\right)[ft] \qquad \textit{Formula 3.17}$$

1 eccezion fatta per la Russia, e buona parte del blocco ex sovietico

restituisce 0 se correttamente eseguita, -1 se h risultasse NAN.

- ***int Caffe_Air_QNH(double PS, double sea_press, double * QNH);***

questa funzione viene utilizzata per il calcolo della quota aria misurata rispetto al livello del mare
richiede in input:
double PS [Pa] pressione statica rilevata dalla presa del tubo di Pitot posto sul velivolo
double sea_press [Pa] pressione statica calcolata a livello del mare
e memorizza il valore calcolato nella variabile puntata da
*double * QNH* [ft] quota aria espressa in [ft]
per il calcolo della grandezza viene utilizzata la seguente formula

$$QNH = 0.3048\left(\frac{288{,}15}{-0{,}0065}\right)*\left(\left(\frac{PS}{sea_{press}}\right)^{(1/5.255879)} - 1\right)[ft] \qquad \textit{Formula 3.18}$$

restituisce 0 se correttamente eseguita, -1 se il rapporto fra pressioni risultasse negativo.

- ***int Caffe_Air_Sea_Pres(double field_press, double field_h , double * sea_press);***

questa funzione viene utilizzata per il calcolo della pressione a livello mare, quando sia nota la pressione rilevata dalla stazione e l'altitudine della stazione stessa rispetto al livello mare.
richiede in input:
double field_h [m] altitudine della stazione rispetto al livello mare
double field_press [Pa] pressione statica rilevata dalla stazione di terra
e memorizza il valore calcolato nella variabile puntata da
*double * sea_press* [Pa] pressione sul livello mare
per il calcolo della grandezza viene utilizzata la seguente formula:

$$P_{sea\,level} = field_{press}\left(-\left(1 \pm \frac{0{,}0065}{288{,}15} * h_{field}^{5{,}255879}\right)\right) \qquad \textit{Formula 3.19}$$

la funzione restituisce 0 se correttamente eseguita.

3.8 Principio di funzionamento del programma CaffeNet_server

Ora che sono state definite tutte le librerie necessarie, è possibile affrontare lo sviluppo del server vero e proprio; questo, come già accennato, è pensato per essere installato sull'unità di calcolo principale e ha il compito di ricavare i dati dai datagrammi CAN, per poi ingegnerizzarli, e raggrupparli secondo un ben preciso criterio ed infine spedirli via TCP/IP ad altri terminali

Si è scelto di separare la funzione di memorizzazione dal corpo principale del programma, per permettere un più comodo sviluppo successivo, infatti creando un Client addetto alla memorizzazione, cioè una applicazione che possa ricevere i pacchetti spediti dal Server GS e salvarli su HD, ogni terminale collegato può assolvere al compito backeup dei dati. Si può pensare di utilizzare un Client installato su un pc collocato in prossimità del sistema Gs e collegato attraverso una rete lan o Wlan, o ricorrere ad un Client posto a grande distanza sfruttando un collegamento internet, o infine, si potrebbe sfruttare lo stesso programma installandolo sull'unità principale di calcolo, così che questa assolava a entrambi i compiti contemporaneamente.

Il Server realizzato è stato denominato *CaffeNet_server* e in ognuna delle 7 versioni implementate svolge i seguenti compiti:

Fase 1: **Inizializzazione Programma**; fase che provvede all'allocazione delle variabili e dell'eventuale memoria dinamica, alla verifica e parse delle opzioni d'avvio e creazioni degli eventuali sistemi di comunicazione fra processi.

Fase 2: **Inizializzazione parametri**; in questa fase il programma provvede alla lettura del file di configurazione, a memorizzare i coefficenti utilizzati per l'ingegnerizzazione dei dati, e a inizializzare le strutture.

Fase 3: **attesa e lettura dati**; il programma attende (se necessario) l'arrivo di un nuovo telegramma in formato CanEid e lo memorizza nello Stack.

Fase 4: **estrapolazione delle informazioni dal telegramma CanEid**; in questa fase viene chiamata la funzione *u8canEidToCaffe*, e tutti i dati contenuti nel datagramma vengono estrapolati e suddivisi in una struttura *t_caffeTelegram.*

Fase 5: **Riconoscimento tipo di dato**; dalla variabile *ParameterId* della struttura *t_caffeTelegram* il programma è in grado di identificare il parametro contenuto nel telegramma e allo stesso modo dal valore di *dataType* ha la possibilità di riconoscere il formato di memorizzazione ed eventualmente può procedere alla conversione in un formato più consono per le successive manipolazioni.

Fase 6: **Ingegnerizzazione del dato**; conoscendo il tipo di parametro il programma è in grado di eseguire l'ingegnerizzazione del dato attraverso una chiamata alla funzione *pol_eng* della libreria *CaffeEng*, convertendo il valore contenuto nel telegramma in un*a* grandezza fisica.

Fase 7: **Inserimento del dato nella struttura assegnata**; il programma essendo a conoscenza del tipo di parametro elaborato è in grado di separare i dati sulla base del criterio enunciato nella sezione 3.5, ovvero considerando le caratteristiche di ogni informazione al fine della ricostruzione delle condizioni di volo e dello stato dei sistemi.

Fase 8: **Verifica completamento struttura**; in questa fase il programma verifica se la struttura in cui si è appena inserito il dato sia pronta per essere promossa per l'invio. Per tale

decisione, il server si avvale di due criteri in somma logica (ovvero legati fra loro con condizione OR). Il primo criterio è la presenza nella struttura di tutti i dati previsti, il secondo è una condizione di TimeOut che si genera quando una struttura non sia ancora completa ma i dati contenuti siano considerati prossimi a essere obsoleti e dunque meno attendibili per rappresentare lo stato attuale delle condizioni del velivolo. Al verificarsi della condizione di TimeOut il programma valuta la possibilità di una promozione all'invio della struttura incompleta se, e solo se, questa contenga i dati minimi per fornire nuove informazioni utili alla ricostruzione della situazione attuale. Nel caso che non tutti i valori essenziali siano presenti la struttura viene ri-inizalizzata (con SNAN).

Fase 9: **Invio struttura via UDP**; il programma procede, alla spedizione di un singolo datagramma UDP, utilizzando la sequenza di byte componenti la struttura dati come un array di char .

Fase 10: **Reinizializzazione della struttura***;* una volta inviati i dati il programma provvede a ri inizializzare la struttura con SNaN e viene aggiornato il tempo di TimeOut.

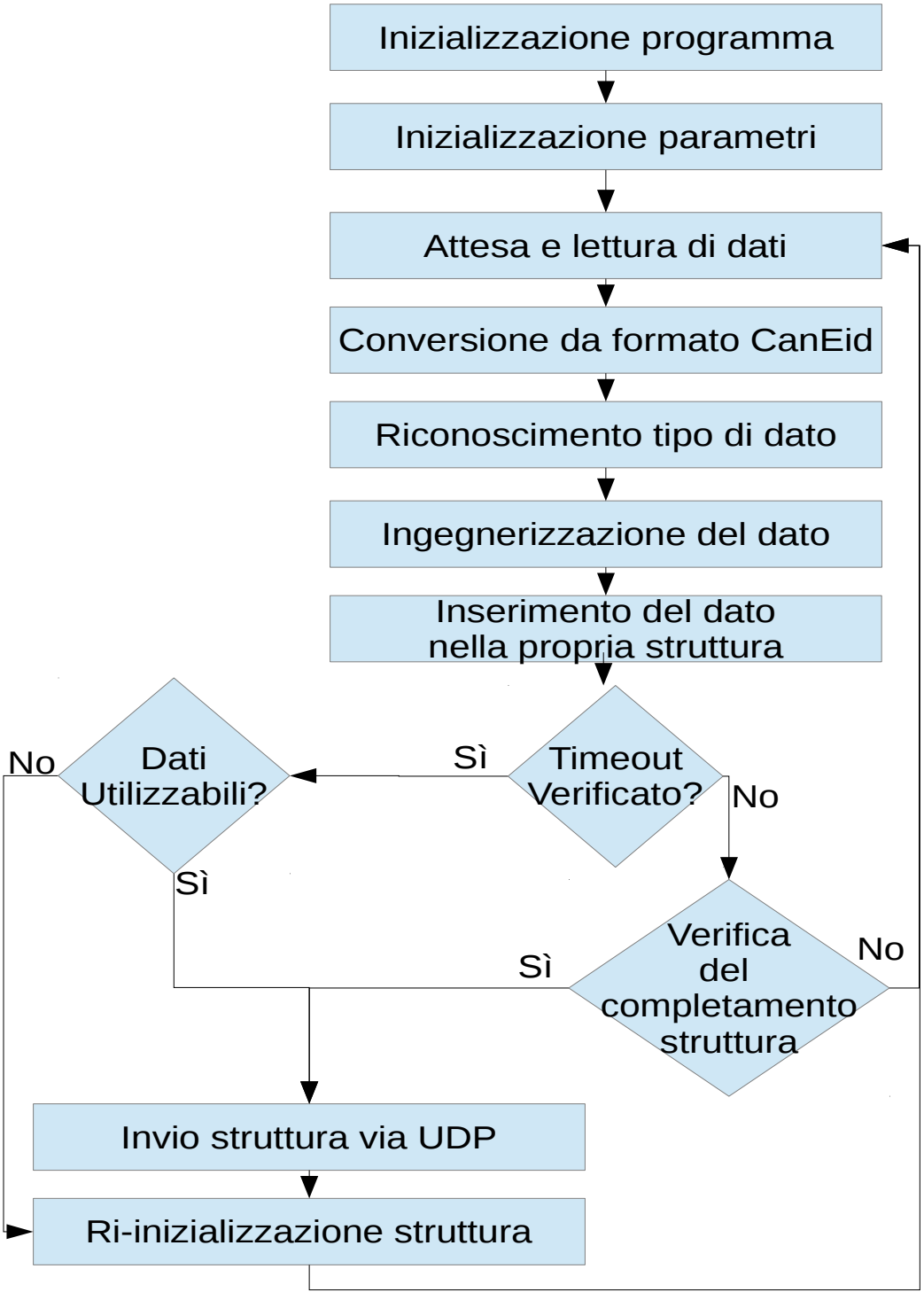

Figura 24 schema generico del funzionamento del sistema GS

Si procede ora alla descrizione di alcune delle versioni realizzate, al fine di evidenziare le peculiarità introdotte, trattando dapprima le caratteristiche che accomunano tutte le versioni e successivamente concentrando l'attenzione sulle peculiarità che contraddistinguono le diverse versioni e le scelte di progetto intraprese.

3.8.1 La scelta del protocollo UDP e scelte connesse

Solo poche scelte progettuali sono presenti in tutte le versioni e fra queste si annovera la scelta di utilizzare il protocollo UDP sulla porta 22333.

UDP è stato già descritto nel paragrafo 3.3.2 come un protocollo connectionless che non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi, ma che può contare su un header di dimensione ridotte.

Si è scelto dunque di inviare i dati in pacchetti di dimensioni comprese fra 200 e 400 Byte, al fine di aumentare il rapporto di efficienza di trasmissione

$$\frac{Payload}{dimensione\,totale\,pacchetto} = \frac{Payload}{payload + Header}$$

garantendo quindi un utilizzo migliore della banda di trasmissione cercando di minimizzare il numero di bytes inviati non utilizzabili per il trasporto delle informazioni, contemporaneamente si garantisce anche un invio atomico delle strutture in tutte le reti più comuni, infatti la dimensione massima di un pacchetto inviato sarà di 432 bytes a livello del layer transmission (400 bytes di payload + 32 Byte di UDP header) quindi ben al di sotto della Maximum Transmission Unit spesso posta a 1024 Byte o 1526 Byte.

Peculiarità del protocollo UDP sono la non certezza dell'ordine di arrivo dei pacchetti e la non ritrasmissione di datagrammi persi.

Per garantire la sequenzialità dei dati, viene inserito nell'header di ogni struttura trasmessa, un indice incrementato ad ogni invio, invece per quanto riguarda la non ritrasmissione dei pacchetti persi, si preferisce mirare ad attenuare l'effetto, piuttosto che a cancellarlo al fine di non complicare eccessivamente il programma e non aggiungere ulteriore carico alla rete.

Si decide l'inserimento nell'header di un secondo contatore, diverso per ogni tipo di struttura che venga incrementato all'invio dello specifico payload ad esso collegato, così facendo nel caso in cui un pacchetto non sia ricevuto, il Client, è in grado di constatare la perdita di un datagramma sin dal primo pacchetto ricevuto successivamente e di identificare il tipo di dati non ricevuto al sopraggiungere della successivo payload del medesimo genere.

3.8.2 Server MK 1

Il primo server sviluppato, detto MK1, è stato pensato prevalentemente come uno studio avanzato di fattibilità del progetto che evidenziasse tutte le criticità del sistema finale. Le caratteristiche richieste in questa prima implementazione sono state l'adempimento di tutte le fasi essenziali, dalla lettura del dato contenuto nel pacchetto CAN sino all'invio delle strutture via UDP, senza badare però, all'ottimizzazione della velocità di esecuzione. Si ricerca inoltre la maggiore linearità / leggibilità / tracciabilità del codice, in modo che questo sorgente possa fungere da linea guida per la conoscenza pratica delle problematiche legate a tutte le fasi necessarie allo svolgimento delle operazioni.

In questo semplice modello si è cercato di eliminare o di limitare tutte le problematiche legate a possibili race-condition e alla comunicazione fra processi, condizioni che possono generare bug molto difficilmente identificabili vanificando l'attenzione posta alla leggibilità del codice.

Il sistema completo di processi e/o thread sarà implementato nelle versioni successive, quando le funzioni di base e le librerie saranno già state testate e abbiano dimostrato l'affidabilità necessaria, potendo così concentrare l'attenzione prevalentemente sull'interazione fra le esecuzioni concorrenziali.

Le prime considerazioni di progetto effettuate sono state sul tipo di memoria da utilizzare per allocare le strutture che avrebbero dovuto contenere i dati durante la fase di riempimento

Ad un'analisi superficiale si potrebbe pensare che queste strutture vengano riempite e inviate con una successione precisa, e questo porterebbe a ritenere l'allocazione dinamica di memoria (tramite il comando malloc()) come un metodo conveniente, ma in realtà il fatto che una struttura possa essere riempita con dati campionati con frequenze differenti fra loro, porta alla necessità di avere sempre una struttura di ogni tipo disponibile, si preferisce quindi, rivolgere l'attenzione verso una implementazione con allocazione statica di una struttura per tipo direttamente all'interno dello stack del main, e procedere alla condivisione delle con le funzioni chiamate attraverso “passaggio per puntatore”.

La struttura del codice allora sarà formata dalla successione

1. definizione delle variabili per il server UDP
2. definizione delle variabili per lo spacchettamento
3. definizione delle variabili per il controllo del tempo
4. chiamata alla procedura di configurazione dei parametri tramite un file di configurazione
5. parse delle opzioni di avvio
6. creazione del socket UDP
7. inizializzazione dell'indirizzo e della porta di trasmissione
8. lettura di un datagramma Can (con la chiamata BitFileGetBitsInt)
9. spacchettamento del datagramma
10. procedura di riempimento della struttura dati

11. procedura di riconoscimento dello stato della struttura dati
12. eventuale invio della struttura
13. eventuale ri-inizializzazione struttura inviata
14. ripetizione dell'algoritmo dal passo 8 fino al termine del flusso di datagrammi CAN

Molti dei precedenti punti sono stati già ampiamente descritte durante la presentazione delle librerie di appartenenza, la trattazione verrà ora concentrata sulle sole fasi che saranno comuni a tutte le versioni successive.

Parse opzioni di avvio

Il parse delle opzioni di avvio è stato sviluppato in modo canonico prevedendo le opzioni:

- h: help, non viene avviato il programma ma viene mostrato un breve riassunto sulle funzionalità delle opzioni
- t: test mode, non è necessario nessuno streaming o nessun file di backup da cui leggere i dati, le strutture vengono riempite con parametri noti per verificare il corretto svolgimento delle operazioni.
- v V W: verbose, le tre opzioni indicano in ordine crescente dettaglio di visualizzazione, ovvero quanto debbano essere prolissi i commenti presentati a schermo durante l'esecuzione.

Procedura di riconoscimento dello stato della struttura dati

In questo primo prototipo di CaffeNet_server è stato approntato un metodo di riconoscimento sulla condizione di riempimento della struttura basandosi unicamente sulla presenza di tutti i dati, ed eventuale ri-inizializzazione della stessa una volta che si presentasse il Timeout.

La scelta di inviare la struttura quando questa presenti tutti i parametri appare ovvia, mentre quella di “cancellare” i dati dopo che sia trascorso un determinato lasso di tempo, ovvero dopo che si sia verificata la condizione di timeout, deriva dalla necessità di rilasciare sempre valori attendibili. Infatti, se un sistema a seguito di un guasto fornisse dati in modo discontinuo, si potrebbe ottenere una struttura popolata troppo lentamente, senza la possibilità di discernere quali valori siano attuali e quali ormai si riferiscano ad alcuni istanti prima, questo implicherebbe che tutte le misure ricostruite a partire dai questi parametri incerti, possano essere affette da una incertezza legata alla prima, come ampiamente descritto nella numerosa letteratura sulla propagazione degli errori ([8]).

Si prevede, fin da questa fase, una implementazione più completa dei meccanismi di riconoscimento dello stato della struttura, sopratutto per la gestione delle eccezioni timeout.

Il metodo implementato in questo stadio di progettazione, consiste nell'allegare un valore double ad ogni struttura. In questo campo viene inserito il numero di millisecondi trascorsi fra l'istante d'inizio

di funzionamento del server e quando venga scritto il primo dato nella struttura; si prevede anche una flag in formato uint8_t che indichi se sia già avvenuta la prima scrittura.
Ad ogni iterazione il programma verifica la flag, se questa è settata a 0 provvede ad imporla a 1, e immediatamente inserisce il valore di tempo corrente nell'apposita variabile, poi scrive il dato ingegnerizzato nell'opportuna posizione. Se la flag di prima scrittura fosse già settata ad 1 il programma verifica che il tempo trascorso non ecceda il valore di timeout e provvede a inserire il dato nella corretta posizione, nel caso la condizione di timeout sia verificata il programma procede a ri-inizializzare la struttura e ad effettuare la prima scrittura con il dato corrente.

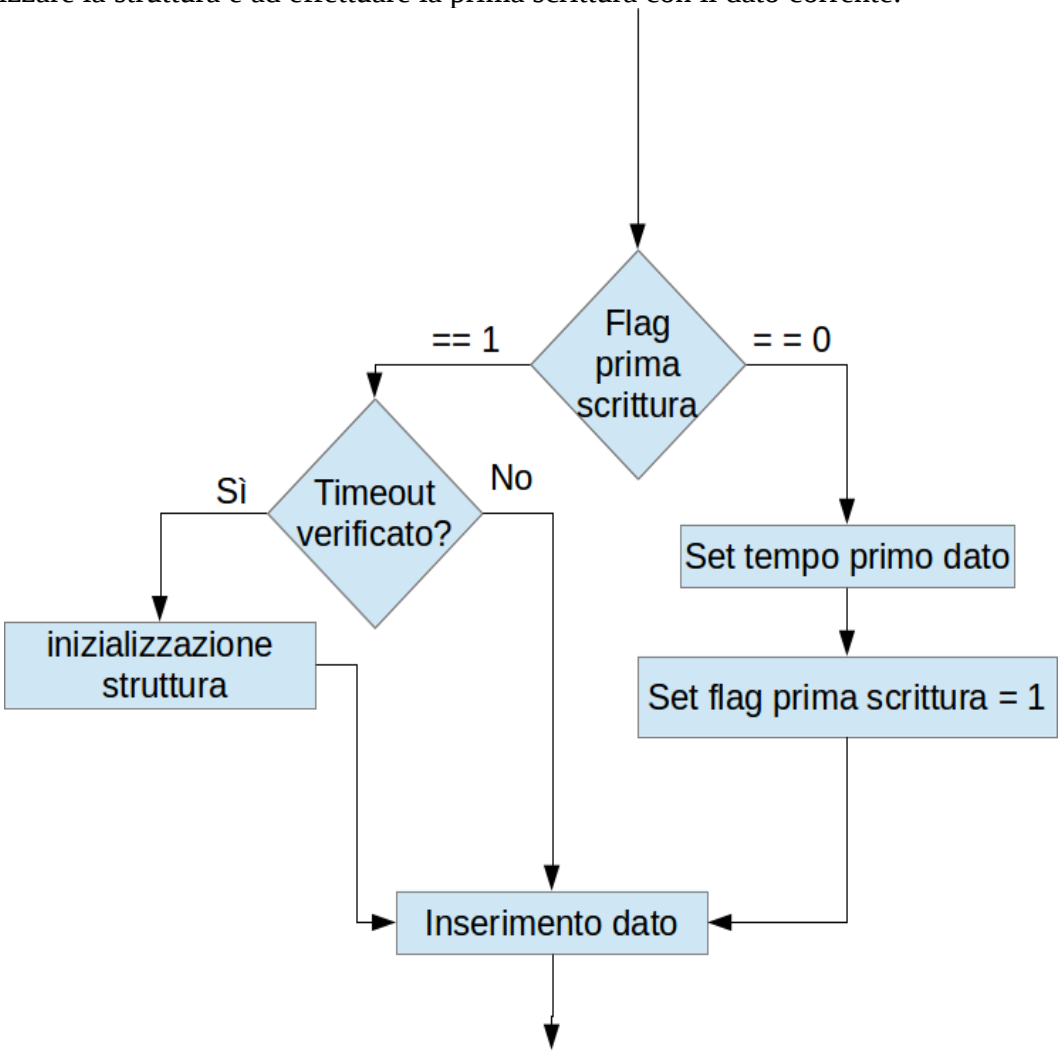

Figura 25 schema dell'algoritmo di inserimento del dato

Invio della struttura dati

Dopo aver determinato che una struttura sia promovibile alla fase di invio, con il metodo introdotto nel paragrafo precedente o con quello che verrà descritto nella sezione “Procedura avanzata di riconoscimento dello stato della struttura dati“, si procede all' invio.

In questa versione del server, che si sviluppa in un singolo processo, l'espletamento dell'invio consta di pochissime righe di codice, non essendo necessario implementare metodi di comunicazione fra processi o strategie per garantire l'esclusività di accesso alle risorse condivise.

L'invio avviene con sendto che è bene ricordare essere una funzione bloccante [1] se non viene appositamente attivata l'opzione O_NONBLOCK sul socket utilizzato.

Tralasciando tutte le necessarie verifiche del corretto svolgimento delle operazioni, e i comandi standard per l'invio, l'intero processo si riduce a una sola chiamata del tipo

int caffeNet_init (int type_str, t_u_complete t_str);* già ampiamente discussa nel paragrafo della libreria caffe_Net, utilizzata per ri-inizializzare le strutture dopo l'invio, ponendo tutti i valori dei dati contenuti nella struttura a valore SNaN (ove possibile).

Considerazioni sul Server MK 1

Pur essendo perfettamente funzionante questo server ha prestazioni, come atteso, ben lontane dall'essere compatibili con quelle desiderate. La gestione delle eccezioni sullo stato di riempimento è troppo conservativa, e tende a far scartare troppo facilmente una struttura che potrebbe essere completa oltre il 90%, quindi, come già anticipato nelle pagine precedenti, per le future versioni questo metodo sarà rivisto, e modificato per essere reso maggiormente efficiente.

Inaspettatamente si è presentato un'ulteriore inconveniente: testando il programma su un terminale piuttosto veloce si sono presentati fenomeni di lag di centinaia di millisecondi, una condizione molto peggiore delle aspettative, probabilmente imputabili alla ripartizione del tempo di calcolo effettuata dal Sistema Operativo, nelle versioni future si adotteranno particolari accortezze nella programmazione.

Il pregio di questo codice è di aver evidenziato tutte le problematiche che si possano presentare durante l'implementazione delle funzioni base del server e aver permesso di verificare la fattibilità del progetto in tempi ragionevoli.

Al termine della fase di studio e programmazione del Server MK1 si è avviata una breve campagna di test mirata alla verifica del corretto funzionamento di questa versione del software.

Tali prove sono state eseguite attraverso il software Wireshark [10] catturando i pacchetti UDP inviati e verificando che la codifica e la trasmissione degli stessi fosse eseguita senza errori, contemporaneamente si è verificato che le librerie precedentemente realizzate fossero correttamente implementate confrontando i risultati ottenuti utilizzando la modalità test che sfrutta il popolamento automatico delle strutture attraverso la funzione caffeNet_test descritta nel paragrafo 3.7.1.

3.8.3 Server da MK 2 a MK 6

In questa sezione inizialmente verranno introdotti i concetti utilizzati per lo sviluppo dei 5 server successivi a MK1, questi mirano ad ottenere prestazioni elevatissime da una macchina con 2 processori quad-core o un processore 8-core, e infine ci si concentrerà sulla descrizione del server Mk 6 che rappresenta il punto di arrivo di questo ramo dell'implementazione.

Processi contro Thread

Per sfruttare a pieno le capacità di un sistema multi-core è fondamentale utilizzare una programmazione multitasking, ovvero una tecnica che permetta l'esecuzione contemporanea di più “porzioni di codice oggetto”.

L'espressione “l'esecuzione contemporanea di più porzioni di codice oggetto” va a raggruppare le due entità preposte al multitasking, il processo e il thread.

Il processo è l'oggetto del sistema operativo a cui sono assegnate tutte le risorse di sistema per l'esecuzione di un programma, il tempo di calcolo della CPU, la memoria, e tutti i descrittori alle risorse/periferiche utilizzate.

Il thread è l'elemento a cui il sistema operativo assegna il tempo di calcolo della CPU per eseguire lo svolgimento del programma.

Volendo presentare meglio il concetto, il *processo*, in informatica, indica un'istanza di un programma in esecuzione; o meglio, un'attività controllata da un programma che si svolge su un processore, in genere sotto la gestione del SO, in ambiente Unix, il processo viene ad essere l'unità fondamentale con cui un sistema alloca ed utilizza le risorse.

Il concetto di processo è associato, ma comunque distinto da quello di thread con cui si intende invece un sottoprocesso in cui l'entità processo può essere suddiviso.

Il thread può essere eseguito a divisione di tempo o in parallelo ad altri thread legati al medesimo processo. Il parallelismo o la sequenzialità della reale esecuzione di due porzioni di codice oggetto strutturate per avere corso simultaneo dipende dal fatto che vi sia un core per ogni thread e da come il Sistema Operativo gestisca il multitasking, la differenza fra il reale corso del programma e ciò che il programmatore si attende che accada sfocia spesso in Race condition, ovvero errori dovuti alla mancanza di sincronismo fra i thread.

Volendo esasperare la sinteticità della descrizione al fine di provvedere maggiore chiarezza si può dire che, un thread è una parte del processo che viene eseguita in maniera concorrente ed indipendente internamente allo stato generale del processo stesso.

Un processo in esecuzione implica sicuramente la presenza di un thread, ma al primo possono essere associati più thread che vengono eseguiti in parallelo.

Una differenza sostanziale fra thread e processi consiste nel modo con cui essi condividono le risorse: i processi sono fra loro indipendenti, utilizzando diverse aree di memoria ed interagiscono soltanto mediante appositi meccanismi di comunicazione detti IPC (Inter Process Comunications Si veda la sezione 3.8.3), al contrario i thread di un processo tipicamente condividono le medesime informazioni di stato, la memoria ed altre risorse di sistema, per questo è necessario porre molta

attenzione alla scrittura e alla lettura in variabili utilizzate da più thread in quanto, fra thread può avvenire una condizione di accesso medesimo in scrittura o scrittura/lettura, o alla semplice lettura di una variabile che sia stata già modificata da un'altro thread, che può portare a errori difficili da identificare.

Ricapitolando brevemente, due processi implicano il doppio della memoria allocata, ma anche l'esigenza della creazione di due riferimenti a periferiche e socket, mentre due thread condividono la stessa area di memoria e le medesime risorse.

Detto ciò, risulta evidente che la creazione di un nuovo processo è sempre onerosa per il sistema, in quanto devono essere allocate le nuove risorse necessarie alla sua esecuzione (allocazione della copia dei segmenti di testo, stack e dati, riferimenti alle periferiche [...] operazioni tipicamente costose per il S.O. [1]).

Il thread invece è parte di un processo e quindi una sua nuova attivazione viene effettuata in tempi ridottissimi con oneri minimi per il SO, ma è bene rimarcare che l'utilizzo di thread necessita di una certa accortezza nell'evitare possibili accessi simultanei alla memoria o ad altre risorse condivise. Contemporaneamente va fatto notare come la comunicazione fra thread dello stesso processo sia molto più immediata da implementare e garantisca una maggiore velocità di “trasferimento” delle informazioni; infatti si possono scambiare informazioni su una parte di memoria allocata a tale scopo dal processo “generatore”, o meglio dal SO su richiesta del processo generatore, senza dover richiedere l'intervento del sistema operativo per ogni chiamata IPC.

Per quanto riguarda la gestione della memoria nel caso della creazione di processi, è già stata menzionata la teoria generale, indica che per 2 processi occorrono il doppio delle risorse che per un solo processo, ma in realtà, con lo sviluppo di sistemi operativi sempre più prestanti, si sono sviluppati metodi per ottimizzare le risorse e i tempi per la creazione di nuovi processi, infatti, il segmento testo sarà condiviso fra i due processi e tenuto in read-only tanto per il padre che per il figlio, mentre per gli altri segmenti Linux utilizza la tecnica del copy on write; questa tecnica comporta che una “pagina di memoria” venga effettivamente copiata per il nuovo processo solo quando avviene la prima scrittura e quindi lo stack (o la heap) del padre è realmente differente di quella del figlio.

In questo modo si rende molto più efficiente il meccanismo della creazione di un nuovo processo, non essendo più necessaria la copia di tutto lo spazio degli indirizzi virtuali del padre, ma solo delle pagine di memoria che sono realmente state modificate, e solo al momento della modifica stessa.

Appena terminata l'operazione di fork l'esecuzione procede dalla prima istruzione sia per il padre che per il figlio appena creato, di conseguenza viene utilizzata la condizione *if (pid1 == 0)* per far eseguire al figlio il proprio codice oggetto, mentre dopo il Jump legato a else sarà posto il codice oggetto del padre.

Inter Process Communication o IPC

In informatica l'espressione Inter Process Communication o IPC (comunicazione tra processi) si riferisce a tutte quelle tecnologie software il cui scopo è consentire a diversi processi o thread di comunicare tra loro scambiandosi dati e informazioni.

I processi solitamente risiedono sullo stesso computer ma vi sono casi in cui questi possano essere distribuiti su una rete. Tutti i sistemi operativi multitasking forniscono qualche meccanismo fondamentale di IPC.

Nell'implementazione di queste versioni di server sono stati sfruttati i seguenti IPC

- Segnale: notifica asincrona trasmessa da un processo ad un altro per mezzo del S.O., è il più semplice fra i metodi IPC. Tipicamente nessun dato viene trasmesso assieme al segnale, è stato spesso utilizzato per notificare cambi di stato sull'occupazione di risorse o notificare eventi pre-concordati.
- Pipe anonima: Le pipe, diversamente dai socket già descritti in precedenza, offrono un canale di comunicazione monodirezionale, quindi permettono un invio di dati a senso unico e per ottenere un reale scambio di informazioni fra due processi è necessario utilizzare una coppia di pipe. Il maggior pregio delle pipe si riscontra nella comunicazione fra processi (o thread) in cui le pipe siano state create prima della chiamata fork (o pthread_create), infatti solo in tal caso padre e figlio avranno già a disposizione entrambi gli Handle della pipe, ovvero la coppia di descrittori per l'accesso alla pipe, ed è sufficiente che ognuno dei due processi (o thread) chiuda quello non utilizzato per poter iniziare la comunicazione.

  La pipe è semplicemente dello spazio di memoria allocato per un buffer first in first out che il S O va ad allocare in kernel space, e i due processi accedono a questo spazio come se fosse un comunissimo file.

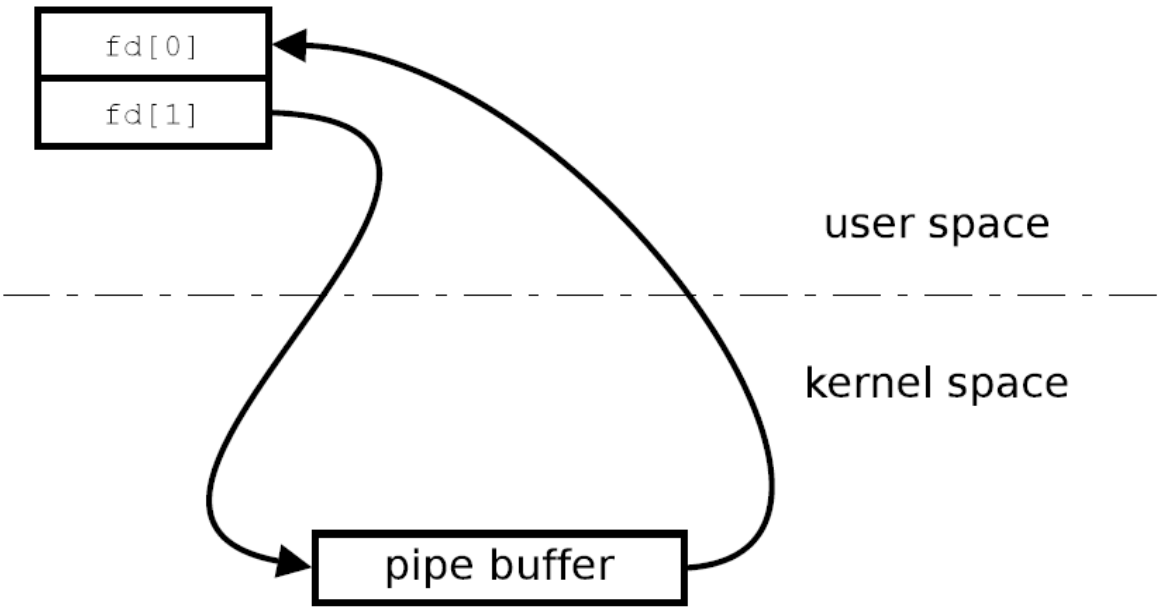

Figura 26: rappresentazione grafica di una pipe

- FIFO: IPC che può mettere in comunicazione due processi che non abbiano nessun rapporto fra loro, sono molto simili alle pipe ma non utilizzano due file descriptor per identificare il buffer di lettura scrittura ma viene utilizzato un “Inode” ovvero un descrittore che ha la preziosa proprietà di puntare un'area risiedente sul file system e che dunque è accessibile da ogni processo in esecuzione. Per essere messi in comunicazione i due processi devono conoscere solo il valore intero prestabilito che verrà assegnato alla fifo.
- Semaforo: tipo di dato astratto gestito da un sistema operativo multitasking per la sincronizzazione di risorse condivise tra processi o thread. È composto da una variabile intera e da una coda di processi, quindi si può affermare che questo sia un metodo che attraverso una IPC, permette la modifica di una variabile intera in modo boleano non consentendo di scambiare dati fra processi, ma servendo piuttosto come meccanismo di sincronizzazione o di protezione per sezioni critiche del codice. Infatti un semaforo è spesso utilizzato per regolamentare l'accesso a risorse condivise fra due o più processi “Cfr sez Inter process libary CaffeNet_Mutex“
- Memoria condivisa: è il tipico metodo di comunicazione fra thread che permette di comunicare utilizzando array o singole variabili allocate nella parte condivisa della memoria, riservata durante l'esecuzione del comando *pthread_create* la medesima tecnica IPC può essere implementata fra processi con la funzione *shmget della libreria sys/shm.h.*
- Mailbox: è una combinazione di un semaforo e una coda di messaggi (o pipe). Una volta inserito un messaggio nella pipe si effettua una chiamata “get” per recuperarlo. La chiamata “get” può essere bloccante o non bloccante ma non può essere asincrona. Esistono anche mailbox speciali che possono essere utilizzate per fornire segnali asincroni di warning o di errore fatale per l'esecuzione

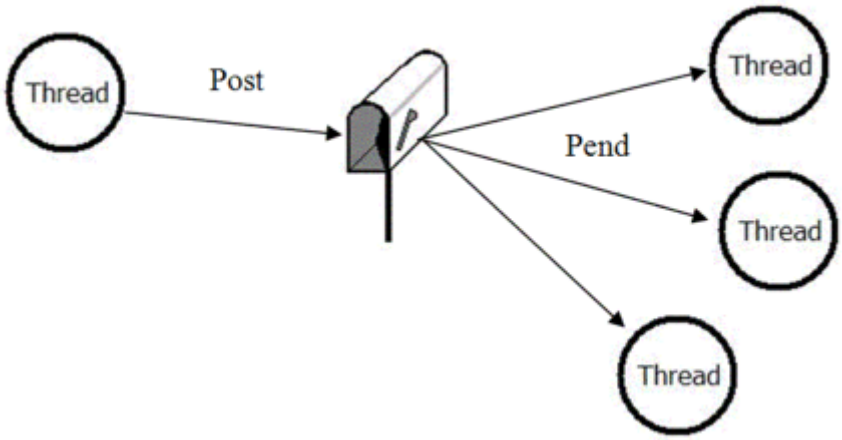

Figura 27: rappresentazione grafica di una mailbox

•

Inter process libary CaffeNet_Mutex

Dopo la breve introduzione mirata prodotta nelle pagine precedenti, è possibile comprendere meglio la descrizione del Server Mk6 e cogliere il senso delle soluzioni utilizzate. Si rimanda alla letteratura citata nei riferimenti per una completa spiegazione dell'inter process comunication.

Per il server Mk6 si è definita una ulteriore libreria, con lo scopo di gestire l'accesso alle risorse condivise, utilizzata nell'IPC al fine di evitare race-condition sulla scrittura della memoria.

La libreria si basa su gruppi di semafori ottenuti e gestiti con sem.h; si è proceduto a rendere l'approccio alla gestione delle risorse più user friendly, inserendo un limitato numero di funzione in grado di svolgere autonomamente tutti i passaggi richiesti.

- *int Caffe_MutexCreate(key_t ipc_key, int nsems)* funzione che crea un gruppo di nsems semafori e grazie a *semget()* lo inizializza, per garantire unicità del semaforo viene utilizzato *ipc_key* come identificativo unico
- int Caffe_MutexFind(key_t ipc_key) permette di ottenere il mutex ID fornito l'identificativo *ipc_key*
- int Caffe_MutexRead(int sem_id, int semnum) dato mutex ID del gruppo semaforo permette di ottenere lo stato del mutex, ovvero se la risorsa sia disponibile in Read/Write oppure occupata
- int Caffe_MutexLock(int sem_id, unsigned short int semnum) mette in stato “lock” un semaforo, ovvero dichiara inaccessibile la risorsa condivisa assoggettata
- int Caffe_MutexUnlock(int sem_id, unsigned short int semnum) mette in stato “unlock” un semaforo, ovvero dichiara accessibile la risorsa condivisa assoggettata
- int Caffe_MutexRemove(int sem_id, int semnum): rimuove il semaforo specificato da nsem del gruppo sem_id inserito.

Procedura avanzata di riconoscimento dello stato della struttura dati

Come annotato nelle conclusioni per il server MK1 la condizione per cui una struttura, al sopraggiungere del timeout, venga scartata a prescindere dai dati presenti in essa è fortemente inefficiente, di conseguenza è necessario implementare un metodo che sia un compromesso fra la necessità di certezza sull'affidabilità dei dati e quella di avvalersi del maggior numero possibile di informazioni utili, ovvero si necessita di una funzione che permetta di scartare, dopo la condizione di time out, le sole strutture che siano realmente inutilizzabili ai fini della ricostruzione dello stato del velivolo, quindi il problema diviene definire come valutare l'utilità / inutilità di una struttura incompleta.

Si osserva il problema dal punto di vista della ricostruzione della condizione di volo e dello stato velivolo, sarebbe da scartare ogni struttura completamente inutile, definendo così una struttura che non sia in grado di apportare nuove informazioni sull'istante corrente.

Per non apportare informazioni una struttura deve essere priva di uno dei dati fondamentali utilizzati per ricavare la maggior parte delle grandezze correlate.

A titolo di esempio non esaustivo, se ad una struttura *Gps_data* mancasse un dato di posizione ecef (ad esempio x_gps) non sarebbe comunque possibile ricostruire la posizione del velivolo e sarebbe inutile inoltrare la struttura ai terminali remoti.

Questo tipo di ragionamento comporta di valutare tre attributi:

- **Presente** con questo attributo viene indicato un dato che è previsto sia realmente acquisito dalla attuale configurazione dei sensori, ovvero questo indice è pensato per considerare che alcuni parametri pur essendo attualmente implementati non sono ancora acquisiti da Mnemosine, e dunque queste aree di memoria non sarebbero mai riempite con le impostazioni in essere.
- **Fondamentale** con questo attributo viene marcato un dato “presente” la cui mancanza comporta la reiezione della struttura durante le verifiche successive alla condizione di timeout
- **Trascurabile** con questo attributo viene marcato un dato “presente” la cui mancanza non comporta la reiezione della struttura durante le verifiche successive alla condizione di timeout

principio di funzionamento della procedura avanzata di riconoscimento dello stato della struttura dati

La tecnica implemenata prevede che si assegni a ogni parametro “presente” un valore “trascurabile” o uno “fondamentale” che possano variare in 0 e 1 per ogni condizione di verifica prevista.

La determinazione di questi attributi avviene tramite delle regole fornite attraverso un file di configurazione, con estensione cfg, chiamato caffenet_server.cfg che indichi i parametri per cui una variabile sia “presente”, e sotto quali regole sia da considerare “trascurabile” o “fondamentale”.

Il file cfg avrà la seguente struttura:

```
GPS_DATA :{
  timeout=600; # millisec
  ecef_pos_x_3 = 1;
  LLH_pos_x_3 = 0;
  speed = -1;
  v_speed = 1;
  fix_qual = 2;
  sat_numb = 2;
  DOP = 0;
  ecef_vel_x_3 = 0;
};
```

dove per ogni gruppo viene definito il valore di timeout in millisecondi, e a ogni variabile o gruppo di variabili, di ogni struttura può essere assegnato un numero intero.

Assegnare un valore negativo significa che il dato non è “presente” e di conseguenza si pone a 0 l'attributo “presente” in fase di verifica, facendo si che questo parametro non concorra a valutare lo stato di riempimento della struttura.

Assegnare il valore 0 ad un parametro nel file di configurazione equivale ad indicare che il dato è sempre “trascurabile”dopo la condizione di timeout, ovvero si assegna 0 all'attributo “fondamentale”, questo implica che al sopraggiungere del timeout i parametri contrassegnati con 0 nel file caffenet_server.cfg non vengano valutati come essenziali e una struttura possa essere promossa per l'invio anche in loro assenza.

La assegnazione di un numero positivo ad un parametro del file caffenet_server.cfg definisce una condizione di dato “fondamentale” che attribuisce il valore 1 all'attributo “fondamentale” per la regola espressa dal valore positivo assegnato, ovvero al sopraggiungere del timeout si innesca la verifica che siano presenti tutti i dati aventi stesso indice.

Il caso della configurazione sopra riportata equivale a “il parametro speed e non è rilevato, in caso di timeout verificare che ecef_pos_x_3 e v_speed siano presenti oppure che fix_qual e sat_numb siano presenti, se una delle due è vera procedere all'invio”

Durante la fase di verifica viene chiamata la funzione

*int isfilled(str_w_time * s_w_t , double * time, shared * sh_ptr)*

questa restituisce 1 in caso di struttura pronta all'invio o 0 in caso contrario, richiede come input *str_w_time * s_w_t* ovvero la struttura da analizzare corredata dall'informazione di quando sia

avvenuta la scrittura del primo dato, il tempo attuale *double * time*, e l'area di memoria condivisa (IPC), che contiene una serie di array che sintetizzano gli attributi assegnati ai parametri Fondamentale e Presente nel file di configurazione.

La funzione *is_filled* innanzitutto verifica lo stato di timeout della struttura, se non è scaduto, imposta la condizione di verifica in cui tutti i valori “presente” siano fondamentali, se invece riscontrasse il Timeout, effettua numerose comparazioni per verificare l'esistenza di almeno una delle condizioni espresse dai parametri “fondamentale” descritti nel file caffenet_server.cfg.

Questo permette di definire strategie complesse di reiezione delle strutture che siano parzialmente piene al sopraggiungere della condizione di “tempo scaduto”, e di modificare velocemente le regole di reiezione al mutarsi delle condizioni del sistema, si pensi ad esempio ad un guasto che blocchi il campionamento di un determinato parametro.

Server Mk 6

Come già anticipato il server Mk6 è il massimo sviluppo alla ricerca di migliori prestazioni da conseguire su una macchina molto potente con almeno 8 core che permettano di sviluppare 6 thread contemporanei.

Viene inserito un buffer di strutture che possa limitare i problemi di ritardi negli invii che si erano manifestati nel server Mk1.

Viene scelta l'allocazione statica in memoria condivisa così da rendere il buffer un mero array di puntatori a strutture, senza la necessità di effettuare onerose copie della struttura per il trasferimento fra processi.

Prevede 8 thread di cui sei che lavorino in parallelo, oltre il processo main che gestisce lo spawn e il coordinamento iniziale dei processi e aspetta in stato idle di coordinare con apposite le chiamate la chiusura del programma.

All'avvio del server il processo main si occupa dell'inizializzazione di tutte le risorse necessarie, allocare la memoria, definire gli strumenti IPC, e avviare i tre processi figli, chiamati main_mamory, main_filling, main_send.

- main_mamory si occupa di gestire il contenuto della memoria condivisa, del buffer di strutture preparate per essere inviate e della ri-inizializzazione delle stesse una volta inviati i dati.
- main_filling si occupa della fase dell'inserimento dei dati nelle apposite strutture, della verifica dello stato della struttura e dell'eventuale conversione da host a Network endiannes
- main_send Si occupa della creazione del socket UDP e della lettura delle strutture presenti sulla coda/buffer di strutture presenti nella memoria condivisa e procede all'invio delle strutture ricevute (immediatamente se i dati sono ricevuti in Live, o al momento corretto in caso di lettura dei dati da file di backup)

Main_mamory sfrutta tre thread, il primo legato al processo stesso che si occupa della gestione della memoria condivisa, di autorizzare gli accessi alla stessa, e gestire il buffer di strutture in attesa del popolamento. Il secondo chiamato *th_str_init* si occupa di inizializzare le strutture quando necessario, all'avvio, e ogni volta che una struttura sia stata spedita o il cui contenuto sia obsoleto e debba essere cancellato. Il terzo thread chiamato *th_free_insert* gestisce le fasi di transitorio in cui una struttura sia stata appena liberata e vada inserita nel buffer pronta per essere riempita nuovamente, avvisa gli altri processi quante siano disponibili per il riempimento e quante siano in attesa di essere spedite, ed eventualmente mette in pausa il processo di filling se le strutture vuote fossero esaurite.

Main_filling sfrutta anche esso 3 thread, il primo direttamente legato al processo si occupa del'inserimento e della ingegnerizzazione dei dati, il secondo thread th_is_filled, dopo aver verificato che una struttura non sia soggetta a scrittura ne acquisisce il mutex_lock e provvede a scansionarla con la funzione is_filled ed eventualmente ne passa il puntatore al thread th_hton che procede alla conversione di endiannes fra host e network

Main_send sfrutta un solo processo e si occupa della ricostruzione realistica della successione dei dati (e del coordinamento dei tempi se si è in modalità palyback)

La successione di operazioni eseguite dal server in scala temporale è rappresentabile graficamente come da 29

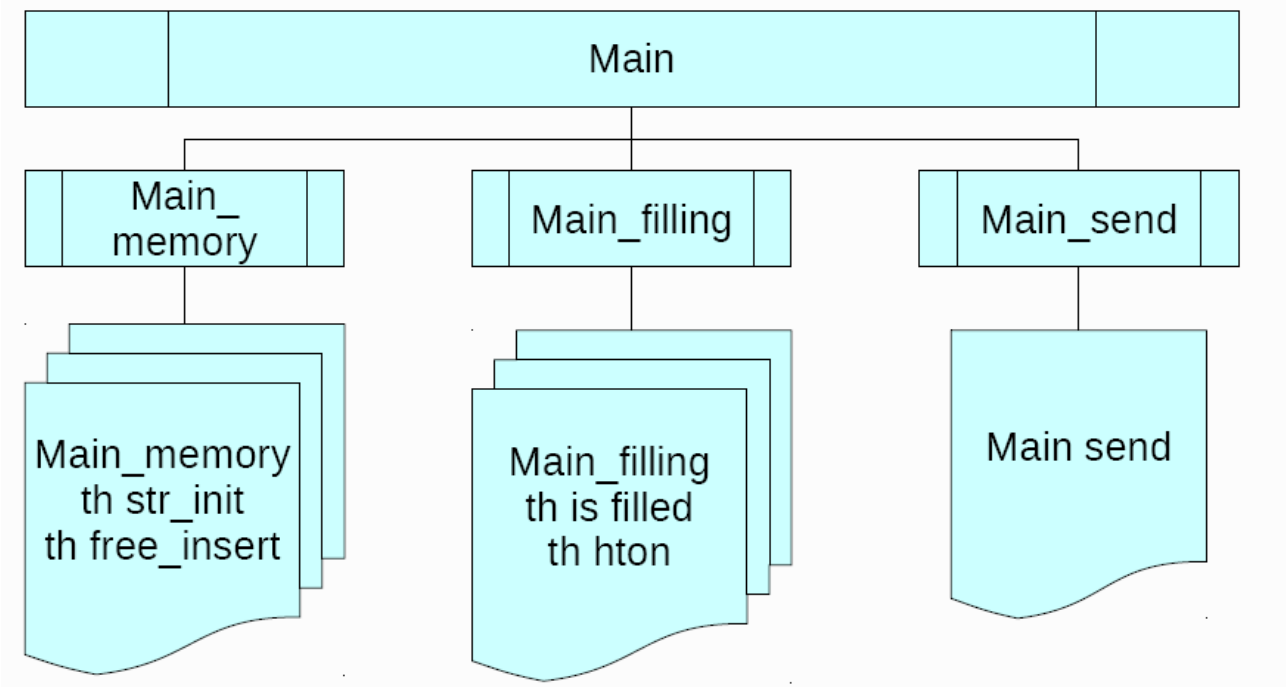

Figura 28: rappresentazione grafica della struttura in processi e thread del server Mk6

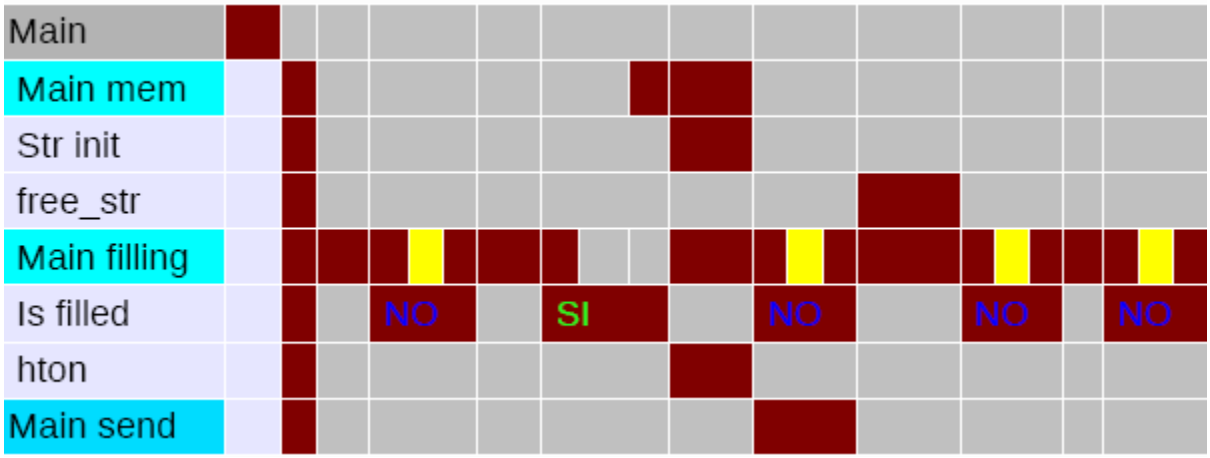

Figura 29: rappresentazione grafica del funzionamento dell'attività dei singoli thread su scala temporale

considerazione su server da Mk2 a Mk6

Lo studio di ottimizzazione per macchine ad alte prestazioni hanno portato a risultati più che accettabili, ottenuti su una macchina quadcore che non poteva dunque sfruttare a pieno l'architettura multitasking del programma.

I problemi delle latenze sono quasi del tutto scomparsi grazie all'introduzione e successivo affinamento del buffer di strutture pronte all'invio.

La ricerca di prestazioni ha comportato un notevole aumento della complessità del codice rendendolo, meno soddisfacente sotto il profilo della manutenibilità e della sviluppabilità futura.

Questi fatti uniti alla non immediata disponibilità di un PC dotato di un processore 8core hanno spinto a considerare lo sviluppo di una versione del CaffeNet_Server, che puntasse ad una semplificazione dell'interazione fra processi e thread e cercando contemporaneamente di penalizzare il meno possibile le prestazioni.

3.8.4 Server MK7

Come anticipato il server Mk7 viene implementato in modo da rendere il più semplice possibile una futura manutenzione del codice e facilitare l'adeguamento del sw a cambi radicali nel sistema di acquisizione dati senza dimenticare però di perseguire, nei limiti della ragionevolezza, adeguate prestazioni nella velocità di esecuzione.

Il server Mk7 si basa su 2 soli processi, in modo da poter sfruttare al meglio un pc dual core di uso comune.

Oltre ai due processi concorrenti viene implementato un main che provvede a creare e coordinare i figli attendendo in stato idle l'apposita chiamata per coordinarne l'esecuzione della procedura di chiusura.

I due processi, denominati filling_ child, send_child, non utilizzano una memoria condivisa ma solo una pipe per dialogare.

Il filling_child provvede a tutte le fasi atte al riempimento delle strutture dati e “spedisce” quelle piene al send_child, il quale leggerà la pipe in modo non Bloccante, acquisendo i dati forniti da filling_child, provvederà a coordinare i tempi di spedizione delle strutture ed in caso non sia ancora il momento di effettuare la spedizione via socket UDP, verificherà di nuovo se vi siano altri dati da leggere in un loop che terminerà solo quando verrà chiusa la pipe.

I 2 buffer, quello delle strutture in attesa di essere riempite e quello delle strutture in attesa di essere spedite, sono implementati sotto forma di due array di puntatori, il primo allocato staticamente nello stack del processo *filling_child* e il secondo nello stack del processo send_child.

filling_child provvede allo spacchettamento e all' ingegnerizzazione dei dati, all'inserimento degli stessi nella struttura di appartenenza, e alla verifica dello stato di riempimento della stessa. Successivamente questo processo valuta l'eventuale promozione all'invio della struttura ed eventualmente la spedisce attraverso la pipe al send_child, infine procede alla ri-inizializzazione della memoria dove era allocata la struttura da spedire e provvede alla gestione del buffer delle strutture dati vuote.

send_child provvede a poche semplici operazioni, accerta la presenza di dati sulla pipe, ed eventualmente li acquisisce, successivamente verifica se fra le strutture già ricevute vi sia una struttura che abbia raggiunto il momento dell'invio e la invia, inoltre send_child gestisce il buffer delle strutture in attesa di essere inviate. Nella maggior parte dei casi questo processo svolge il loop in pochissimi millesimi di secondo, leggendo in modo non bloccante la pipe e verificando che manchino ancora alcuni millisecondi all'invio della prima struttura, torna a leggere la pipe.

Il metodo di comunicazione via pipe dell'intera union, (struttura dati corredata da un double in cui è inserito il tempo del primo dato ricevuto), implica che questa abbia attributo di compilazione packet *__attribute__((packed))* mutuando le precedenti considerazioni sull'ottimizzazione del codice già introdotte nel paragrafo 3.7.1.

Il codice del server Mk7 implementa tutte le librerie descritte in precedenza, eccezion fatta per CaffeNet_mutex, che risulta inutile non intervenendo nell'esecuzione risorse condivise che possano incorrere in accessi simultanei. Mk7 accoglie anche una perfetta copia della funzione isfilled() utilizzata per il server Mk6 introducendo quindi gli stessi benefici per la gestione delle eccezioni dovute a timeout di riempimento.

3.8.5 Esempio di flessibilità: Sviluppo del codice per MK_Sperimentazione_in_volo_2012

Pur non essendo ancora pronto l'hardware per la ricezione dei dati dal velivolo si è deciso di testare "sul campo" la versione Mk7 utilizzando i file di backup del sistema MNEMOSINE al fine di validarne i concetti fondamentali, e si è scelto di utilizzare i dati delle le prove del corso Sperimentazione in Volo del giugno 2012.

Le esigenze per le prove in questione erano limitate ad ottenere lo spacchettamento e l'ingegnerizzazione dei dati registrati a bordo e il successivo salvataggio in file CSV (Comma Separated Value) con una suddivisione per tipo di dato, e non in strutture come nei precedenti server. Si è quindi approntato un nuovo codice basato sul filling_child, eliminando tutte le funzioni del Send_child ed infine inserendo il salvataggio dei dati su Hd (creando di fatto uno studio avanzato per il futuro sviluppo del Client di memorizzazione dati di cui si è già accennato).

Queste modifiche hanno portato ad un programma in grado di soddisfare adeguatamente i requisiti di velocità, in quanto in meno di due minuti è possibile analizzare un volo medio di 30 minuti, contenente circa 2 milioni di comunicazioni CAN utili, spacchettarne i datagrammi contenuti, ingegnerizzare i dati e suddividerli per poi salvarli in un file CSV per ogni parametro.

Il programma Mk_voli_2012 è anche programmato per creare, al termine delle operazioni, un file di testo chiamato readme_caffenet2012.log che oltre a contenere un breve log che riporti:

- la versione del programma,
- la data e l'ora di esecuzione,
- eventuali errori riscontrati nell'esecuzione,
- una leggenda che descrive il contenuto di ogni uno dei 50 file CS creati, e per ognuno di essi specifichi l'unità di misura utilizzata per descrivere il parametro contenuto.

La conversione del codice dal server Mk7 a Mk_voli_2012 ha richiesto, circa 10 giorni, compresa la messa a punto delle curve di calibrazione.

Il breve tempo di conversione è stato raggiungibile grazie all'architettura "modulare" data alle librerie che sono state riutilizzate senza alcuna modifica.

Successivamente è stato implementato un file di comandi shell (automazione_Mk_voli_2012.sh) al fine di coordinare MK_Sperimentazione_in_volo_2012 con un programma di compressione dati e l'applicazione proprietaria del claud Dropbox, così che si automatizzasse la seguente procedura:

- creazione dei file CSV di ognuno dei 16 voli,
- compressione di tutti i file CSV relativi allo stesso volo
- upload sul Claud Dropbox dei pacchetti di dati compressi

L'intera procedura eseguita su 16 voli richiede 42 minuti per l'elaborazione, mentre il tempo di caricamento è risultato fortemente dipendente della banda concessa dal server Dropbox.

3.8.6 Client per test e validazione via Wireshark

Per la validazione del funzionamento dei server si sono utilizzate due metodologie la cattura di pacchetti con Wireshark, e la realizzazione di un Client che spacchettasse le strutture e mettesse a display una sola struttura per tipo, confrontando i dati ottenuti con quelli pre-memorizzati sia nel Client Test sia nella funzione Test dei server in modo da avere immediata verifica della correttezza di funzionamento di tutto il processo di elaborazione dati.

Nella scelta di realizzare un Client per i test, si è considerato come questa implementazione potesse fornire una solida base per lo sviluppo del client per la recezione dei dati su terminali remoti in attesa di ottenere le interfacce software da utilizzare per dialogare con applicazioni grafiche.

Il funzionamento del client è il seguente:

1) allocazione di tutte le variabili
2) inizializzazione della struttura sockaddr
3) chiamata a getaddrinfo
4) apertura socket
5) bind del socket
6) lettura (bloccante) con recvfrom
7) spacchettamento e conversione in client endianess della struttura ricevuta
8) verifica della sequenzialità totale e per tipo di struttura eventuale messaggi di errore
9) memorizzazione delle strutture ripetizione dal punto 6 per 500 volte
10) scelta di 4 strutture per tipo, le prime dopo la 50-esima, 150-esima, 250-esima, 350-esima struttura ricevuta
11) display a schermo del report e del confronto fra dati attesi e dati ottenuti per ogni parametro di ogni struttura

Per la verifica dei dati durante il transito sulla rete ed effettuare un test di corretta formattazione del traffico si è utilizzato Wireshark un software per l'analisi delle reti che permette di verificare e ispezionare ogni singolo pacchetto in transito sulla rete. I pacchetti così catturati risultano molto ostici da interpretare in quanto i dati contenuti nelle strutture sono in network endianess e appaiono come una sequenza di numeri esadecimali senza soluzione di continuità per tutta la lunghezza del pacchetto, questo tipo di verifica è utile per verificare solo i parametri dell'header del pacchetto UDP che sarebbe impossibile ottenere con il client_test.

4 Hardware per l'acquisizione di posizione e pressione atmosferica

In questo capitolo si affronterà lo sviluppo dell'hardware deputato alla raccolta dei dati riguardanti la posizione e la pressione atmosferica rilevati dal sistema GS e successivamente verrà trattata l'integrazione delle misurazioni così ottenute, al fine di stimare più velocemente e accuratamente la quota rilevata ed estendere la correzione ottenuta alla pressione rilevata dal barometro.

La fusione dei dati fra sensori differenti comporta lo scambio di informazioni fra i due strumenti allo scopo per correggere le misurazioni di uno con le informazioni estrapolate dalle misure effettuate dall'altro, tale tecnica è possibile solo se le due grandezze rilevate hanno fra loro una relazione.

L'hardware utilizzato per l'acquisizione consiste in 3 moduli adeguatamente connessi fra loro,

- un gps U-blox (o µ-blox) ANTARIS®4GPS Modules
- un sensore di pressione HCA-BARO 5006 Series
- un modulo per connessione USB Serial-UART/I2c FT4232H Mini Module .

Nelle prossime pagine si procederà ad una sommaria descrizione degli elementi sopra elencati.

4.1.1 ANTARIS®4 GPS Modules

È un modulo GPS basato sul chip Ublox LEA 4H, è dotato di eprom per la memorizzazione dei parametri utili al funzionamento (efemeridi, frequenze di aggiornamento, protocollo prescelto, modalità di filtraggio) e implementa già al suo interno molti filtri numerici specifici per ogni condizione di utilizzo, più una impostazione che si presume fornisca dati non filtrati chiamata static fix.

I protocolli utilizzati dal modulo per la trasmissione dati sono Nmea e UbX (protocollo binario proprietario di U-blox), che utilizzano 2 porte USART o Serial UART, con baud-rate da 4800 a 115200 bit/sec

Dato che i comuni PC, e mini ITX non implementano di default porte per connessione USART si necessita quindi di un modulo in grado di instradare le comunicazioni USART in una porta seriale virtuale del PC, sfruttando una connessione USB.

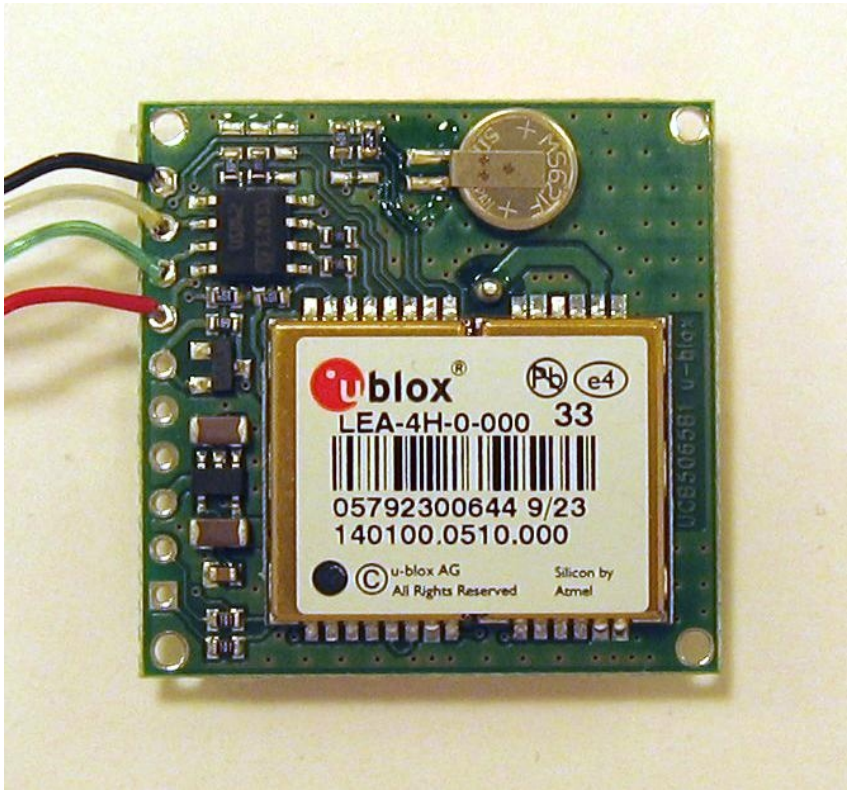

Figura 30: un modulo GPS utilizzante il chip di Antaris 4

4.1.2 HCA-BARO HCA0611AR serie 5006

HCA-BARO HCA0611AR E' un modulo che svolge il compito di un barometro digitale ed è in grado di generare un'uscita su bus I2C che indica la pressione misurata dal sensore. Il range di funzionamento è 600-1100 mBar con sensibilità 0.015 mbar (1,5 Pa), quindi appositamente pensato per la misurazione ad alta risoluzione di pressioni atmosferiche.

Il sensore esegue ciclicamente, con frequenza di 1000 Hz, campionamento, conversione, e memorizzazione del valore di pressione in un registro a 15 bit.

Il dato di pressione è digitalizzato con valori fra 1 (600 mBar) e 32767 (1100 mBar) La comunicazione del counter avviene attraverso bus I2C bidirezionale, in cui il barometro è un terminale slave con indirizzo fisso (1111000xb) ogni volta che questo nodo riceve al proprio indirizzo di rete la richiesta di invio dati risponde con i 15 bit memorizzati nel registro che indicano il valore letto durante l'ultimo ciclo di misurazione della pressione.

Figura 31: Barometro HCA0611AR

4.1.3 FT4232H Mini module, USB Serial-UART/I2c

E' un modulo dotato di quattro uscite che possono essere configurate come interfacce FIFO seriali o parallele sincrone o asincrone.
Due di queste hanno la possibilità di essere configurate per integrare il funzionamento di un motore MPSSE, ovvero Multi-Protocol Synchronous Serial Engine che fornisce un mezzo flessibile per interfacciare dispositivi a connessione seriale sincrona con una porta USB ovvero proprio ciò che è necessario ai nostri scopi per connettere il modulo HCA0611AR ad un comune PC.
Si desume dal manuale operativo del modulo che questo è stato progettato per permettere al chip FT4232H di operare nel pieno delle sue funzionalità, ovvero come due porte UART / Bit-Bang, e contemporaneamente come due motori MPSSE utilizzati per emulare JTAG, SPI, I2C, bit-bang o ogni altra modalità seriale sincrona. Questo consentirebbe di utilizzare il FT4232H per connettere allo stesso tempo anche il modulo Antaris 4 al PC (e altri 2 moduli se necessario).
Il Mini Modulo FT4232H è una scheda elettronica che provvede all'alimentazione del chip e collega i contatti del core FT4232H_IC a due connettori a doppia fila 26-PIN.
FT4232H Mini Module necessita di driver proprietari per la periferica USB, e dunque per la comunicazione PC ↔ embedded viene installata l'apposita libreria.

Figura 32: 3.1.3 FT4232H Mini module

4.2 Il sistema completo

I 3 moduli, come già accennato, sono connessi fra loro con 2 diversi tipi di bus, uno I2C e uno USART, e presentano seguente schema di connessione:

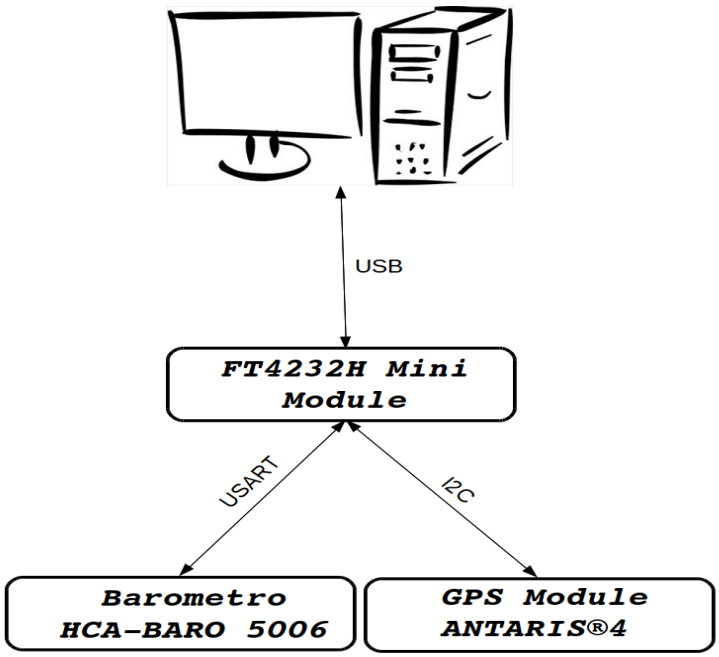

Figura 33: schema dei bus utilizzati per la connessione dei Moduli

I dati utilizzeranno dunque due “canali di comunicazione” diversi fra il sensore ed il PC, i dati barometrici viaggeranno secondo lo standard di trasmissione I2C fino al modulo FT4232H e successivamente verranno letti mediante le librerie proprietarie FTDI Chip, mentre i dati GPS viaggeranno attraverso USART fino al modulo di connessione e poi verranno inviati al pc come se il modulo fosse una periferica Seriale virtuale, questa connessione trasporterà i dati GPS“formattati” secondo il protocollo prescelto sia questo UBX o Nmea.

4.3 Protocolli Seriali

Per garantire una miglior comprensione degli argomenti trattati in seguito, nei prossimi paragrafi verranno illustrati i protocolli studiati per lo sviluppo del sistema.

4.3.1 I2C

O I²C, (Inter Integrated Circuit), è un sistema di comunicazione seriale sincrona bipolare utilizzato tra circuiti integrati.

Il bus I2C è composto da almeno un master che arbitra il bus ed uno o più slave.

Questo tipo di bus fu ideato dalla Philips nel 1982 e nel 1992 è stata rilasciata la prima versione non commerciale del protocollo che ha subito poi diversi aggiornamenti ed ha generato bus simili, che hanno funzionalità identica a I2C ma piccolissime differenze che permettono la protezione a fini puramente commerciali del protocollo.

Come già accennato l'hardware per il bus I2C prevede una duplice connessione a cavi twistati che raggiunga tutti i device connessi, i due cavi sono comunemente denominati:

SDA (Serial DAta line) per il trasporto dei dati

SCL (Serial Clock Line) per la propagazione del segnale di clock

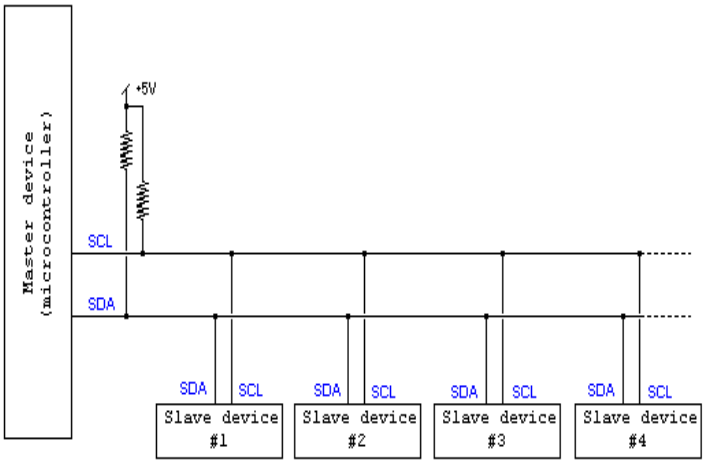

Figura 34: esempio circuito I2C (a 5V)

Alle connessioni già indicate va aggiunta una linea di alimentazione, Vdd, a cui sono connessi i resistori di pull-up.

I nodi di un bus I2C utilizzano una comunicazione con indirizzi a 7 bit e supportano fino a 127 componenti, alcuni moduli estesi utilizzano indirizzi a 10 bit per gestirne fino a 300. Le velocità standard di comunicazione si aggirano fra i 10kbps e i 3.4Mbps nella modalità Hi-speed.

La sequenza di comunicazione inizia con il master che invia un bit di start sulla line SDA seguito dall'indirizzo dello slave a 7 bit e infine un bit che seleziona l'operazione: 1 per leggere e 0 per scrivere.

Nel caso in esame per la richiesta di dati dal barometro verrà scritto sul bus I2C:

1 1111000 0

A questo punto, lo slave, cioè il barometro, invierà un bit singolo al master con la funzione di acknowledgment.

Successivamente verranno inviati i dati di pressione sul bus .

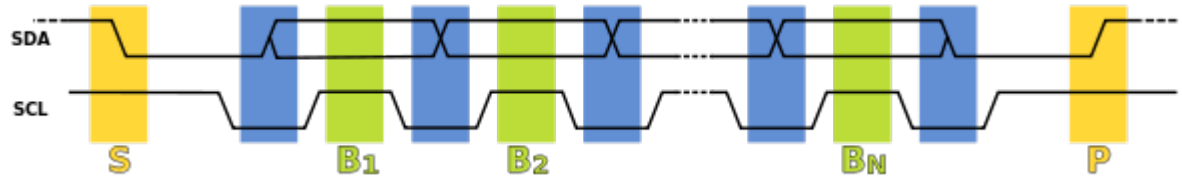

Figura 35: flusso di bit in un bus I2C in giallo i bit di start e stop, in blu il cambio di stato nei bit di dati in verde il momento di sample dove viene letto il valore dall'apparato ricevente

4.3.2 USART (Universal Synchronous-Asynchronous Receiver/Transmitter)

USART "ricevitore trasmettitore sincrono asincrono universale" è un dispositivo hardware, utilizzato per la comunicazione on-field. Converte flussi di bit di dati da un formato parallelo ad un seriale asincrono o viceversa. Praticamente ogni famiglia di microprocessori ha la sua UART/USART dedicata.
I microcontrollori che supportano questa funzione spesso delegano tutte le operazioni ad un apposito hardware integrato mentre nei moderni Personal computer, le porte USART hanno il compito di gestire le comunicazioni delle interfacce seriali RS-232 che utilizzano 12 V.
USART costituisce una evoluzione di UART ed è in grado di gestire anche trasmissioni seriali sincrone, come si evince dal nome.

4.3.3 Nmea

NMEA è il protocollo standard di comunicazione dati più utilizzato in nautica e nella comunicazione di dati satellitari GPS.

L'ente che gestisce e sviluppa il protocollo è la National Marine Electronics Association.

Questo protocollo si basa sul principio che la fonte, detta talker, può soltanto inviare i dati (sentences) e la ricevente, detta listener, che può soltanto riceverli, mentre altri protocolli proprietari (ad esempio UBX) utilizzati dai più moderni apparati GPS permettono la comunicazione in entrambi i sensi permettendo uno scambio di informazioni più efficace, e anche memorizzare

parametri di configurazione direttamente sulla eprom del dispositivo.

La frase Nmea ha un formato fisso nella composizione ma non nella lunghezza:

*$PREFISSO,dato1,dato2 ... datoN-1,datoN*CHECKSUM*

ogni trasmissione Nmea è effettuata in caratteri Ascii e inizia con il carattere $ seguito da un

PREFISSO cioè una sequenza di 5 caratteri che designa il tipo di dispositivo che trasmette (GP per ricevitore GPS) seguito dal tipo / dai tipi di dati a seguire ad esempio $GPGLL indica un GPS che invia dati di tipo “posizione LLH” successivamente viene trasmesso

dato1,dato2 ... *datoN-1,datoN* che sono il payload ad esempio per $GPGLL ci attenderemo

Nord o Sud longitudine Est o ovest tempo UTC stato di validità della connessione infine verrà inviato il
CHECKSUM ovvero il bit di parità con lo scopo di verificare la corretta ricezione della sentence.

4.4 Test del Sistema

L'intero schema delle connessioni del sistema, è stato inizialmente riprogettato per effettuare un test cieco sulla precedente implementazione successivamente si è testato l'HW con varie tecniche al fine di escludere la mancanza di continuità elettrica o l'involontaria connessione di una o più tracce e si sono svolti test di acquisizione dei dati dal Modulo GPS, attraverso il componente FT4232H, utilizzando i driver proprietari di FTDIchip, e il programma U-center fornito da U-blox.

Tutte le prove precedenti non hanno evidenziato difetti nel sistema.

Infine si è proceduto a testare l'acquisizione dei dati dal barometro con il motore MPSSE in configurazione I2C.

4.5 studio dell'integrazione GPS-Barometro

L'hardware è progettato per fornire al pc due dati distinti, uno di pressione (da cui è possibile calcolare la quota) e uno di quota sul livello del mare.

Si è interessati ad avere una integrazione dei dati forniti dai due sistemi al fine di ottenere una misurazione più accurata della quota, con un risultato che sia attendibile possibile sin dai primi istanti.

L'Integrazione “Multi Sensors Data Fusion” indica la teoria, le tecniche e gli strumenti che sono utilizzati per combinare misurazioni effettuate da diversi sensori allo scopo di migliorare la qualità delle informazioni, ovvero in modo da ottenere una stima della grandezza osservata, che sia più precisa di quanto sarebbe possibile se le fonti dei dati fossero utilizzate singolarmente.

Per fare ciò è necessario fornire al metodo di datafusion tutte le informazioni, che i singoli sensori tendono a ignorare o non poter sfruttare.

Pur avendo la possibilità di campionare dati barometrici con frequenza maggiore, si decide di limitarla a 4Hz per evitare l'appesantimento del problema di fusione dovendo applicare la teoria collegata al campionamento con frequenze diverse delle misure che comporterebbe modelli di integrazione con maggiore complessità, come dimostrato da [50].

Quest'ultima semplificazione è resa possibile dall'ipotesi che tutte le dinamiche di interesse abbiano frequenza inferiore a 4 Hz, sicché è plausibile ritenere che tale data-rate non infici la bontà dei risultati.

Le fasi dello studio di integrazione sono state:

1) studio del rumore di misura del sistema GPS
2) studio del rumore di misura del barometro
3) implementazione di alcuni metodi di datafusion

4.5.1 studio del rumore di misura dell'apparato GPS

Come già introdotto il modulo GPS, con l'attuale configurazione, fornisce dati ad una frequenza di campionamento di 4Hz.

Il modulo Antaris 4 viene settato con impostazione “static fix → raw data” come descritto da [51], questo permette di ottenere dati non filtrati, che escludano quindi rumori con dinamiche colorate dovute al prefiltraggio EKF (*Extended Kalman Filter)* o IEKF (*Invariant Extended Kalman Filter)* implementato direttamente nel modulo.

Se la modalità static fix dovesse permettere di bypassare i filtri, dovrebbe essere possibile ottenere un andamento dell'errore descrivibile da un processo a rumore bianco, ovvero la misura dovrebbe essere caratterizzata da funzione di Auto-correlazione con un picco per l'istante di misurazione e una andamento minore per i campioni antecedenti.

L'importanza della verifica di bianchezza risiede nel non voler trascurare nessuna informazione durante la fusione dei due segnali, infatti un rumore colorato è indice di dinamiche non catturate dal modello di misurazione, che venendo omesse in fase di fusione dei dati porterebbero ad un risultato non ottimale, difatti si trascurerebbero informazioni utili per fornire una migliore ricostruzione dell'andamento della variabile osservata.

Si effettua un test di bianchezza con metodi spettrali, partendo da un campionamento statico della durata di 24 ore, eseguito in luogo aperto privo di ostacoli e con un cono di visione celeste maggiore di 165°, quindi rispettando ampiamente le condizioni ottime di funzionamento previste da [51].

Si effettua una analisi spettrale con autocorrelazione, e fft, così da verificare l'effettiva bianchezza del disturbo.

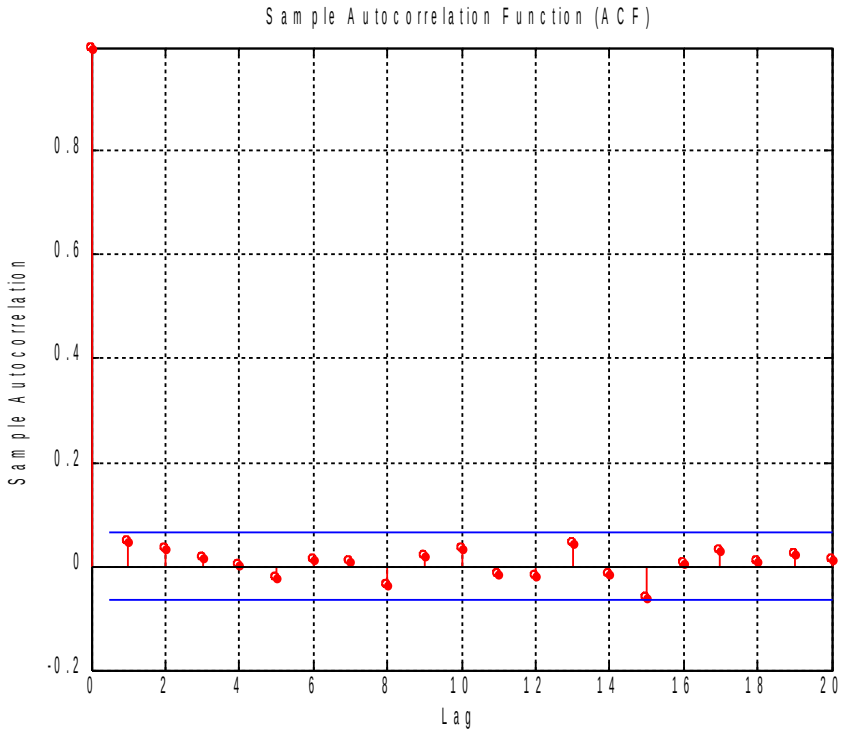

Figura 36: autocorrelazione dei dati GPS

Il rumore di misura appare certamente bianco infatti l' autocorrelazione mostra come, per ogni istante precedente all'attuale, il dato ricada nell'intervallo di confidenza del 90% di probabilità.

Dall'analisi con la Fast Fourier Transformations dell'errore si nota che non vi è la presenza di picchi importanti (confrontati con il segnale di potenza 9 10^6) che determinino una ricorrenza nei dati

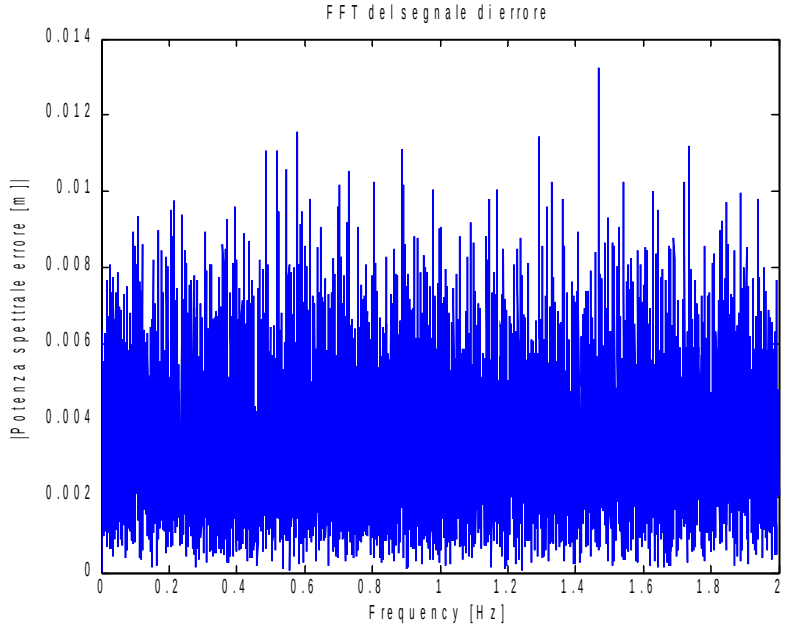

Figura 37: potenza del segnale del solo errore

4.5.2 studio del rumore di misura del Barometro

Si vuole svolgere un test analogo al precedente sul barometro in condizioni ottime per lo strumento per effettuare un test di bianchezza sul rumore di misura.

In attesa della disponibilità dell' HW, si opta per una valutazione teorica-semiempirica basata sulla letteratura disponibile.

Si utilizzano i due approcci come proposto da Bose [11] e Vivek Dadu[12]

Lo studio di Bose porta a considerare gli errori del barometro descrivibili come due processi di Gauss-Markov [14] del primo ordine uno legato alla scala dell'errore e uno legato all'errore di Bias, mentre Vivek Dadu in una analisi empirica di dati raccolti da un barometro digitale perviene ad un processo di Gauss-Markov di secondo ordine.

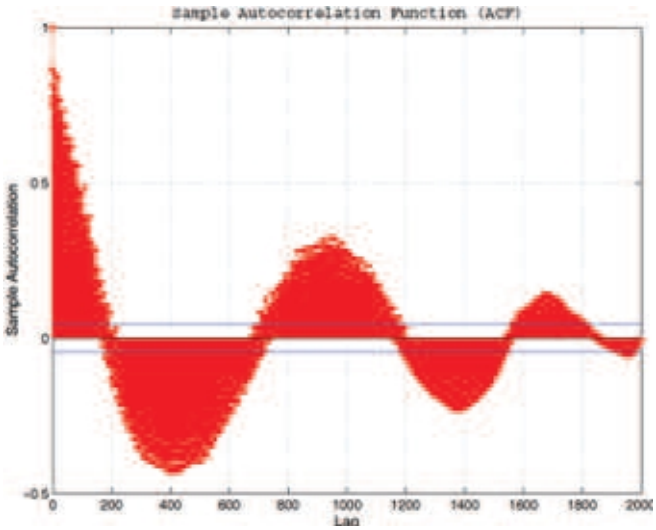

Figura 38: immagine riportata della Autocorrelazione riscontrata da Dadu in [12]

L'analisi degli errori proposta da Bose evidenzia come la non linearità di relazione, fra quota e variazione di pressione, porti a poter identificare un andamento del fattore di scala dell'errore di altezza, che non sia costante ma vari da un massimo positivo per quota nulla ad un massimo negativo per la quota massima rilevabile, linearizzando tale andamento si perviene alla:

$$k = k_0\left(1 - \frac{ht}{h_0}\right) \qquad \textit{Formula 4.1}$$

dove k è il fattore di scala dell'errore, k_0 è il fattore di scala rilevato a livello del mare, h_0 è il punto di azzeramento del fattore di scala, ht è l'altitudine corretta.

Di conseguenza si ottiene che l'errore di altezza rilevata dal barometro avrà andamento:

$$\delta h_b = \Delta_{h_b} + \int_0^{ht} K\, dh_t \qquad \textit{Formula 4.2}$$

dove Δ_{hb} indica l'errore di altitudine a quota zero, mentre gli altri simboli hanno il significato precedente.

Δ_{hb} e k_0 sono definiti come processi di Markov, ovvero modelli caratterizzati dal seguente schema:

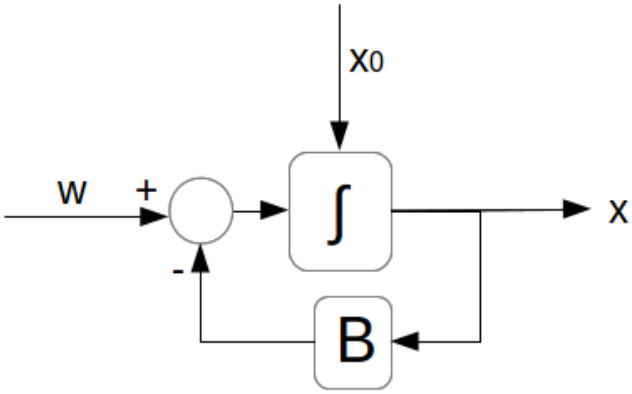

Figura 39: rappresentazione dell'errore secondo Bose

ottenendo uno schema dell'errore altimetrico come il seguente:

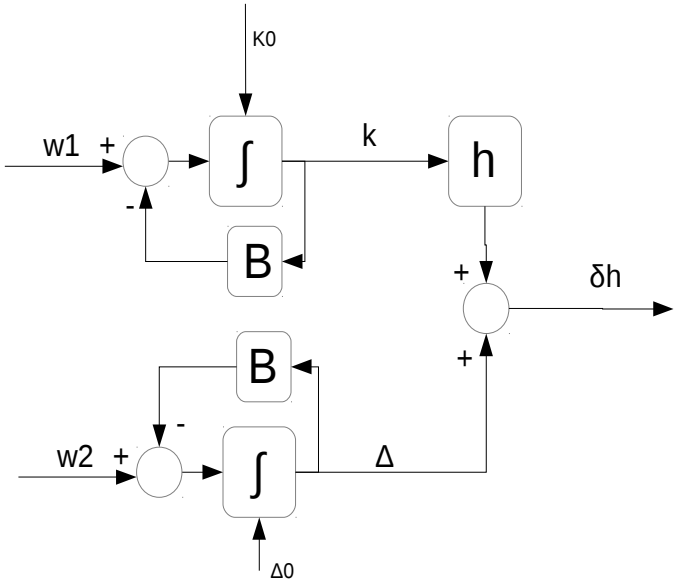

Figura 40 schema errore altimetrico formulato secondo Bose

Vivek Dadu invece perviene alla conclusione che il rumore abbia schema a processo di Markov di secondo ordine, come il seguente:

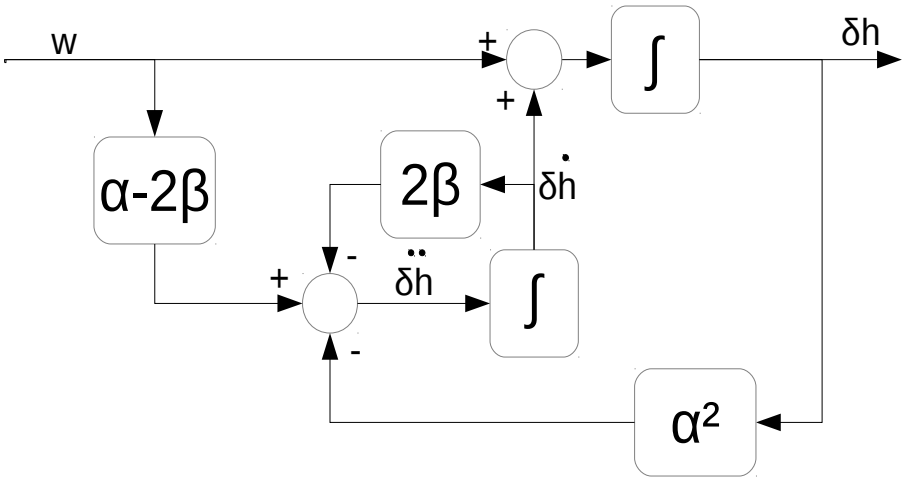

Figura 41: schema dell'errore altimetrico secondo Dadu

ovvero che l'errore abbia una formulazione

$$\begin{aligned} \delta\dot{h} &= \delta\dot{h} + w \\ \delta\ddot{h} &= -\alpha^2\delta h - 2\beta\delta\dot{h} + (\alpha - 2\beta)w \end{aligned} \qquad \textit{Formula 4.3}$$

che può essere scritta in forma matriciale come:

$$\begin{bmatrix} \delta\dot{h} \\ \delta\ddot{h} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\alpha^2 & -2\beta \end{bmatrix} \begin{bmatrix} \delta h \\ \delta\dot{h} \end{bmatrix} + \begin{bmatrix} 1 \\ \alpha - 2\beta \end{bmatrix} w \qquad \textit{Formula 4.4}$$

in questo caso, si nota come vi sia una cross relazione fra il rumore iniettato nelle due equazioni, mentre la formulazione di Bose prevedeva che fosse w1 ≠ w2.

Si procede a scrivere la forma discreta del sistema implementato da Dadu in modo da poter successivamente implementare più agevolmente un algoritmo di filtraggio dei dati.

$$\begin{bmatrix} \delta h \\ \delta\tilde{h} \end{bmatrix}_{k+1} = \begin{bmatrix} 0 & 1 \\ -\alpha^2\delta t & -2\beta\delta t \end{bmatrix} \begin{bmatrix} \delta h \\ \delta\tilde{h} \end{bmatrix}_k + \begin{bmatrix} 1 \\ \alpha - 2\beta \end{bmatrix} w \qquad \textit{Formula 4.5}$$

dove δt rappresenta il tempo fra i campionamenti (0.25 sec) e ovviamente i parametri α e β rimangono da determinare dai dati, utilizzando il metodo di massima verosomiglianza, come suggerito da Bittanti [13] e [14].

4.5.3 Metodi per il processo di data fusion

Dopo aver descritto compiutamente i modelli degli errori di misura attesi dai due sensori, è ora possibile affrontare la fusione dei dati.

Visto l'approccio teorico sostenuto per il rumore del barometro, la limitata letteratura in materia e i tipi differenti di Hardware utilizzati da Bose e Dadu per le rispettive prove, è possibile che in fase di reale acquisizione si pervenga a un errore differente che potrebbe anche essere confondibile con un processo a puro White Noise (Wn).

Se questo accadesse ci si troverebbe in presenza di due sensori con errore caratterizzato da Wn, un filtro semplice ma efficace sarebbe la media ponderata sulla varianza in caso si stia acquisendo un segnale statico o la media ponderata sulla varianza dell'errore per segnali che abbiano una connotazione dinamica.

Nelle seguenti sezioni si andranno ad analizzare alcuni metodi di fusione dei dati per evidenziarne le caratteristiche e i comportamenti in relazione all'applicazione allo specifico problema in analisi.

Media ponderata sulla varianza

Innanzi tutto si considera in cui la grandezza acquisita sia costante, in questo caso è necessario definire la stima ricorsiva della media e della varianza per l'apparato GPS come segue:

$$\bar{h}_{gps\,k+1} = \bar{h}_{gps\,k} + \frac{h_{gps\,k+1} - \bar{h}_{gps\,k}}{k+1} \qquad \textit{Formula 4.6}$$

$$\sigma^2_{gps\ k+1} = \left(1 - \frac{1}{k}\right)\sigma^2_{gps\ k} + \left(\bar{h}_{gps\ k+1} - \bar{h}_{gps\ k}\right)^2 + \frac{\left(\bar{h}_{gps\ k+1} - \bar{h}_{gps\ k+1}\right)^2}{k} \qquad \textit{Formula 4.7}$$

e similarmente per il barometro,

$$\bar{h}_{baro\ k+1} = \bar{h}_{baro\ k} + \frac{h_{baro\ k+1} - \bar{h}_{baro\ k}}{k+1} \qquad \textit{Formula 4.8}$$

$$\sigma^2_{baro\,k+1} = \left(1 - \frac{1}{k}\right)\sigma^2_{baro\ k} + \left(\bar{h}_{baro\ k+1} - \bar{h}_{baro\ k}\right)^2 + \frac{\left(\bar{h}_{baro\,k+1} - \bar{h}_{baro\,k+1}\right)^2}{k} \qquad \textit{Formula 4.9}$$

un metodo plausibile per la fusione dei dati potrebbe avvenire, come già accennato, per mezzo di una media ponderata sulla stima delle varianze appena calcolate.

$$\bar{h}_{df\ k+1} = \frac{\dfrac{h_{baro\,k+1}}{\sigma^2_{baro\,k+1}} - \dfrac{h_{gps\,k+1}}{\sigma^2_{gps\ k+1}}}{\dfrac{1}{\sigma^2_{gps\ k+1}} + \dfrac{1}{\sigma^2_{baro\,k+1}}} \qquad \textit{Formula 4.10}$$

Nel caso in cui il barometro avesse rumore non rappresentabile da un puro rumore bianco, ma come un rumore colorato da un processo di Markov (sia questo di primo o secondo ordine), questo procedimento per la fusione dei dati risulterebbe comunque corretto ma, porterebbe ad una perdita di informazioni legate alla dinamica nel tempo dell'errore, che sarebbe possibile utilizzare per affinare la stima della quota sin dai primi campioni.

Filtro di Kalman discreto

La soluzione vagliata per risolvere il problema di datafusion, nel caso in cui la dinamica dell'errore di uno o più sensori sia assimilabile ad un rumore colorato, è l'utilizzo di un filtro di Kalman che risulta essere una adeguata soluzione per sistemi che necessitino di seguire variazioni che la media tenderebbe a filtrare.

Si è impostato il sistema per la fusione dei dati preferendo la descrizione della sola dinamica dell'errore, al posto di inserire nel modello anche la variabile h della quota, per poter eventualmente sfruttare questa stessa implementazione in altri sistemi on-board che non potranno godere della proprietà di stazionarietà di h.

Per la definizione delle matrici del sistema agli stati si sono poste le seguenti ipotesi:

δh_{gps}, errore nella quota fornita dal sistema GPS, il rumore che genera questa variabile è assimilabile ad un processo Wn come dimostrato nel paragrafo 4.5.1

δh_{baro}, errore nella quota ricavata dal Barometro attraverso l'equazione 3.2, il processo che descrive tale rumore colorato è descrivibile come un modello di Markov del secondo ordine come rilevato da Dadu in [12].

dunque la rappresentazione discreta agli stati del sistema risulta essere:

$$\begin{bmatrix} \delta h_{baro} \\ \delta \tilde{h}_{baro} \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & \delta t \\ -\alpha^2 \delta t & -2\beta\delta t \end{bmatrix} \begin{bmatrix} \delta h_{baro} \\ \delta \tilde{h}_{baro} \end{bmatrix}_k + \begin{bmatrix} 1 \\ \alpha - 2\beta \end{bmatrix} w_{baro} \qquad \textit{Formula 4.11}$$

$$\bar{X}_{k+1} = [F]\ \bar{X}_k + \bar{W}$$

scrivendo l'uscita del sistema come:

$$Y = h_{vera} - h_{baro} = (h_{vera}) - (h_{baro} + \delta h_{baro}) + v_k = \underbrace{[1 \quad 0]}_{[H]} \begin{bmatrix} \delta h_{baro} \\ \delta \tilde{h}_{baro} \end{bmatrix} + v_k \qquad \textit{Formula 4.12}$$

e avendo indicato con Y le misure ottenute, con v_k il rumore di misura e ponendo δh_{baro} come la differenza fra h_{gps} e h_{baro} dove h_{gps} viene considerata confondibile con la δh_{vera}.

Si definisce Q, la matrice della varianza del processo $Q=E[W\ W^T]$, i parametri così ottenuti sono da assumere indicativi ed eventualmente modificabili per effettuare un settaggio semi-empirico durante la fase di verifica del funzionamento dell'algoritmo.

Si definisce inoltre $R=E[v_k\ v_k^T]$ la matrice (o lo scalare) della varianza delle misurazioni ottenute dai sensori, che è bene ricordare debba essere obbligatoriamente definita positiva e P sarà la matrice di varianza dell'errore sullo stato.

Ora è possibile scrivere l'algoritmo per il filtro di Kalman dove con $X_{k+1/k}$ si indica la predizione dello stato (X) al passo k+1 ottenuta dai dati all'istante k.

La prima fase prevede il calcolo della matrice R da un set di campioni già ricevuti, in alternativa è possibile ricavare in laboratorio il valore da assegnare agli elementi e memorizzarli per il successivo utilizzo.

Successivamente si inizializza

$$P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textit{Formula 4.13}$$

con valori sulla diagonale diversi da 0 per evitare che lo stato iniziale sia assunto come esatto e conseguentemente non venga mai aggiornato, ovviamente più un valore iniziale dello stato può essere considerato attendibile, più è possibile far tendere a 0 il rispettivo valore P_0 ij.

Si definisce X_0 come il vettore dello stato iniziale, ovvero il vettore contenente dei valori ritenuti attendibili all'istante 0, in modo che il sistema possa convergere velocemente alla soluzione; qualora lo stato fosse completamente misurabile è bene porre gli elementi di X_0 come la media dei primi N campioni misurati dello stato. Si precisa che i campioni da utilizzare per la media potrebbero essere necessari anche per il calcolo delle varianza da inserire in R, quindi l'inizializzazione del vettore X_0 non comporterebbe ulteriori ritardi nell'inizializzazione dell'algoritmo.

Dopo aver definito

$$\begin{aligned} X_{k=0/k=0} &= X_0 \\ P_{k=0/k=0} &= P_0 \end{aligned} \qquad \textit{Formula 4.14}$$

è ora possibile effettuare la prima iterazione dell'algoritmo che viene comunemente suddiviso in due fasi:

1) predizione dello stato, si può interpretare come fase “predictor”

$$P_{k+1/k} = F\ P_{k/k} F^T + Q \qquad \textit{Formula 4.15}$$

Si aggiorna il valore della varianza dell’errore in base alla stima del disturbo sullo stato

$$X_{k+1/k} = F\ X_{k/k} \qquad \textit{Formula 4.16}$$

Si esegue la predizione della variabile di stato in base alle informazioni che si hanno del modello

2) aggiornamento dello stato, interpretabile come fase “corrector”

$$K_{k+1} = P_{k+1/k} H_k^T \left[H_k P_{k+1/k} H_k^T + R_k \right]^{-1} \qquad \textit{Formula 4.17}$$

E' bene notare che il guadagno K è legato al rapporto fra la previsione della varianza sull'errore dello stato $P_{k+1/k}$ e l'incertezza sulla misura Rk.

K risulterà tanto maggiore quanto l'incertezza sullo stato prevarrà su quella della misurazione.

$$P_{k+1/k+1} = P_{k+1/k} - K_{k+1} H_k P_{k+1/k} \qquad \textit{Formula 4.18}$$

in questo passaggio avviene l'aggiornamento della matrice di varianza dell'errore sullo stato, utilizzando K come parametro sulla affidabilità della previsione di $X_{k+1/k}$, rapportata alla qualità delle misurazioni.

Infine si calcola,

$$X_{k+1/k+1} = X_{k+1/k} - K_{k+1}[H_k X_{k+1/k} - Y] \qquad \textit{Formula 4.19}$$

Dove $X_{k+1/k+1}$indica lo stato ricostruito dal filtro di Kalman discreto, ora è possibile effettuare una nuova iterazione a partire dalla fase di predizione aggiornando l'indice k=k+1.

Test della fusione dei dati

Per la verifica della bontà dell'algoritmo si effettuano delle simulazioni attraverso il programma Matlab generando 35 000 e 500 000 campioni, utilizzando per il sistema GPS un segnale con media 140 m e con varianza unitaria, mentre per il rumore del barometro un processo di Markov del secondo ordine così rappresentabile

$$\begin{bmatrix} \delta h_{baro} \\ \delta \tilde{h}_{baro} \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & \delta t \\ -0.175^2 \delta t & -2*0.04 \delta t \end{bmatrix} \begin{bmatrix} \delta h_{baro} \\ \delta \tilde{h}_{baro} \end{bmatrix}_k + \begin{bmatrix} 1 \\ 0.176 - 0.08 \end{bmatrix} w_{baro} \quad \textit{Formula 4.20}$$

dove w_{baro} indica un rumore bianco con varianza di 0,4, e il segnale del barometro è ottenuto sommando al processo di Markov una quota costante di 140 m.

I due disturbi sono volutamente esagerati rispetto alle peggiori condizioni rilevabili dai datasheet dei moduli per evidenziare le qualità o le eventuali pecche dei metodi di fusione.

Una rappresentazione dei disturbi introdotti sulle misure è riportato in figura 42, dall'immagine è possibile notare come il processo di Markov riconducibile al Barometro, pur essendo generato da Wn con varianza minore del GPS, fornisca dati molto più fluttuanti che quelli forniti dal sistema GPS, infatti la varianza dei dati provenienti dalla simulazione del modulo Barometro hanno varianza superiore a 20, questo implica che sarà possibile effettuare una fusione fra dati ottenendo un ampio margine di miglioramento.

Successivamente si è proceduto al filtraggio con il filtro di Kalman discreto come descritto in precedenza, verificando la bontà dell'algoritmo implementato.

Come si vede chiaramente dalla Figura 45 la misurazione del barometro filtrato con algoritmo KF risulta più attendibile, ovvero con varianza minore (circa 0,47), rispetto alla misura effettuata con il GPS (a varianza unitaria).

Si procede successivamente a simulare la fusione dei dati con un diversi metodi, Media ricorsiva, Media Ricorsiva pesata sulla varianza (ricavata ricorsivamente) e infine data fusion con Kf.

Si nota come la media pesata sulla varianza campionaria fornisca molto velocemente una stima corretta della grandezza, invece la media “pura” sia polarizzata dalle fluttuazioni del modello di Markov e richieda più di 140 minuti e 34 000 campioni per tendere asintoticamente al valore corretto.

L'andamento dei dati filtrati con KF sono i meno stabili e fluttuano descrivendo idealmente un rumore bianco di intensità che tende a essere pari a quella del Wn usato nel modello di generazione dell'errore prescelto.

Il fatto che si pervenga ad un segnale che tenda ad un rumore bianco è indice che tutte le dinamiche conosciute sono state assorbite durante il filtraggio e quello ottenuto e il valore più preciso ottenibile della misurazione.

Si procede a valutare l'evolversi delle medie e delle rispettive varianze:

- la media “pura” come già accennato richiede più di 34000 campioni per convergere al valore asintotico e ammettendo di disporre fin dal primo campione della descrizione statistica esatta del segnale, la varianza associata alla media pura, confondibile con l'intervallo di confidenza ad un σ, risulta inferiore alla varianza del segnale ottenuto attraverso il KF solo dopo 5 o 6 campioni, ovvero dopo un secondo e mezzo

- la media pesata sulla varianza invece tende molto velocemente al valore asintotico della grandezza misurata, richiede approssimativamente 100-150 campioni (ovvero 25-37,5 secondi) per ottenere una approssimazione migliore del Filtraggio alla Kalman e 300 - 350 campioni (1 minuto e mezzo circa) per entrare nell'intervallo di confidenza ± 0.5% Valore reale.

  Considerando la disponibilità delle varianze corrette sin dal primo istante, la varianza della media pesata risulta minore della varianza del segnale ricostruito con filtro di Kalman sin dal quarto campione. E' indispensabile precisare che la stima ricorsiva delle medie e delle varianze non sia considerabile come una esatta descrizione statistica dei processi fin dai primi dati, in quanto gli stimatori, pur se corretti, tendono ai valori di media e varianza esatte solo in presenza di un numero infinito di campioni.

  Si valuta ora anche l'idea di utilizzare una media a finestra mobile pesata sulla varianza dell'errore dei sensori, che ben si adatterebbe inoltre a misurazioni di grandezze con comportamenti non costanti; chiaramente le varianze dell'errore degli strumenti si presume già calcolata durante precedenti prove empiriche.

  Come in ogni media con finestra mobile è necessaria una riflessione sulla scelta del un numero di campioni adeguato per poter filtrare correttamente il segnale e non introdurre un eccessivo ritardo nella dinamica ricostruita.

  Nel caso in esame, un numero adatto di campioni da considerare, sarebbe all'incirca di 10 – 20 ma in questa fase dello studio non è possibile indicare, se con l'implementazione finale, il numero di campioni da includere nella finestra possa aumentare, ciò comporterebbe un maggior filtraggio delle dinamiche più veloci, e di conseguenza l'introduzione un ritardo ancora più apprezzabile nella risposta del valore filtrato rispetto all'andamento reale.

  Data la difficoltà di valutare in questa simulazione costante, quale possa essere la soluzione più performante, fra una media pesata sulla varianza dell'errore con finestra mobile e KF, per il problema in esame, si provvede a confrontare i due metodi in una successiva simulazione.

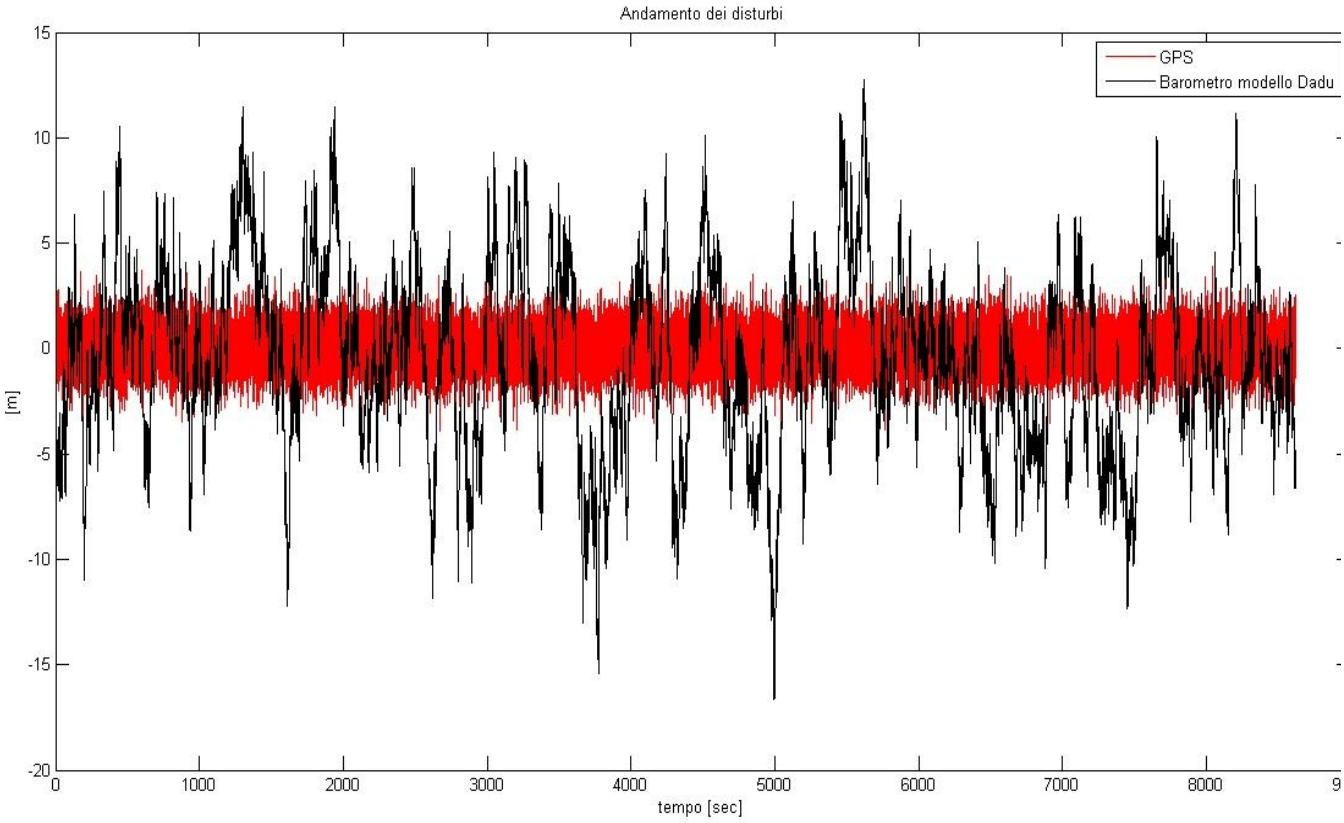

Figura 42 andamento dei disturbi GPS e Barometro.

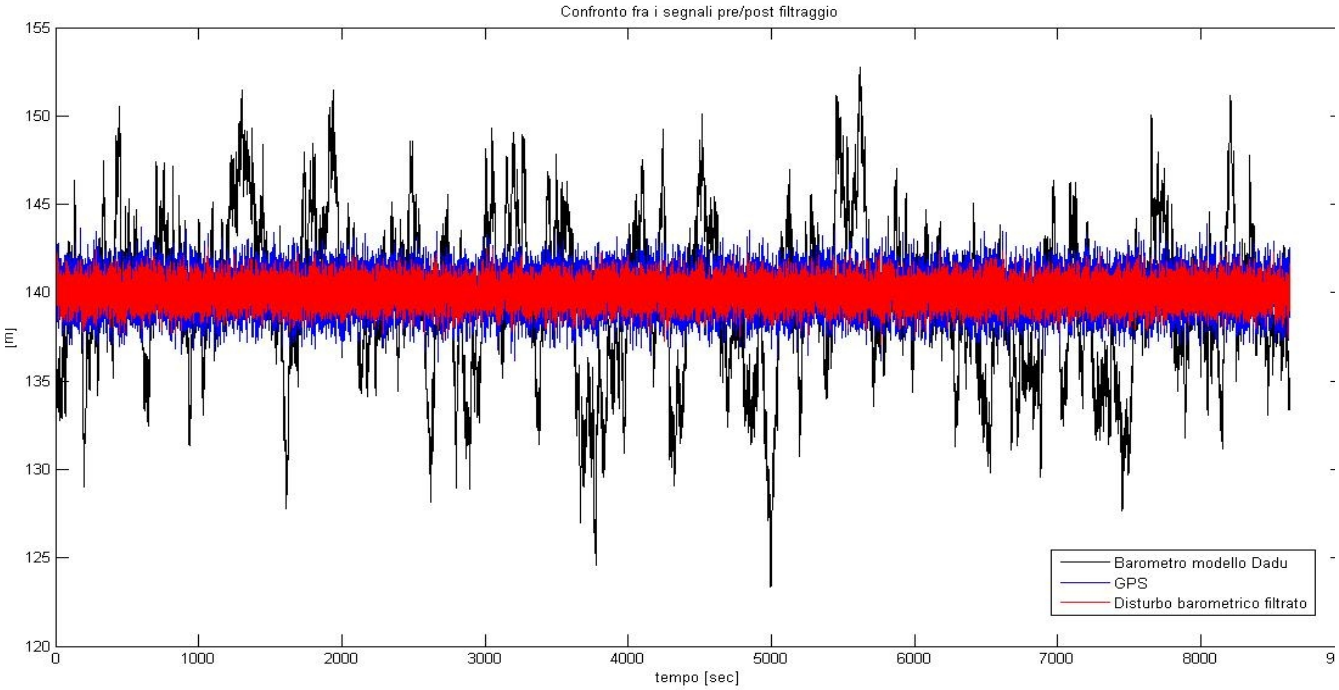

Figura 43: confronto fra la misura del Barometro filtrato, non filtrato e misurazione GPS

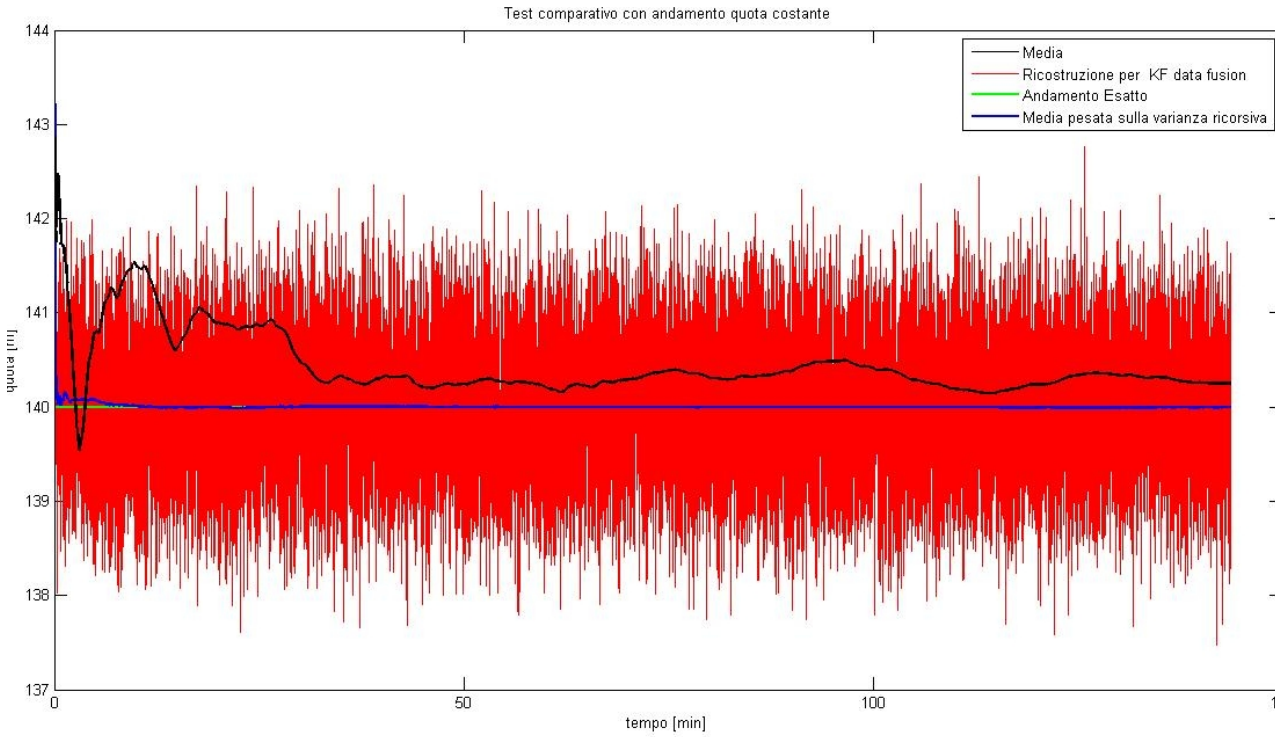

Figura 44: ricostruzione del segnale costante con vari metodi di fusione dei dati

La seconda simulazione prevede che l'altezza misurata dai due sensori vari, simulando il caso di un apparato posto su un velivolo in volo, dove la dinamica associata ai cambi di altitudine vengono implementata come la somma di tre segnali sinusoidali con costanti di tempo 375 sec, 3750 sec, 6250 sec; si preferisce omettere anche in questa fase il confronto Kf / media a finestra variabile pesata sulla varianza dell'errore dei sensori, riservandosi di esaminarle a seguito dell'inserimento delle dinamiche più veloci del modello.

Il risultato mostrato in figura 45 evidenzia come la media non sia uno stimatore adatto a eventi dinamici, e che se si vuole poter sviluppare un algoritmo per l'integrazione dei dati che sia valido anche per migliorare la stima della quota del sistema in volo, è necessario utilizzare un filtro di Kalman.

Si noti come la dinamica ricostruita con KF ricalchi quasi perfettamente l'andamento esatto, fornendo un errore massimo di 3,7 m, cioè inferiore all'errore massimo commesso dalla simulazione della misura del solo apparato GPS.

Ci si può dunque ritenere soddisfatti dell'algoritmo di fusione dei dati sempre che risulti valida l'ipotesi che il modulo HCA0611AR presenti un rumore che sia descrivibile da un processo di Markov di primo o secondo ordine, come ipotizzato da Bose o Dadu e che vengano correttamente stimati i parametri che lo descrivano.

Si valuta ora se sia accettabile il ritardo introdotto da una media una finestra mobile di 10-20 campioni, pesata sulla varianza dell'errore dei sensori in presenza di dinamiche veloci.

É bene notare che per la GS questo problema non si pone, essendo un sistema statico che può essere affetto da disturbi riconducibili a lenti cambi di pressione atmosferica. La ricostruzione del segnale quasi statico, attraverso media a finestra mobile pesata sulla varianza dell'errore e KF, fornirebbero risultati paragonabili a fronte di una differenza computazionale rilevante.

Si simula ancora una volta un andamento dinamico della quota introducendo una dinamica più veloce di quelle precedenti che simuli una manovra di volo, e confrontando i risultati ottenuti dalla Media a finestra mobile pesata sulla varianza dell'errore e KF si ottiene il grafico di figura 46 dove si può notare che il filtraggio applicato dalla media con una soli 15 campioni non garantisce sempre un perfetto inseguimento del segnale, e contemporaneamente è possibile si apprezza come tale finestra causi un ritardo fra il segnale ricostruito con KF e media mobile sia tutt'altro che trascurabile, si desume da questa prova che in caso venga previsto un filtraggio dei dati provenienti dal velivolo sia auspicabile l'utilizzo del KF.

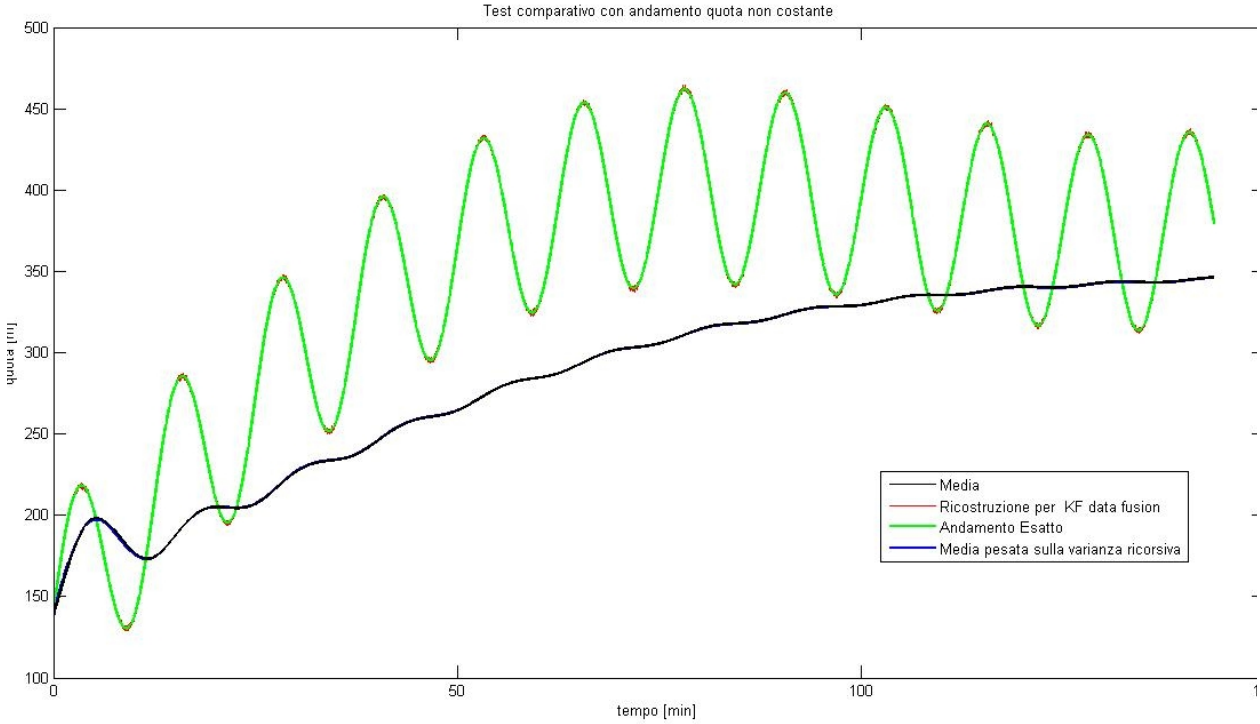

Figura 45: andamento della ricostruzione con simulazione di variazione di quota di un velivolo

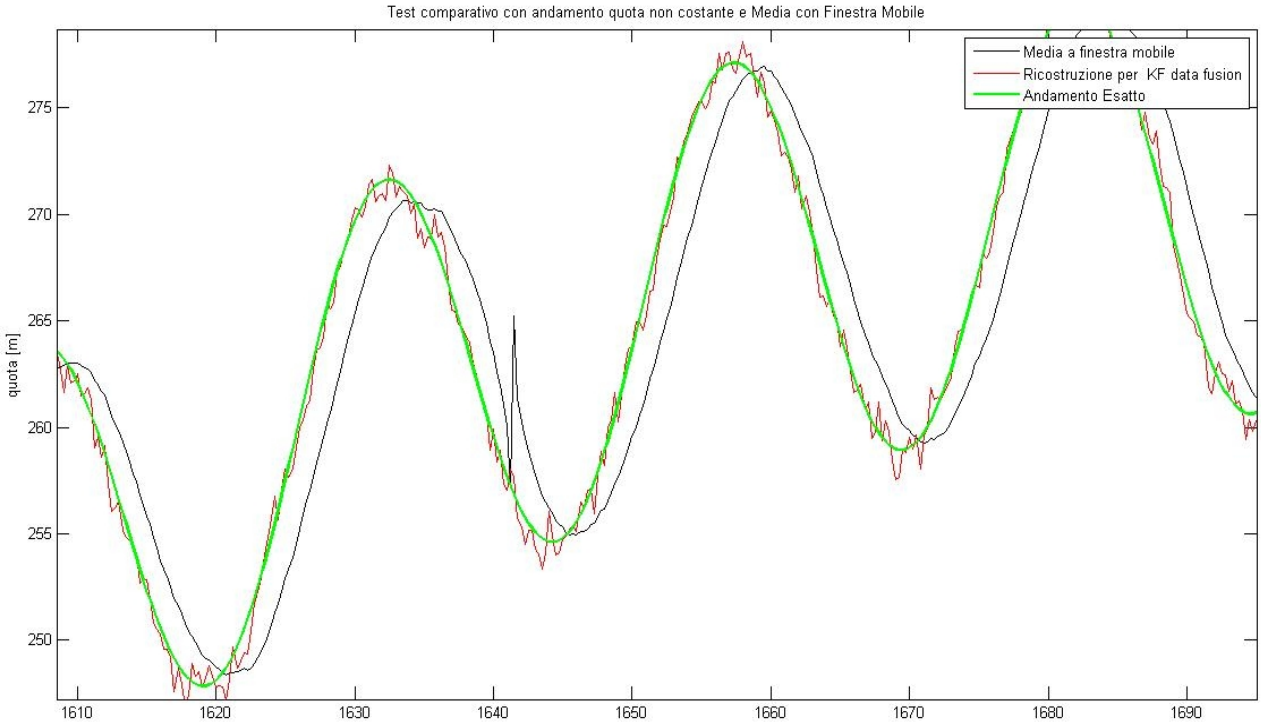

Figura 46: confronto KF media mobile varianza errore

5 Studio di fattibilità del Sistema di puntamento dell'antenna

In questa sezione verrà condotto uno studio di fattibilità dell'unità di puntamento d'antenna, descrivendo il sistema, analizzando le scelte operate per la selezione del Motor Driver e i problemi di implementazione analizzati

5.1 Scelta degli elementi per il sistema puntamento d'antenna

Il sistema di puntamento è stato progettato come un apparato separato dal server principale e composto da una unità di calcolo di tipo mini ITX. Questa è collegata al server GS attraverso una connessione Lan, così da poter ricevere i pacchetti contenenti la posizione GPS del velivolo e della GS e, sfrutta una linea CAN Bus o seriale per interfacciarsi con il controller dei motori,

Il sistema puntamento antenna è concepito come rappresentato nel seguente schema:

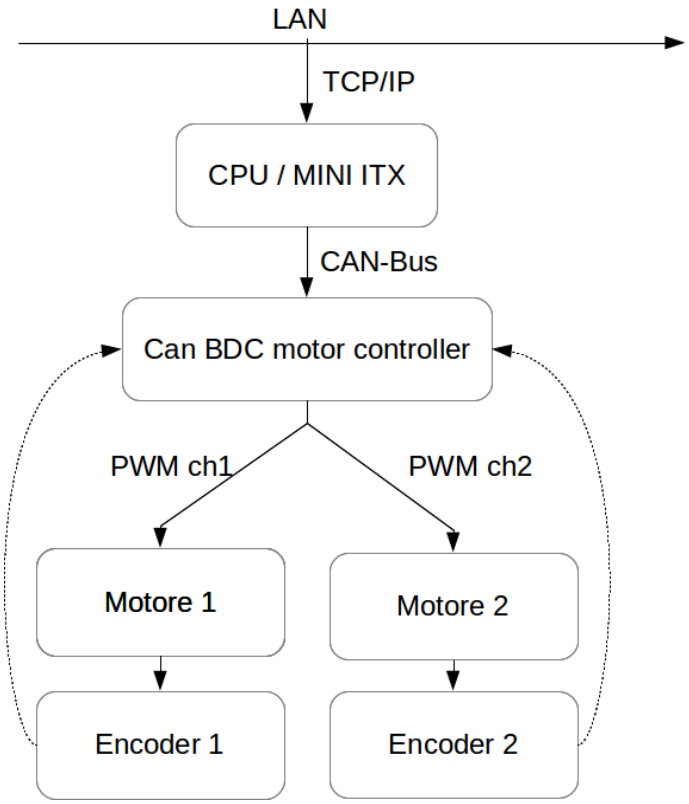

Figura 47: schema del sistema controllo e direzionamento antenna

5.1.1 Controller per motori DC Brushed

Le specifiche utilizzate per la scelta del modulo di controllo dei motori per il puntamento dell'antenna sono state ricavate principalmente da due esigenze:

- Interfacciare il modulo di controllo dei motori con il terminale Mini ITX, da qui nasce l'esigenza di supportare il protocollo CAN-Bus, e possibilmente anche quello seriale.

- Supportare la piattaforma tilt – pan Quickset Moog QPT 20 dotata di due motori e due potenziometri, da questo vincolo progettuale vengono ricavati tutti i parametri di Amperaggio massimo continuo e di picco, massima potenza richiesta continua, massimo voltaggio ammesso.

Le caratteristiche così ottenute sono state:

- il modulo deve essere alimentabile con tensione 220V AC o DC inferiore
- presenza di una o più porte CAN-Bus e/o porte seriali
- possibilità di gestire almeno due motori
- Massima corrente continuativamente erogabile per motore 1.0A
- Massima corrente di picco 10 A
- voltaggio di alimentazione motori 20-24V
- Frequenza di PWM > 1KHz
- due ingressi analogici per la lettura dei potenziometri

costituisce requisito preferenziale:

- possibilità di settare i limiti di finecorsa nella memoria interna del modulo
- Presenza di due ingressi digitali e per l'eventuale uso di encoder
- possibilità di controllare il sistema in posizione e velocità.
- Possibilità di supportare carichi fino a 3A per motore, a 50V (Per specifiche necessità che verranno esposte nella sezione 5.2, si è assunto come limite massimo di corrente il valore di 3A @ 50VDC)

una accurata ricerca porta a definire due classi di prodotti adatti allo scopo, i controllori utilizzati in campo robotico e quelli progettati per l'automazione industriale.

I primi, hanno costi più contenuti, vengono proposti per applicazioni di laboratorio, non posseggono certificazione per utilizzi intensivi e non esistono modelli che supportino nativamente il protocollo CAN-Bus

I controllori per automazione industriale, invece, vengono certificati appositamente per utilizzi intensivi hanno margini di errore di uno o due ordini di grandezza inferiori, spesso implementano sia la porta CAN che quella seriale, al contrario dei controller per utilizzo robotico presentano costi elevati.

E' stato subito possibile notare come tutti gli apparati utilizzabili in campo robotico abbiano caratteristiche molto simili fra loro (pur essendo 15 prodotti di 12 case produttrici diverse) e manchino tutti dei requisiti accessori.

I prezzi di questi dispositivi variano dai 49 a 109 euro.

Invece un raffronto più accurato è necessario per i controller del campo dell'automazione industriale le cui caratteristiche più salienti per il progetto in corso sono state elencate nella seguente tabella

marca	modello	tutti requisiti essenziali	Supporto 3A @ 50 V	limiti attuatori memorizzabili	Ingressi per potenziometri	controllo in posizione e/o velocità	seriale	CAN	Motori supportati
Elmo MC	Elmo Cell 5/60	SI	SI	si	opzionali	si	si	si	2
Elmo MC	Elmo Gold Whistle	SI	SI	si	NO	si	si	si	2
Texas Inst.	MDL-BDC24	SI	SI	si	4 e A\D 24bit	si	si	si	1
PMD	ION 500	SI	SI	NO	NO	NO	opzionale	si	1
CIRO	BCD130-CAN	SI	SI	NO	NO	NO	NO	si	1

Tabella 5.1: caratteristiche dei controllori per motori Brushed del campo Automazione industriale

Fatta questa comparativa si designa il controller Elmo Cell 5/60 come il più adatto per i modelli per automazione industriale. Invece si designa il modello B0099 della Basic Micro come il candidato più adeguato fra i controller di campo robotico, basando la scelta puramente sulla disponibilità di un dettagliatissimo datasheet, e di uno dei prezzi più economici sul mercato a parità di componentistica utilizzata.

5.2 Studio delle traiettorie per inseguimento velivolo

Lo scopo del sistema di puntamento è quello di direzionare l'antenna sempre verso il velivolo al fine di mantenere a valori adeguati il guadagno di ricezione a garantire così il datalink.

Si stima che l'apertura del cono d'antenna sia all'incirca di 5°, richiedendo dunque un consona precisione nel puntamento.

Il sistema si basa sull'analisi dei dati di posizione GPS del velivolo e della stazione GS; questi una volta elaborati con la funzione di Caffe_GPS ComputeAzimuthAndElevationFromECEF permettono di ottenere gli angoli pan e tilt ovvero Azimuth ed elevazione per puntare correttamente l'antenna e garantire un solido data-link fra FTI on board e la GS.

Limiti dell'attuatore dell'antenna

L'attuatore pan-tilt dell'antenna denominato Quickset Moog QPT-20 non prevede una rotazione continua ma, ha fine corsa a ±217,5° (ovvero di 435°) in pan con una velocità di attuazione fra 2 e 35°/ sec, mentre in Tilt il limite dell'attuatore è di 180° dall'orizzonte con velocità di attuazione fra 1 e 12°/sec,

Si identifica come primo problema l'effetto del finecorsa in pan dell'attuatore nel caso in cui il velivolo si trovi a volare in circolo intorno alla GS, una situazione che durante le prove di volo non è affatto remota.

Analizzando le specifiche dell'attuatore si definisce un'altra condizione critica, il caso del sorvolo della antenna, qui il limite fondamentale non è la posizione raggiungibile in tilt ma la massima velocità di attuazione di 12°/sec.

Idee vagliate eliminare gli effetti di fine corsa

L'effetto del finecorsa è l'impossibilità nell'inseguire il velivolo dopo che abbia volato in circolo intorno alla GS per più di 435°, di conseguenza si è sviluppata la ricerca di un metodo che permetta di “azzerare” la posizione in pan una volta raggiunto un limite massimo.

Sono stati sviluppati due approcci differenti al problema:

- il primo approccio prevede che, una volta raggiunto il finecorsa, l'attuatore compia alla massima velocità possibile una rivoluzione in senso opposto di 360° fino a riagganciare il velivolo
- Il secondo approccio prevede che, una volta raggiunto il finecorsa, l'attuatore che direziona l'antenna compia, sempre alla massima velocità, una rotazione in senso opposto di soli 180° e contemporaneamente che l'antenna assuma un angolo di tilt supplementare al precedente intercettando nuovamente il velivolo.

Entrambi gli approcci hanno dei pregi e dei difetti che ora si andranno ad analizzare nello specifico.

Il primo approccio richiede un maggior tempo per riagganciare il velivolo e conseguentemente se il questo volasse a meno di 1500 m di distanza dalla GS e virasse bruscamente (e/o cambiasse repentinamente quota), nei 10,3 secondi necessari per la rotazione di 360°, l'antenna potrebbe non

essere più in grado di intercettare il velivolo, cercandolo in una posizione errata, di conseguenza verrebbe a mancare definitivamente il flusso di dati provenienti dal FTI on board.

Il pregio di questo metodo invece, è di richiedere una sola rotazione ogni 360°. Viene inoltre ritenuto che in questo tipo di approccio sia sensato suddividere il campo di moto dell'attuatore in 2 settori di ±185°, consentendo così una zona di sovrapposizione di vista fra il pre e post riaggancio, in modo da poter eventualmente implementare un metodo di ricerca del velivolo basato su algoritmi di dead reckoning.

Il secondo metodo, prevede un movimento tilt+pan, richiede la metà del tempo rispetto al metodo precedente, (5,15 secondi) per riagganciare un velivolo che voli almeno 59° sopra l'orizzonte (una condizione non sempre verificabile nelle prove pratiche).

La condizione dei 59° è dettata dalla differente velocità di rotazione fra i due assi, infatti nel tempo che la piattaforma descrive di 180° in pan a 35°/sec, l'antenna riesce a ruotare in tilt di soli 62° (a 12°/sec), questo comporta che tutte le posizioni in cui il velivolo si trovi al di sotto di 59° di alzo richiedano un tempo aggiuntivo di puntamento per il tilt che supera i 5.15 secondi.

Analizzando il caso peggiore immaginabile, si nota che per un velivolo che si trovi sull'orizzonte, dunque con tilt iniziale a 0°, sarà necessario compiere un pan di 180° e un tilt di 180°, come già detto essendo il velivolo sotto i 59° il movimento che determina il tempo di riaggancio è la rotazione in tilt che richiederebbe 15 secondi, inficiando completamente l'effetto benefico della corsa più breve in pan.

Volendo implementare un solo metodo per l'eliminazione dell'effetto dei finecorsa non rimarrebbe che scegliere il primo approccio, ovvero il posizionamento della torretta a -180° una volta raggiunto il limite dei +180°.

In realtà sarebbe più proficuo creare un algoritmo per le leggi di moto, in grado di utilizzare entrambi gli approcci sfruttando come discriminante l'ultima posizione di alzo dell'antenna prima del raggiungimento del finecorsa, se questo fosse superiore ad una certa soglia sarebbe opportuno utilizzare il metodo tilt+pan

Invece nel caso l'ultimo angolo di tilt registrato prima del raggiungimento del fine corsa fosse inferiore alla soglia limite sarebbe più corretto eseguire la sola rotazione in pan di 360°.

Non rimane che effettuare una prima stima di quale possa essere un corretto valore dell'angolo di alzo da assumere come soglia di switch fra i due metodi.

Ad una prima analisi il limite di 59° di alzo risulta il più logico, in quanto la lentezza di rotazione in tilt non pregiudica il tempo di riaggancio del velivolo, ma ponendo il caso di un velivolo che sia a soli 55° sull'orizzonte della GS, si otterrà un tempo di riaggancio di 10.3 secondi per la sola rotazione in pan, mentre il riaggancio con approccio tilt+pan richiederebbe solo 5.8 secondi.

Nel caso il velivolo si trovi a 28° sull'orizzonte si ha che i due tempi di riaggancio fra i due metodi siano sostanzialmente identici, dunque in questa situazione sarebbe da preferire la rotazione di puro pan che permette di inseguire il velivolo per ulteriori 360° di rotazione, (con l'ipotesi che il velivolo stia ruotando attorno alla GS).

Si può allora accettare come un buon compromesso la soglia di switch dei 45° in tilt che con con approccio tilt+pan permette di riagganciare il velivolo in soli 7.5 secondi con un risparmio del 27% del tempo richiesto dalla rotazione di 360° in pan.

Idee vagliate per il sorvolo antenna

Un velivolo ultraleggero può raggiungere velocità ragguardevoli stimabili fra i 55 m/s e i 70m/s (ovvero fra i 200Km/h e i 250Km/h) questo comporta che il sorvolo a velocità massima dell'antenna sarebbe correttamente eseguibile solo se effettuato con quota minima fra i 264m e i 336m, inoltre anche un sorvolo a velocità di stallo (intorno ai 70 Km/h) richiederebbe una quota minima di poco inferiore a 100 m, questo non è assolutamente accettabile durante le prove in volo. Quindi ci si pone il problema di ovviare alla bassa velocità di rotazione in tilt, che si traduce nell'impossibilità di seguire correttamente un velivolo che sorvoli la GS con legge di moto espressa in metri:

$$V_{sorvolo} > \frac{Quota\,sulla\,stazione}{0.20944} \qquad \textit{Formula 5.1}$$

Non vi sono molte soluzioni possibili da vagliare: è necessario aumentare la velocità dell'attuatore.

Riesaminando il datasheet dell'attuatore si nota che il Duty cycle consigliato è 20%, questo valore indica che l'attuatore è stato dimensionato per funzionare con una fase attiva pari ad 1/5 del tempo totale, ciò implica che tutti i parametri riportati nel datasheet come massimi siano in realtà i valori sopportabili con continuità senza danneggiare l'attuatore, e che siano stati ottenuti utilizzando un duty cycle all'incirca del 20% e comunque certamente inferiore al 50%.

Questa caratteristica suggerisce di esaminare la possibilità di sovra alimentare l'attuatore per pochi secondi durante la fase di sorvolo, al fine di ottenere delle prestazioni migliori, per poi farlo tornare a regime di esercizio normale dove si è fiduciosi possa disperdere agevolmente il calore accumulato nella condizione di sovra alimentazione.

Va ricordato infatti che un Motore Brushed DC viene danneggiato da un uso prolungato al disopra del voltaggio massimo continuo consentito e inferiore al doppio del voltaggio massimo, a causa dell'impossibilità di disperdere tutto il calore accumulato durante l'esercizio [15].

Non essendo reperibili le curve di funzionamento dei motori utilizzati dall'attuare QPT 20 si prevede, appena sia disponibile il Controller, una campagna di prove empiriche sull'attuatore di tilt al fine di verificare che una sovra alimentazione a 35-40V del motore permetta di ottenere per pochi secondi prestazioni abbastanza elevate che permettano almeno un raddoppio della velocità di rotazione ovvero 24°/sec, verificando contemporaneamente che le temperature sugli avvolgimenti non superino mai i 150°C limite consigliato che ricaviamo dal testo [15].

6 Conclusioni e Sviluppi futuri

In campo software con le versioni Mk6 e Mk7 si può ritenere che il lavoro sia stato completato per il funzionamento con file di Backup e che il server sia già pronto per processare i dati ricevuti dal vivo, ma è necessario definire una libreria apposita per gestire la “lettura” dello streaming dall'antenna. Il fatto che il software si sia dimostrato correttamente funzionante e sia stato testato con diverse metodologie non esclude che rimangano ancora aperti molti punti di possibile sviluppo per integrare nuove funzionalità non previste nella fase iniziale come: una interfaccia grafica e/o la possibilità di tracciare immediatamente i grafici di talune grandezze; ovviamente, sono possibili anche ulteriori ottimizzazioni del codice al fine di migliorarne le performance, come aumento della velocità di esecuzione e ottimizzazione dell'utilizzo della memoria.
E' bene rimarcare come tutte le librerie sviluppate in questo progetto si prestino ad essere utilizzate come elementi per lo sviluppo di altri programmi completamente differenti da questo, che richiedano ingegnerizzazione dei dati in ambiente Linux, elaborazione di posizione GPS e dati Aria. Inoltre le librerie Caffe_Air e Caffe_GPS possono essere implementate in qualsiasi sistema operativo, o anche in applicazioni embedded essendo stata verificata la compatibilità delle chiamate e degli argomenti passati alle funzioni con le principali librerie embedded (armlib e avr-libc).
La dimostrazione della versatilità delle librerie implementate, ottenuta con lo sviluppo del software per l'ingegnerizzazione dei dati chiamato “MK sperimentazione in volo 2012”, ha permesso inoltre una verifica intensiva in condizioni reali del corretto funzionamento di tutte le funzioni principali del SW sviluppato.
Per quanto concerne la scelta di un algoritmo di datafusion, per affinare la stima della quota, si è potuto apprezzare come la media a finestra mobile pesata sulla varianza dell'errore non sia adatta a descrivere andamenti dinamici della variabile, mentre per applicazioni a terra, dove la quota risulta costante è sicuramente fra i metodi più adatti ed in grado di fornire dopo alcuni secondi di campionamento risultati migliori di un filtraggio alla Kalman.
Successivamente si è proceduto ad analizzare i due modelli teorici di errore del Barometro proposti in letteratura, scegliendo lo sviluppo del modello di Markov del secondo ordine proposto da Dadu, fiduciosi che questo possa ben approssimare anche un sistema di Markov del primo ordine con le varianti proposte da Bose.
Quando saranno disponibili le misurazioni del barometro si potrà applicare il metodo di massima verosomiglianza per ricavare i parametri che descrivano il processo di generazione dell'errore dello strumento e conseguentemente sarà possibile verificare le ipotesi sui modelli utilizzati e implementare il filtro di Kalman, per ottenere una fusione dei dati provenienti dal barometro e dal GPS anche per i dati acquisiti dal sistema FTI on board potendo fornire così una stima migliore della quota del velivolo durante il volo.
Per quanto riguarda il modulo hardware dedicato alla raccolta dei dati di pressione a terra e di posizione della Ground Station, si è proceduto ad una accurata verifica della progettazione delle connessioni e al test di funzionamento dei singoli elementi.
Infine si è proceduto allo studio di fattibilità del sistema di puntamento dell'antenna che ha condotto alla precisa definizione di tutti i componenti necessari, all'implementazione dello schema di funzionamento e alla soluzione dei due principali problemi evidenziati dall'utilizzo dell'attuatore Quikset moog QPT20, ovvero come poter continuare l'inseguimento del velivolo una volta raggiunto il finecorsa in pan, e in caso di sorvolo ravvicinato dell'antenna.

Le soluzioni presentate sono solo uno studio di fattibilità e si prestano ad essere sviluppate in innumerevoli modi in una successiva fase avanzata ed arricchiti da algoritmi per la previsione della posizione di riaggancio del velivolo basata sulle ultime posizioni GPS rilevate dal FTI on board.

Più complesso ma dai risultati sicuramente molto più prestanti sarebbe l'affiancamento all'attuale metodo di tracking di un sistema di riconoscimento ottico computerizzato del velivolo.

7 Riferimenti

[1] Daniel J. Barrett, Richard E. Silverman, Robert G. Byrnes, *"SSH, The Secure Shell: The Definitive Guide",* OREILLY, 2005

[2] ISO, *"Information processing systems -- Open Systems Interconnection -- Basic Reference Model",* ISO, 1989

[3] AA VV, "*Using the GNU Compiler Collection"* GNU licence, http://gcc.gnu.org/onlinedocs/gcc-4/gcc.pdf, 2012

[4] AA.VV., *"IEEE 754: Standard for Binary Floating-Point Arithmetic",* IEEE Computer Society, 2008

[5] Mark A. Lindner, *"http://www.hyperrealm.com/libconfig/libconfig_manual.html#Introduction",* 2012

[6] EUROCONTROL, IfEN, *"WGS 84 IMPLEMENTATION MANUAL"* 1998

[7] R.P.G. Collinson *"Introduction to Avionics Systems Third Edition"*, 2011, pp 377 – 418

[8] John R. Taylor, *"An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements"*, 1997, pp60 – 103.

[9] Simone Piccardi, *"GaPiL Guida alla Programmazione in Linux" GNU Free Documentation",* 2011,

[10] Wireshark *"http://www.wireshark.org",* 7/2012

[11] Sam C. Bose, *"An Intensive course in GPS/INS Multisensors Kalman filter Navigation",* 2003

[12] Vivek Dadu ,B.Venugopal Reddya, Brajnish Sitaraa, R.S.Chandrasekhara & G.Satheesh Reddy, *"Baro-INS integration with Kalman Filter",* Hindustan Aeronautics Ltd, Korwa, 2008

[13] Sergio BITTANTI, *"Identificazione dei modelli e controllo adattativo"*, Pitagora, 1990

[14] Sergio BITTANTI, *"Teoria della predizione e del filtraggio",* Pitagora, 1990

[15] G Magnani, G Ferretti, P Rocco, *"Tecnologie dei sistemi di controllo",* McGraw-Hill, 2007

[16] Glenn D. MacGougan, *"The Essential GNSS Project"* http://gnsstk.sourceforge.net/ ultima vistita 7/2013

[17] Brian Hall, *"Beej's Guide to Unix Interprocess Communication",* 2007,

[18] Kay A. Robbins, Steven Robbins, *"Unix Systems Programming: Communication, Concurrency and Threads", Prentice Hall* ,2003

[19] Eric S. Raymond, "*The Art Of Unix Programming, Addison-Wesley",* 2003

[20] Ren C. Luo, hih-Chen Yih, and Kuo Lan Su *"Multisensor Fusion and Integration: Approaches, Applications, and Future Research Directions",* IEEE Jurnal of sensors N 2 , 2002

[21] Mark Mitchell, Jeffrey Oldham, Alex Samuel, *"Advanced Linux Programming",* New Riders , 2001

[22] W. Richard Stevens, *"UNIX Network Programming, Volume 1"* Addison Wesley Professional, 2005.

[23] Alessandro Bellini, Andrea Guidi, *"Linguaggio C - Guida alla programmazione 4/ed",* McGraw-Hill Companies, 2006.

[24] H. M. Deitel, *"C: How to Program",* DEITEL, 2006

[25] Brian Hall, *"Beej's Guide to Network Programming Using Internet Sockets",* GNU licence, http://www.beej.us, 2008

[26] AA VV, *"gentoo.org Spiegazioni sui thread POSIX, parte 1",* http://www.gentoo.org/doc/it/articles/l-posix1.xlm, visitato 8/2010

[27] Schwartz, K. P. *"ENGO 421 Lecture Notes - Fundamentals of Geodesy."* 1997

[28] AA VV, *"gentoo.org Spiegazioni sui thread POSIX, parte 2",* http://www.gentoo.org/doc/it/articles/l-posix2.xlm, visitato 8/2010

[29] AA VV, *"gentoo.org Spiegazioni sui thread POSIX, parte 3",* http://www.gentoo.org/doc/it/articles/l-posix3.xlm, visitato 8/2010

[30] Unibo Sistemi Operativi L-A , Paolo Bellavista, Eugenio Magistretti e Ing. Marco Montali, *"lucidi del corso lezione 10", http://lia.deis.unibo.it/Courses/sola0506-info/lucidi/10.pdf,* 2010

[31] Marco Latini, Paolo Lulli , *"Imparare il C", GNU licence,* http://www.lulli.net/prj/imparareC/ ,2006

[32] Gary R. Wright, W. Richard Stevens, *"TCP/IP Illustrated, Volume 2: The Implementation",* Addison-Wesley Professional ,1995

[33] Giuseppe Lipari, Luca Abeni, Antonino Casile *"Programmazione concorrente su sistemi Unix",* http://www.sitoserio.it/cpp/index.htm, 2006

[34] Vernon C. Hoxie, *"The Linux Serial Programming HOWTO part 1"* URL http://www.lafn.org/~dave/linux/Serial-Programming-HOWTO.txt, ultima visita 2012

[35] R. Dantes, *"NetBeans IDE Cookbook",* PACKT, 2011

[36] Tang, Haochen, Blanch, Juan, Walter, Todd, Enge, *"Flight Test Data Validation of GPS Ranging Error Characteristics",* ION, 2009

[37] Sergio BITTANTI, *"Serie temporali e processi casuali",* Pitagora, 2005

[38] U-blox, *"NTARIS4_Modules_SIM(GPS.G4-MS4-05007) Manual" U*-blox, 2008

[39] Sensortechnics inc , *"HCA-BARO Series Manual",* Sensortechnics inc, 2009

[40] FTDI chip, *"FT4232H Mini Module manual",* Future Technology Devices International Limited, 2012

[41] FTDI chip, *"FT4232H data sheet",* Future Technology Devices International Limited, 2012

[42] FTDI chip, *"AN 111 Programmers Guide for High Speed FTCSPI DLL",* Future Technology Devices International Limited, 2012

[43] FTDI chip, *"AN 124 User Guide For FT PROG",* Future Technology Devices International Limited, 2012

[44] FTDI chip, *"AN 127 User Guide For FT2232HD Factory test utility",* Future Technology Devices International Limited, 2012

[45] FTDI chip, *"AN 178 User Guide for LibMPSSE-SPI",* Future Technology Devices International Limited, 2012

[46] FTDI chip*, "AN 180 FT232H MPSSE Example - USB Current Meter using the SPI interface",* Future Technology Devices International Limited, 2012

[47] FTDI chip, *"AN I2C-Bus-HDI-HCLA-HCA-SSI E 11155",* Future Technology Devices International Limited, 2012

[48] FTDI chip, *"AN 135 FTDI MPSSE Basics", Future Technology Devices International Limited, 2012*

[49] J.Kim, S. Sukkarieh, *"A baro-altimeter augmented INS/GPS navigation system for an uninhabited aerial vehicle",* Australian Global Positioning System society Inc, 2003

[50] Hugh Durrant-Whyte , *"Multi Sensor Data Fusion Note", A*ustralian Centre for Field Robotics, 2011

[51] Ublox, *"ANTARIS 4 GPS Modules SIM System Integration Manual",* Ublox , 2007

[52] Quickset International Inc, *"MOOG QPT-20 Series Pan & Tilt Positioners complete manual & datasheet",* Quickset International Inc, 2003

[53] SSD addicted database data *URL: www.ssdaddict.com/chart_USE.php,ultima vistita 7/2013*

[54] Ing. Stefano Maggi *"Dispensa_CAN",* http://docenti.etec.polimi.it/IND32/Didattica/--/Dispensa_CAN.pdf, 2005