ALMA MATER STUDIORUM -
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Sede di Forlì

CORSO DI LAUREA in
INGEGNERIA AEROSPAZIALE

Classe L-9

ELABORATO FINALE DI LAUREA
In Controlli Automatici

STUDIO E PROGETTO DI UN DISPOSITIVO PER LA REPLICA IN PLAYBACK DI VIBRAZIONI MECCANICHE

Candidato:

Anthony Paccioni

Relatore:
Prof. Matteo Zanzi
Corelatore:
Ing. Antonio Ghetti

Anno Accademico 2014/2015

Sessione III^a

Indice

INDICE DELLE FIGURE

INTRODUZIONE

Il lavoro di tesi qui presentato ha avuto l'obiettivo, attraverso lo studio dei più comuni servomotori stepper, di replicare profili di accelerazione registrati o definiti con l'utilizzo di una piattaforma hardware Arduino in grado di salvare ed elaborare dati. Per raggiungere tale risultato è stato realizzato un braccio meccanico in grado di replicare l'accelerazione su un singolo asse di interesse.
L'intento del nostro progetto su questo meccanismo è fungere da base per l'elaborazione di un dispositivo a tre assi utilizzabile in maniera estesa nei diversi campi presentati alla fine di questo scritto.

Organizzazione delle attività

Tale lavoro è stato così suddiviso:

- la prima fase ha riguardato la messa in funzione della piattaforma ArduMega, dispositivo facente parte del catalogo Arduino e la sua programmazione, l'implementazione del sistema ArduPilot concepito per la guida di UAV ad ala fissa ed il relativo pacchetto software per poter utilizzare la IMU Shield fornita di accelerometri e sensori giroscopici associata alla piattaforma.

- La seconda fase è stata dedicata alla realizzazione della replica d'assetto orizzontale della piattaforma utilizzando servomotori, attraverso i dati di accelerometri e giroscopi, e alla verifica delle potenzialità di tali sensori.

- Il terzo passo è constato nella scrittura del codice che funge da impalcatura al software finale, in cui è possibile scrivere una registrazione di accelerazioni e riaccedervi a posteriori senza utilizzare il Software HIL (in the loop).

- La parte più corposa del lavoro ha invece riguardato lo studio dei servomotori e la realizzazione di un modello Simulink realistico.

- L'ultima fase del lavoro è consistita nel completamento del software in grado di comandare un qualsiasi servomotore connesso alla piattaforma Arduino per replicare l'accelerazione registrata.

ARDUINO ED ARDUMEGA

Il progetto Arduino

Arduino è un microcontrollore di prototipazione open-source votato alla accessibilità sia in campo hardware che software. Il progetto nasce presso l'*Interaction Design Institute* dall'idea dei fondatori di permettere in maniera completamente libera di "customizzare" la piattaforma per scopi professionali o hobbistici: tutte le informazioni riguardanti lo schema elettronico sono pubblicate come "*Creative Commons license*", quindi disponibili e modificabili.
Stesso discorso vale per il corredo software, completamente *free* e scritto in codice Wiring, un linguaggio basato su C e C++, semplice ed intuitivo, che permette sia a principianti che a programmatori più esperti di sfruttare al meglio le potenzialità del microcontrollore. Inoltre è fornito un IDE (*integrated development environment*), ossia un ambiente di sviluppo integrato, e una libreria software per poter interagire con i pin connessi alle porte INPUT/OUTPUT , con la porta USB e con il regolatore di tensione.

ArduPilot Mega

ArduPilot Mega è un autopilota open source creato da Chris Anderson e Jordi Muñoz e poi sviluppato dalla comunità di DIY Drones. Basato su Arduino Mega e sull'elettronica ATmega2560, è in grado di controllare multicotteri, elicotteri, droni ad ala fissa e rover, e come Arduino adotta la filosofia dell' open-source.

Le piattaforme utilizzate per i nostri scopi è composta da due elementi hardware principali:

- Il microcontrollore della *main board* è come si è detto l'ATMega2560 della Atmel con CPU a 8-bit e con frequenza massima di 16 MHz, che combina memoria flash da 256KB, SRAM (*Static Random Access Memory*) da 8 KB ed EEPROM (*Electrically Erasable Programmable Read-Only Memory*) da 4 KB. E' presente inoltre un processore AT328 ausiliario, utile al *failsafe* in caso di malfunzionamento, che trasferisce il controllo dei pin all'autopilota se si verifica un break down del Radio Control System. E' fornito inoltre di 16 canali liberi divisi tra input e output che permettono la connessione di ulteriori sensori o di uscite controllate dal sistema.

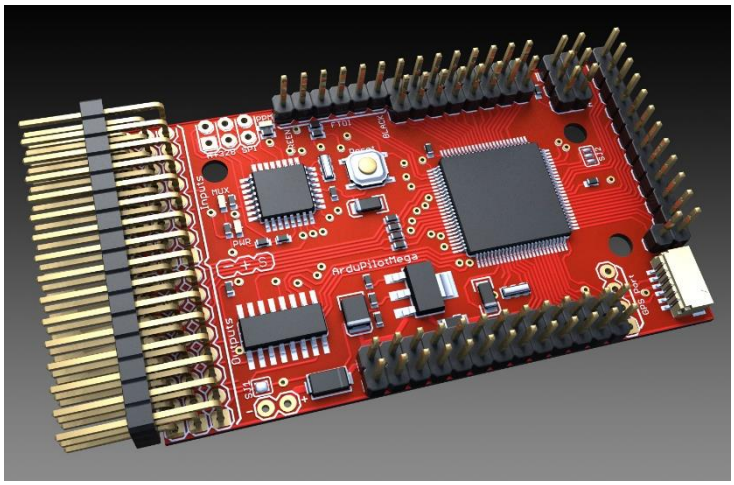


Figura 1 - ArduMega

- La scheda IMU shield (Inertial Measurement Unit) / OilPan presenta un'ampia gamma di sensori strettamente necessari in campo UAV o nella robotica, tra i quali giroscopi ed accelerometri sui tre assi, magnetometri, barometro e termometro. Attraverso questi dati si forniscono al velivolo una serie di informazioni per la determinazione dell'assetto e della traiettoria. Attraverso il calcolo procedurale della DCM (Direction Cosine Matrix Estimation) con una correzione in anello chiuso della deriva dei giroscopi si sfruttano le misure fornite dagli accelerometri effettuando una

forma di sensor data fusion meno onerosa dal punto di vista computazionale del ricorso a un filtro di Kalman, ed si è così in grado di ricostruire in ogni istante temporale l'orientamento del velivolo(roll, pitch e yaw) rispetto alla terna cartesiana di riferimento terrestre NED (North-East-Down). E' inoltre presente un'altra memoria flash da 4MB per il salvataggio dei data logging.

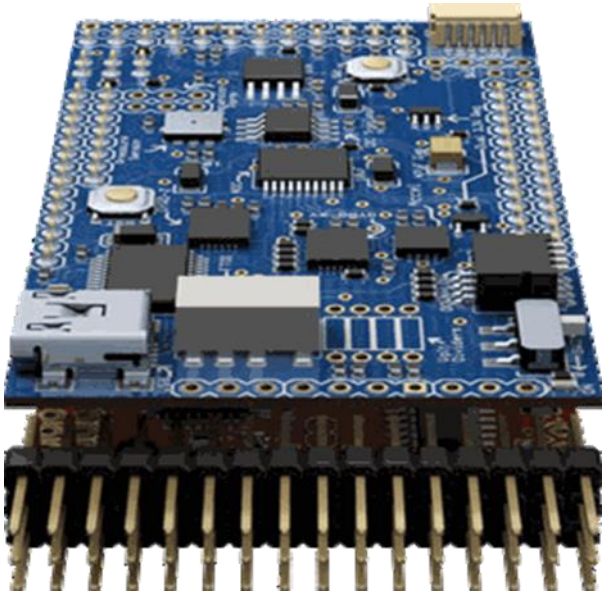


Figura 2 - IMU Shield

Firmware Ardupilot

Come anticipato la forza dell'autopilota ArduPilot è proprio nel firmware sviluppato in itinere all'interno dell'open-project DIY Drones. Infatti il codice, pur essendo abbastanza corposo non è complesso e si presta bene alla personalizzazione. Questo stesso pregio ha però creato delle difficoltà causate dall'utilizzo di una scheda ormai obsoleta: il continuo aggiornamento dei firmware

comporta la revisione di tutto il pacchetto di librerie necessarie al suo funzionamento; inoltre è necessario tenere conto della compatibilità con Arduino ed il suo ambiente di sviluppo, anch'esso in evoluzione. Ciò provoca una difficile reperibilità delle versioni più datate del firmware ed una sua incompatibilità con le versioni recenti di Arduino IDE e del software di Ground Control Station (GCS) (Centro di Controllo per le operazioni di programmazione e di missione di un UAV).

La problematica appena presentata ha costretto nel nostro caso a una sostituzione del microcontrollore: in un primo momento si era infatti utilizzato un ATMega1280. L'unico firmware ancora reperibile, compatibile con l'ambiente di sviluppo, risultava però essere troppo pesante per la capacità di memoria di tale scheda. Si è pensato quindi di andare ad eliminare parte del codice non utile al fine del nostro lavoro, ad esempio librerie e richiami alle funzioni legate ai sensori barometrici o ai magnetometri. Lo spazio memoria ricavato era però limitatissimo. Si è stati allora costretti alla sostituzione della piattaforma con una a memoria flash di 256 KB quando la porzione di programma aggiunta all'originale è diventata più importante ed ingombrante.

Il firmware elaborato dal processore per gestire le operazioni di autopilota deve essere infatti caricato all'interno della memoria del nostro dispositivo attraverso l'interfaccia USB presente sulla IMU Shield grazie al terminale CLI (Command Line Interface) all'interno dell'ambiente di sviluppo Arduino.

Replica di assetto

Una volta presa confidenza con il linguaggio di programmazione di Arduino e con il pacchetto software si è passati alla realizzazione della prima applicazione pratica

con il microcontrollore: attraverso l'utilizzo di due servi meccanici connessi alla Main board replicare l'assetto in elevazione ed inclinazione, cioè gli angoli di beccheggio (pitch) ed rollio (roll) della piattaforma.

Questa applicazione riprende la linea generale di una piattaforma stabilizzata o di un gimbal in ambito fotografico, nelle cui applicazioni si movimenta la piattaforma in modo da mantenere costante l'assetto della stessa rispetto ad un sistema di riferimento definito, contrastando la variazione di assetto del corpo a cui la piattaforma è vincolata.

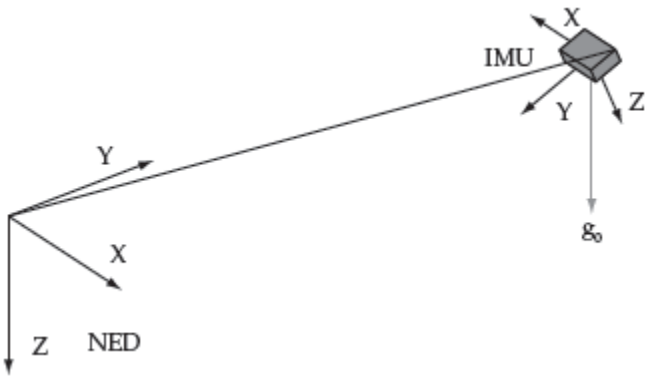


Figura 3 - Assi NED ed assi BODY

Sistemi di riferimento

Definiamo i sistemi di riferimento utili alla nostra analisi:

Il NED (North-East-Down) è un sistema mobile definito in riferimento al piano tangente alla superficie terrestre in un qualsiasi suo punto. In particolare,

prendendo l'origine del sistema su un punto della superficie terrestre, la terna di assi rimarrà solidale alla Terra e quindi può essere considerato come riferimento inerziale.

Nel nostro caso, la terna di assi NED ha le seguenti caratteristiche:

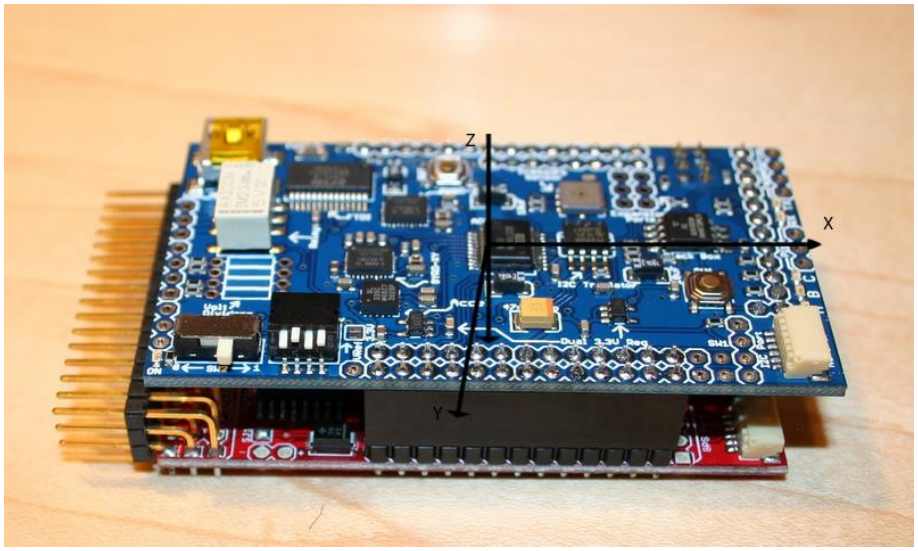
- Centro: nella posizione di home;
- Asse X: è diretto verso il Polo Nord geografico (asse North);
- Asse Y: è diretto verso Est (asse East);
- Asse Z: è perpendicolare agli altri due assi ed è rivolto verso il basso, quindi diretto verso il centro della Terra (asse Down).

Un altro sistema di riferimento molto importante è il sistema di Assi Corpo (Body). Esso è quello solidale al corpo a cui la piattaforma è vincolata ed ha le seguenti caratteristiche:

- Centro: nel baricentro del corpo
- Asse X: lungo l'asse longitudinale;
- Asse Z: giace sul piano di simmetria verticale e perpendicolare all'asse X;
- Asse Y: è l'asse trasversale perpendicolarmente agli assi X e Z in modo da formare una terna ortogonale.

Definiamo infine il sistema di riferimento mobile similare al Body ma solidale alla piattaforma (Assi Piattaforma) come mostrato in figura. Gli assi del s.d.r. coincidono con quelli dei sensori accelerometrici e giroscopici montati sulla scheda.

Figura 4 - Assi Piattaforma



Sensori Giroscopici

I giroscopi sono sensori che permettono di misurare la velocità di rotazione attorno ad un asse, detto asse di ingresso del giroscopio.

I principali tipi di giroscopi sono:

- meccanici (rotore che ruota: fenomeno della precessione e dell'inerzia giroscopica).
- ottici (Ring Laser Gyro (RLG), e Fiber Optical Gyro (FOG): effetto Sagnac)
- MEMS che si basano sugli effetti della forza di Coriolis su masse che si muovono

Considerando il sistema di riferimento body (b in figura) in rotazione rispetto ad uno inerziale assoluto (a in figura) i giroscopi misureranno le velocità $\omega_{b/a}$ di rotazione del primo rispetto al secondo.

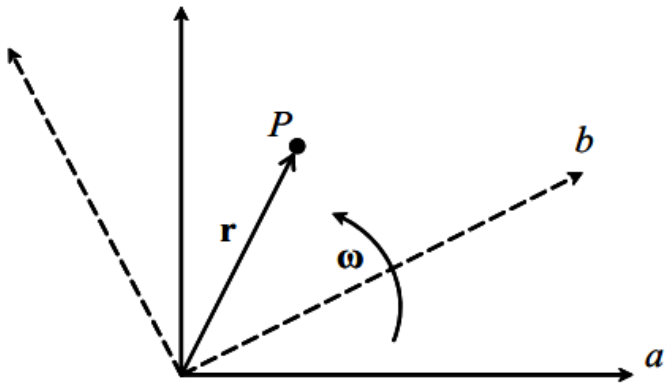


Figura 5 – Sistemi di riferimento

Realizzazione

Per riuscire a replicare l'assetto della nostra piattaforma sugli assi X e Y del sistema di riferimento NED fisso, quindi inerziale se consideriamo (come è lecito nella nostra applicazione) la terra piatta e non rotante, dovremo utilizzare dei servomeccanismi proporzionali comandati direttamente dalla scheda all'interno del firmware.

Tali servi hanno la caratteristica di essere comandati in posizione: ai diversi valori di tensione che possiamo applicarvi corrisponde una posizione angolare intermedia tra inizio e fine corsa. E' importante vedere come l'uscita dei pin di ArduPilot sia digitale e che il comportamento analogico sia ottenuto attraverso la tecnica del PWM (Pulse Width Modulation), in cui la durata degli impulsi, misurata in μs, è proporzionale al valore comandato.

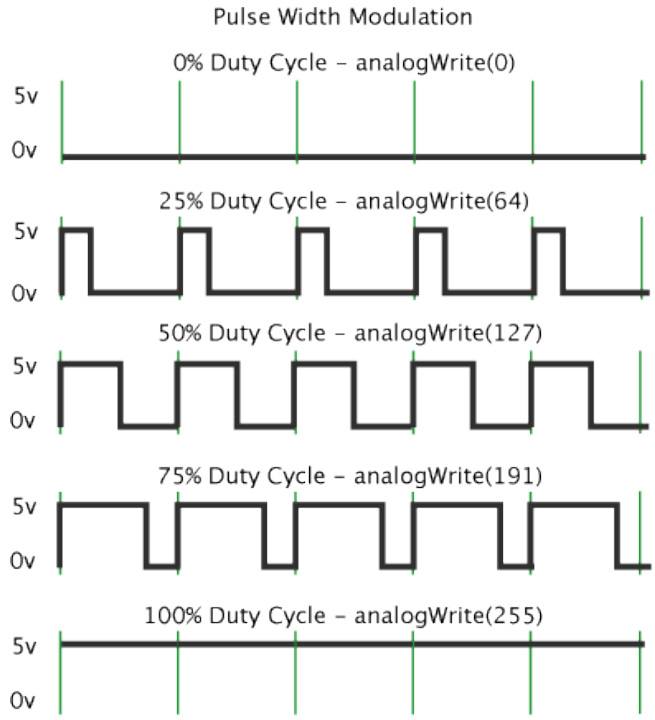


Figura 6 - PWM

E' necessario allora conoscere o valutare i valori m e q della funzione caratteristica del nostro servo:

$$\alpha = m * PWM + q$$

Noto il servo tali informazioni sono ricavabili dai datasheets del motore rilasciati dall'azienda produttrice.

Nella prima fase si sono utilizzati i dati registrati dai gyros: integrando le velocità di rotazione si ottengono le variazioni di angolo nell'intervallo di tempo considerato. I valori in uscita dai giroscopi hanno un modello del tipo:

$$\omega_m = \omega + d$$

con $\omega_m$vettore della velocità di rotazione misurata dai tre giroscopi ortogonali, ω vettore velocità di rotazione inerziale della terna giroscopica e d vettore delle tre derive. La presenza delle derive diventa particolarmente rilevante in periodi di funzionamento estesi in quanto l'integrazione di tali costanti porta ad ottenere dei risultati totalmente inesatti rispetto al valore atteso.

Si consideri ad esempio un giroscopio che si voglia impiegare per calcolare il valore, nel tempo, di una rotazione angolare $\vartheta(t)$ intorno ad un asse. Si supponga di conoscere il valore iniziale dell'angolo di rotazione all'istante iniziale zero, ϑ_0, e si assuma di misurare con un giroscopio la velocità di rotazione ω attorno all'asse. Si ha che vale

$$\dot{\vartheta}(t) = \omega(t) \ , \ \vartheta(0) = \vartheta_0$$

da cui

$$\vartheta(t) = \int_0^t \omega(\tau) d\tau + \vartheta_0$$

Abbiamo però dai giroscopi, considerando la deriva costante:

$$\omega_m(t) = \omega(t) + d$$

per cui

$$\vartheta_m(t) = \int_0^t \omega_m(\tau) d\tau + \vartheta_0 = \int_0^t (\omega(\tau) + d) d\tau + \vartheta_0 = \vartheta(t) + td$$

E' evidente che l'errore

$$\mathrm{e}_\vartheta(t) = \vartheta_m(t) - \vartheta(t) = td$$

si accumula nel tempo e raggiungerà valori sempre maggiori.

Nel nostro caso la variazione di angolo è ottenuto da un calcolo discreto

$$\Delta\vartheta = \omega_m(t) * GdT$$

in cui GdT non è altro che il tempo di elaborazione della funzione principale di ArduPilot. Infatti il firmware è strutturato in loop, o ripetizioni, a diverse velocità

di esecuzione all'interno delle quali vengono effettuati i processi necessari a calcolare i dati per il pilotaggio automatico.
La ripetizione più rapida (fast loop) viene effettuata a 50 Hz cioè ogni 0.02s ed al suo interno vengono, tra le altre funzioni, acquisiti i dati dalla IMU, aggiornata la matrice DCM e comandati i PIN in uscita. Il nostro tempo di integrazione è proprio quello della ripetizione del fast loop.
La variazione dovrà poi essere sommata alla posizione precedente, corvertita in PWM e comandata al servo.

Analizziamo ora le variazione fatte al codice ArduPilot.

File APM_Config.h

```
22.
23. #define TILT_SERVO 5
24. #define PAN_SERVO 6
25. #define PAN_RATIO 10.31
26. #define TILT_RATIO 10.31
```

In questa porzione di codice andiamo a definire i PIN di Output per il comando ai servomeccanismi (il 5 ed il 6) e la m (RATIO) precedentemente definita .

File ArduPilotMega.pde

Definizioni:

```
415.static void piattaforma_loop();
416.void replica( float);
417.boolean flag_piattaforma ;
418.float y [2] ;
419.float theta_p ;
420.float psi_p ;
421.int Tilt ;
422.int Pan ;
423.float theta_p_dot ;
424.float psi_p_dot ;
```

Nella funzione **void fast_loop()** portiamo i servo in posizione tale da avere gli assi coincidenti col NED e richiamiamo la funzione di replica

```
565.
566. if (flag_piattaforma)
567. {piattaforma_loop();}
568. else
569. {    APM_RC.OutputCh (TILT_SERVO, 1500 ) ;
570.     APM_RC.OutputCh (PAN_SERVO, 1500 ) ;
571.     flag_piattaforma=true;}
```

```
986.static void piattaforma_loop()
987.{replica( G_Dt);}
```

File replica.pde

```
1.  float Var_Time;
2.  int pwm_pause_tilt ;
3.  int past_Tilt;
4.  int new_Tilt;
5.  int pwm_pause_pan ;
6.  int past_Pan;
7.  int new_Pan;
8.  boolean init_attitude=0 ;
9.
10. void replica (float Var_Time)
11. {
12.     if (init_attitude);
13.     else
14.
    {past_Tilt = (int) ( ( ( float ) (dcm.roll_sensor ) ) / 100.0 );
15.         past_Pan = (int) ( ( ( float ) (dcm.pitch_sensor ) ) / 100.0
    );
16.         pwm_pause_tilt = 1500 + past_Tilt * TILT_RATIO;
17.         pwm_pause_pan = 1500 + past_Pan * PAN_RATIO;
18.         init_attitude = true;}
19.
20.     Vector3f gyro = imu.get_gyro( ) ;
21.     float p=gyro.x ;
22.     float q=gyro.y ;
23.     float r=gyro.z ;
24.
25.     Delta_Tilt = (int) ( ToDeg( p) * Var_Time );
26.     Delta_Pan = (int) ( ToDeg( q) * Var_Time );
27.
28.     new_Tilt= past_Tilt + Delta_Tilt;
29.     new_Pan= past_Pan + Delta_Pan;
30.
```

```
31.
    APM_RC. OutputCh (TILT_SERVO , pwm_pause_tilt + new_Tilt *TILT_RATIO
    ) ;
32.
    APM_RC. OutputCh (PAN_SERVO , pwm_pause_pan + new_Pan *PAN_RATIO) ;
33.     past_Tilt = new_Tilt;
34.     past_Pan = new_Pan;
35.
36.
37. }
38.
```

Come descritto in precedenza ciò che facciamo è calcolare la variazione di angolo ogni ripetizione del loop e correggere l'uscita del PIN associato al giusto servo. In questa funzione è stata aggiunta una sorta di inizializzazione per portarsi alla posizione iniziale della piattaforma: andiamo cioè a ricavare ϑ_0. Vengono poi acquisiti i dati dai gyros grazie alla funzione `imu.get_gyro()`,che convertiamo in archi sessagesimale e utilizziamo per i calcoli. Infine vengono comandati i servi grazie alla funzione `OutputCh()`ed aggiornati i valori di assetto per il loop successivo.

Come si è anticipato, per periodi di funzionamento protratti la deriva diventa importante: pur essendo le variazioni di assetto indotte sulla piattaforma compatibili con le escursioni angolari dei servi e nell'intorno del loro punto di funzionamento medio, si osserva come la posizione replicata vada ad accumulare errore.

A questo proposito si è pensato di confrontare le posizioni da noi calcolate con quelle più raffinate della DCM che attraverso il data fusion dovrebbe limitare l'azione della deriva. Abbiamo allora limitato la replica ad un solo asse implementando questa versione della funzione:

```
1. float uav_pitch;
2. float uav_roll;
3. float uav_yaw;
```

```
4.
5.  float Var_Time;
6.  int pwm_pause_tilt ;
7.  int pwm_pause_tiltDCM ;
8.  int past_Tilt;
9.  int new_Tilt;
10. boolean init_Tilt=0 ;
11.
12. void replica (float Var_Time)
13. {
14.     if (init_Tilt);
15.     else
16.
    {past_Tilt= (int) ( ( ( float ) (dcm.roll_sensor ) ) / 100.0 );
17.         pwm_pause_tiltDCM = 1500 + past_Tilt * TILTDCM_RATIO;
18.         pwm_pause_tilt = 1500 + past_Tilt * TILT_RATIO;
19.         init_Tilt = true;}
20.
21.     Vector3f gyro = imu.get_gyro( ) ;
22.     float p=gyro.x ;
23.     float q=gyro.y ;
24.     float r=gyro.z ;
25.     uav_pitch = ( ( float ) (dcm.pitch_sensor )) /100.0 ;
26.     uav_roll = ( ( float ) (dcm.roll_sensor ) ) / 100.0 ;
27.     uav_yaw = ( ( float ) (dcm.yaw_sensor ) ) / 100.0 ;
28.     if (uav_yaw < 0)
29.     uav_yaw += 360;
30.
31.     TiltDCM =(int) ( uav_roll ) ;
32.     Delta_Tilt = (int) ( ToDeg( p) * Var_Time );
33.
34.     new_Tilt= past_Tilt + Delta_Tilt;
35.
36.
    APM_RC.OutputCh (TILTDCM_SERVO , pwm_pause_tiltDCM + TiltDCM *TILTDC
    M_RATIO) ;
37.
    APM_RC. OutputCh (TILT_SERVO , pwm_pause_tilt + new_Tilt *TILT_RATIO
    ) ;
38.
39. past_Tilt = new_Tilt;
40.
41.
42.
43. }
```

In questo caso l'incremento dell'errore nel tempo, mentre la piattaforma viene movimentata, è quasi equivalente per le due modalità di comando dei servomotori. Una sostanziale differenza di comportamento si osserva invece quando, dopo aver accumulato errore si lascia ferma la piattaforma: il servo comandato attraverso

l'integrazione dei valori dei giroscopi mantiene l'errore, mentre quello movimentato attraverso i valori d'assetto della DCM lo smaltisce, grazie alle informazione ottenute dagli accelerometri, riportandosi quindi in posizione uguale a quella della piattaforma.

Logging dei dati

Una delle caratteristiche più interessanti per quanto riguarda le potenzialità di ArduPilot è quella di registrare i dati acquisiti dai sensori durante la fase di funzionamento e di poterli analizzare a missione conclusa. Iniziamo quindi una analisi delle modalità di acquisizione dati della piattaforma avvalendoci del suo firmware.

Il software caricato sul microcontrollore prevede 3 modalità di funzionamento:

- HIL DISABLED: i sensori sono tutti attivi e tutte le funzioni utili alla navigazione vengono richiamate all'interno del loop.
- HIL ATTITUDE: la modalità di hardware in-the-loop è attiva e il software di Ground Station simula direttamente i valori di assetto
- HIL SENSORS: modalità in cui i sensori sono emulati dalla Ground Station, per cui i veri sensori non sono attivi, ma tutto il carico computazionale è ancora da sopperire.

La modalità di nostro interesse è evidentemente quella in cui l'HIL è disattivato e in cui la IMU Shield fornisce valori reali così da avere l'opportunità di scegliere di salvare i tipi di dati, o logs, di proprio interesse tra cui quelli GPS, quelli di *attitude*, quelli riferiti al rendimento di propulsione, etc. I dati utili al nostro obiettivo sono quelli salvati in modalità RAW cioè i valori misurati dagli accelerometri e dai giroscopi. Attraverso il Terminal della Ground Station è poi possibile fare il download dei dati e visualizzarli.

La funzione di registrazione viene richiamata nel *fast loop* della funzione principale del firmware, per cui comincia non appena tutti i sensori hanno effettuato la loro inizializzazione. Ad ogni avvio del software viene creato un nuovo log, strutturato in funzione della tipologia di dato che andrà a contenere, i cui ad ogni ripetizione del main loop sarà aggiunto un pacchetto di dati.

Al fine di decodificare le diverse sessioni di registrazione ogni log contiene caratteri specifici che ne evidenziano inizio e fine (2 HEAD_BYTE ed 1 END_BYTE) e uno per caratterizzarne la tipologia. Queste accortezze sono utili alla funzione interna al software per separarli e leggerli in maniera corretta.

Dalla funzione di acquisizione dei log, *log.pde* possiamo andare a studiare in che tipo di variabile sono immagazzinati i dati. Nel caso di nostro interesse, cioè nel log RAW, accelerazioni e velocità angolare sono memorizzate in 2 vettori tridimensionali di componenti *long int*, (accel e gyro), dopo essere state moltiplicate per 10^7. La lettura e l'acquisizione a posteriore dei dati dovrà necessariamente avvenire su variabili compatibili, e per ottenere i valori misurati, sarà necessario questa volta dividere per 10^7.

Strutturata allora una idea generale del processo di acquisizione dei log possiamo sviluppare uno schema a blocchi che descriva una funzione del firmware di ArduPilot che rifletta le nostre intenzioni: acquisire in un intervallo di tempo desiderato un log di dati RAW e richiamarli a posteriori per confrontarli coi i valori attuali misurati dall'IMU Shield.

Notiamo come servano già, per realizzare questa prima schematizzazione, almeno 2 diversi comportamenti della piattaforma. Vogliamo però evitare, come nel caso delle MODE di HIL precedentemente descritte, di dover modificare e ricaricare il

firmware per variarne il comportamento. La nostra intenzione è proprio quella di realizzare un sistema pronto all'uso che l'utente non debba dover elaborare.

Per differenziare i comportamenti abbiamo a disposizione, già presente sulla IMU Shield, un bottone inutilizzato all'interno del firmware, da poter sfruttare (l'ingresso associato assume valore 1 (*high*) se premuto o 0 (*low*) in caso contrario). Siamo allora in grado di distinguere, dai dati della piattaforma, una pressione del tasto prolungata da una breve.

Altra accortezza che può ottimizzare il nostro progetto è quella di lavorare su un solo log, ad esempio il primo. In questo modo si eliminano completamente problematiche sulla scelta dello stesso tra quelli presenti in memoria semplificando di molto lo schema di funzionamento. Il log desiderato viene salvato nel primo slot di memoria ed i successivi, registrati durante la comparazione, negli slot seguenti. Per acquisire un nuovo log comparativo sarà allora necessario perdere il precedente: implementiamo, prima di una nuova registrazione, una pulizia automatica della memoria per non imbattersi in un esaurimento altrimenti inevitabile dello spazio memoria che richiederebbe un intervento sulla piattaforma.

Per rendere evidente in quale fase di funzionamento stiamo agendo sfruttiamo i led presenti sulla piattaforma: il led rosso andrà a caratterizzare la fase di acquisizione mentre quello verde la fase di comparazione.

Lo schema a blocchi sopra rappresentato sintetizza i ragionamenti fino ad ora espressi. Vogliamo realizzare un firmware in cui all'accensione la piattaforma effettui l'inizializzazione delle variabili e dei sensori senza però avviare l'acquisizione di log. Il salvataggio dati inizierà solo alla pressione del bottone. Per riuscire a differenziare il comando che porta ai due diversi blocchi è necessario però un altro controllo sul tempo di pressione del tasto.

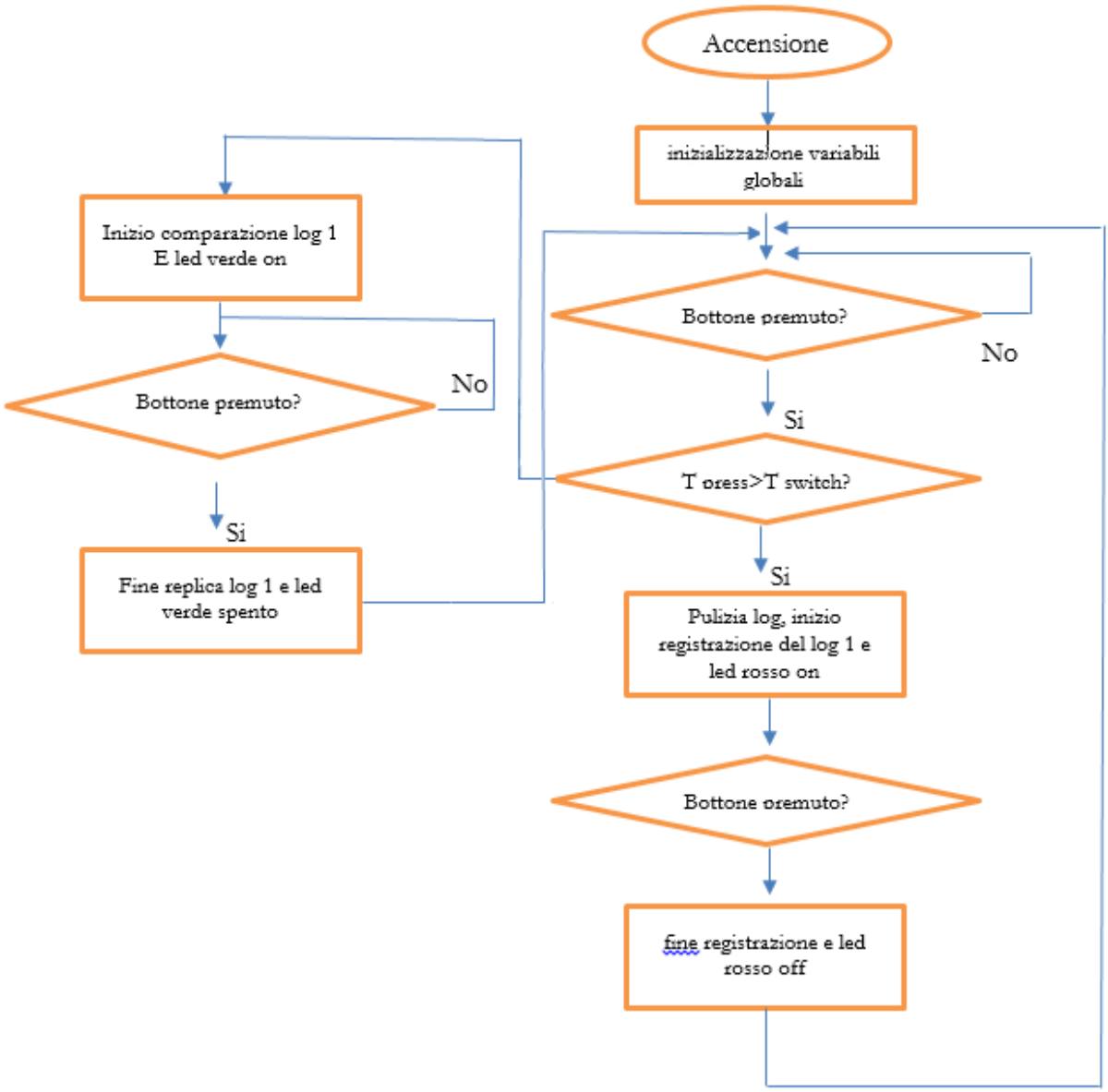


Figura 7 Schema a blocchi

Andiamo a tutelare la conservazione del log di comparazione da eventuali errori inserendo il comando di pulizia dopo la condizione di pressione prolungata del tasto.

Analizziamo le modifiche del firmware per integrare il funzionamento descritto:

```
480.int Bottone;
481.int Tpress;
482.static void comparazione_loop();
483.static void acquisizione_accelerazione();
484.void comparazione( float);
485.int flag_comparazione = 0 ;
486.
487.long tempobottone = 0;     // the last time the output pin was togg
    led
488.long debouncebott = 200;
489.long debouncebott_2 = 5000;
490.int previous = HIGH;
491.
492.boolean erase_log_flag = 0;
493.boolean limitilog = 0;
494.
495.int page_ext;
496.byte log_step = 0;
497.int packet_count = 0;
498.int packcount;
499.
500.#define HEAD_BYTE1 0xA3
501.#define HEAD_BYTE2 0x95  // Decimal 149
502.#define END_BYTE 0xBA   // Decimal 186
```

Interessanti tra queste definizioni sono quello di *tempobottone*, *debouncebottone* e *debouncebottone2*. La prima variabile rappresenta il tempo (dell'orologio interno) dell'ultima pressione del tasto, mentre le altre due la durata minima di pressione al fine di distinguere una digitazione breve da una prolungata: il primo tempo è di 0.2 secondi mentre il secondo è di 5 secondi.

Sono poi presenti le diverse variabili booleane che ci serviranno a effettuare le inizializzazioni necessarie.

```
587./*
588.  #if HIL_MODE == HIL_MODE_DISABLED
589.    if (g.log_bitmask & MASK_LOG_ATTITUDE_FAST)
590.
    Log_Write_Attitude((int)dcm.roll_sensor, (int)dcm.pitch_sensor, (uin
    t16_t)dcm.yaw_sensor);
591.
592.    if (g.log_bitmask & MASK_LOG_RAW)
593.      Log_Write_Raw();
594.  #endif
595.   */
```

E' stato poi necessario spostare la porzione di codice che scrive i log all'interno della funzione richiamata con la pressione del bottone. Nel codice modificato è stata resa invisibile dai simboli **/*…*/.**

```
645. Bottone = digitalRead(PUSHBUTTON_PIN);
646.
647. if ( previous == LOW )
648.  Tpress = (millis() - tempobottone);
649.
650. if (Bottone != previous)
651.    tempobottone = millis();
652.
653. if (Bottone == HIGH && previous == LOW ) //&&
    (millis() - tempobottone) > debouncebottone)
654.      { if (Tpress > debouncebott_2)
655.         {flag_comparazione = flag_comparazione - 1 ;}
656.       flag_comparazione = flag_comparazione + 2;
657.       erase_log_flag = 1;
658.       Tpress=0;
659.       if (flag_comparazione > 2 || flag_comparazione < 0 )
660.           {flag_comparazione = 0;}
661.      }
662.
663.     previous = Bottone;
664.
665.    switch(flag_comparazione
666.    {
667.      case 0:
668.                digitalWrite(C_LED_PIN, LOW);
669.                digitalWrite(A_LED_PIN, LOW);
670.
671.        break;
672.
673.      case 1:
674.
675.                acquisizione_accelerazione();
676.
677.
```

```
678.         break;
679.
680.       case 2:
681.                 comparazione_loop();
682.
683.         break;
684.             }
```

All'interno del fast_loop inseriamo la parte di programma che sensibilizza il bottone. La funzione della libreria di Input e Output *digitalRead()* dà in uscita 0 per bottone premuto e 1 per bottone non premuto: grazie a questa andiamo a salvare nella variabile *tempobottone* il clock dell'ultimo cambiamento di stato del bottone. *Tpress* invece ci valuta la durata della pressione.
E' poi strutturata in maniera più sintetica possibile l'indirizzamento alla giusta funzione in base alla pressione percepita.

```
1125. static void comparazione_loop()
1126.
1127.{
1128. # if HIL_MODE == HIL_MODE_DISABLED
1129.     if (g.log_bitmask & MASK_LOG_ATTITUDE_FAST)
1130.
    Log_Write_Attitude((int)dcm.roll_sensor, (int)dcm.pitch_sensor, (uint
    16_t)dcm.yaw_sensor);
1131.
1132.     if (g.log_bitmask & MASK_LOG_RAW)
1133.       Log_Write_Raw();
1134.  #endif
1135.
1136.
1137. digitalWrite(A_LED_PIN, HIGH);
1138. digitalWrite(C_LED_PIN, LOW);
1139. comparazione(G_Dt);
1140.}
```

In condizione Comparazione la funzione all'interno della file principale effettua la scrittura dei log, accende il led desiderato e richiama la funzione esterna di comparazione.

```
1140.  static void acquisizione_accelerazione()
1141.
1142. {
1143.
1144.  if (erase_log_flag!=0)
1145.    {
1146.   Serial.printf_P(PSTR("\nErasing log...\n"));
1147.   for(int j = 1; j < 4096; j++)
1148.     DataFlash.PageErase(j);
1149.   DataFlash.StartWrite(1);
1150.   DataFlash.WriteByte(HEAD_BYTE1);
1151.   DataFlash.WriteByte(HEAD_BYTE2);
1152.   DataFlash.WriteByte(LOG_INDEX_MSG);
1153.   DataFlash.WriteByte(0);
1154.   DataFlash.WriteByte(END_BYTE);
1155.   DataFlash.FinishWrite();
1156.   Serial.printf_P(PSTR("\nLog erased.\n"));
1157.
1158.     erase_log_flag = 0;
1159.    }
1160.
1161.
1162.
1163.  # if HIL_MODE == HIL_MODE_DISABLED
1164.     if (g.log_bitmask & MASK_LOG_ATTITUDE_FAST)
1165.
    Log_Write_Attitude((int)dcm.roll_sensor, (int)dcm.pitch_sensor, (uin
    t16_t)dcm.yaw_sensor);
1166.
1167.     if (g.log_bitmask & MASK_LOG_RAW)
1168.       Log_Write_Raw();
1169. #endif
1170.
1171.   digitalWrite(C_LED_PIN, HIGH);
1172.     digitalWrite(A_LED_PIN, LOW);
1173.
1174. }
1175.
1176.
```

In condizione acquisizione invece viene effettua la pulizia dei log, si accende il led desiderato ed inizia la scrittura dei log.

```
1.  #define HEAD_BYTE1 0xA3
2.  #define HEAD_BYTE2 0x95  // Decimal 149
3.  #define END_BYTE 0xBA   // Decimal 186
4.
```

```
5.
6.  float Var_Time;
7.  float reg_log[6] ;
8.  float now_acc[3] ;
9.  float now_gyro[3] ;
10.
11. boolean firstlog = 0;
12.
13. int dump_log_start2;
14. int inizio_log;
15.
16. int dump_log_end2;
17. int fine_log;
18. int vpr
19.
20.
21.
22. void comparazione (float Var_Time)
23.    {
24.
25.
26.
27.     if (!limitilog)
28.
29.      {
30.       get_log_boundaries(replica_log, dump_log_start2, dump_log_end2
    );
31.       inizio_log = dump_log_start2;
32.       fine_log = dump_log_end2;
33.       limitilog = 1 ;
34.       }
35. if (page_ext <= fine_log)
36.  {
37.   Log_Read2(inizio_log, fine_log, acc_log);
38.    now_acc = imu.get_accel();
39.    now_gyro = imu.get_gyro();
40. for (vpr = 0; y < 3; y++)
41.     {
42.         Serial.print(now_acc[vpr] , DEC);
43.     Serial.print(comma);
44.   }
45. for (vpr = 0; y < 3; y++)
46.     {
47.     Serial.print(riferim[vpr] , DEC);
48.     Serial.print(comma);
49.   }
50.  }
51. else { flag_comparazione = 0 ;}
52.
53.
54.
55. }
56.
57.
58. void Log_Read2(int start_page, int end_page, float * riferimento)
```

```
{
  byte data;

    packcount = packet_count;

    if (!firstlog) {
      page_ext = start_page ;
      DataFlash.StartRead(start_page);
      firstlog= 1;}

   while (page_ext < end_page && page_ext != -
1 && packcount == packet_count ){
    data = DataFlash.ReadByte();

switch(log_step)     // This is a state machine to read the packets
      {
      case 0:
        if(data == HEAD_BYTE1) // Head byte 1
        log_step++;

        break;
      case 1:
        if(data == HEAD_BYTE2) // Head byte 2
          log_step++;
            else
          log_step = 0;
        break;
      case 2:
                if(data == LOG_ATTITUDE_MSG){
          Log_Read_Attitude();
          log_step++;

        }else if(data == LOG_MODE_MSG){
          Log_Read_Mode();
          log_step++;

        }else if(data == LOG_CONTROL_TUNING_MSG){
          Log_Read_Control_Tuning();
          log_step++;

        }else if(data == LOG_NAV_TUNING_MSG){
          Log_Read_Nav_Tuning();
          log_step++;

       }else if(data == LOG_PERFORMANCE_MSG){
         Log_Read_Performance();
         log_step++;

       }else if(data == LOG_RAW_MSG){
                    Log_Read_Raw2(riferimento);
         log_step++;

       }else if(data == LOG_CMD_MSG){
         Log_Read_Cmd();
         log_step++;
```

```
112.
113.          }else if(data == LOG_CURRENT_MSG){
114.            Log_Read_Current();
115.            log_step++;
116.
117.          }else if(data == LOG_STARTUP_MSG){
118.            Log_Read_Startup();
119.            log_step++;
120.          }else {
121.            if(data == LOG_GPS_MSG){
122.              Log_Read_GPS();
123.              log_step++;
124.            }else{
125.
    Serial.printf_P(PSTR("Error Reading Packet: %d\n"),packet_count);
126.              log_step = 0;  // Restart, we have a problem...
127.            }
128.          }
129.          break;
130.        case 3:
131.          if(data == END_BYTE){
132.             packet_count++;
133.          }else{
134.
    Serial.printf_P(PSTR("Error Reading END_BYTE: %d\n"),data);
135.          }
136.          log_step = 0;       // Restart sequence: new packet...
137.          break;
138.        }
139.            page_ext = DataFlash.GetPage();
140.            }
141.
142.}
143.
144.
145.
146.
147.void Log_Read_Raw2(float * riferim)
148.{
149.  float logvar;
150.    for (int y = 0; y < 6; y++)
151.    {
152.    riferim[y] = (float)DataFlash.ReadLong() ;
153.        Serial.print(riferim[y] , DEC);
154.    Serial.print(comma);
155.
156.
157.  }
158.
159.}
160.
```

In questa funzione andiamo a stampare sul serial monitor i valori del log registrato e quelli acquisiti in quel loop. Per far questo è necessario definire i limiti del log 1 e la sua tipologia. In questo ci aiutano come visto i caratteri sopra citati.
Sfruttiamo inoltre la funzione già presente all'interno del file Log.pde *get_log_boundaries()*.

Da questo punto del lavoro in poi, per evitare di caricare in maniera eccessiva il processore di ArduMega, la scrittura dei dati registrati in tempo reale dai sensori sulla memoria flash è stata sostituita da una stampa su serial monitor. Infatti i continui salti all'interno della memoria provocavano pesanti rallentamenti nel fast loop ed in alcuni casi il *crash* del sistema. Mantenendo invece una lettura lineare dei dati del log registrato ed effettuando in parallelo la comunicazione attraverso la porta USB si è snellito il carico computazionale della piattaforma riuscendo a sviluppare in maniera ulteriore il nostro progetto.
Con l'ausilio di MathLab siamo in grado, attraverso dati tabulati, di graficare l'andamento dei dati d'interesse che prima venivano elaborati dalla software di Ground Station.

SERVOMOTORI

Studio generale

Iniziamo ora con lo studio delle tipologie di servo e le loro potenzialità.
Categorizziamo i motori che è possibile comandare attraverso la piattaforma Arduino in due tipologie che ne condizionano il funzionamento nella nostra applicazione:

- i servomotori stepper o a velocità costante
- i motori a velocità variabile

L' analisi si concentrerà sulla prima tipologia cercando però di mantenere una generalità che permetta di utilizzare il modello in entrambi i casi.

Il nostro obiettivo è quello di costruire un sistema che sia in grado di replicare accelerazioni associate a vibrazioni. Quindi l'input del nostro modello dovrà essere una forma d'onda in accelerazione. Il sistema operativo ha il compito di calcolare il valore di posizione (o velocità nel secondo caso) che realizzi in uscita un'accelerazione più vicina possibile a quella in ingresso.

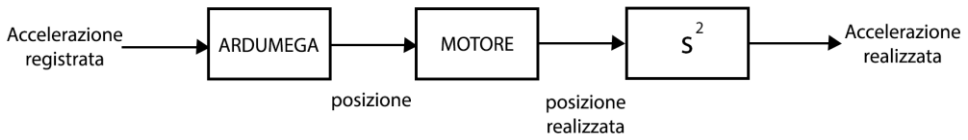


Figura 8 - Sistema

Il blocco cruciale di questo modello è quello che simula il motore, che insegue un comando in posizione. Dalla posizione effettivamente realizzata è poi facile risalire

all'accelerazione a cui è stata sottoposta la piattaforma: nella figura è stato rappresentato il passaggio con un blocco che richiama la funzione di trasferimento s^2, che richiama una doppia derivazione.

Partiamo dalla realizzazione in Simulink del modello a catena aperta che simula il comportamento del nostro apparato. Prendiamo per semplicità un input in accelerazione di tipo sinusoidale.
La relazione che lega l'accelerazione percepita dal nostro sensore cioè quella inerziale e la spostamento è:

$$a = -\ddot{p}$$

Nel nostro caso essendo lo spostamento su un arco di circonferenza per costruzione del meccanismo avremo

$$a_y = -a_t = -\ddot{\theta} r$$

dove θ è l'angolo percorso dal nostro sensore con

$$a(t) = A\sin(\omega t + \frac{\pi}{2})$$

Avremo quindi che la posizione angolare che realizza tale accelerazione per $v(0) = 0$ e $\theta(0) = 0$

$$\theta(t) = -\frac{1}{r}\iint_0^t a\, dt^2$$

Dove nel dominio delle frequenze avremo

$$A(s) = \mathcal{L}\big(a(t)\big) = -s^2 \mathrm{r}\Theta(s)$$

da cui

$$\Theta(s) = -\frac{1}{rs^2}A(s)$$

Otteniamo la posizione da dare in pasto al nostro motore per realizzare l'accelerazione in ingresso. Si vedrà come implementare questo tipo di funzione in maniera discreta all'interno del software nelle successive sezioni.

Andiamo ora ad analizzare il blocco MOTORE. I caratteri limitanti del meccanismo sono in linea più generale 3 :

- l' escursione angolare massima
- la velocità angolare massima e minima
- la tipologia di motore

Il primo punto è riferito principalmente ai servomotori che solitamente hanno una apertura angolare di funzionamento corretto di un centinaio di gradi. Per quanto riguarda i *velocità variabile* è molto più comune avere motori *brushless* in grado di realizzare la rotazione completa sull'asse comandato.

La seconda caratteristica influenza invece entrambi le tipologie. Se negli *stepper* la velocità è a livello ideale costante e massima, per i *variable-speed* il controllo è effettuato attraverso l'intensità di corrente che attraversa il motore: la limitazione superiore sarà funzione della massima potenza erogabile.
La soglia di velocità minima è invece introdotta dal tipo di controllo output di ArduMega. Come visto il comando in PWM discretizza la tensione in uscita applicata ai PIN e questo provoca una risoluzione discreta in posizione per gli *stepper* o in velocità per i *variable-speed.*

Introduciamo queste due limitazioni attraverso blocchi già presenti nella libreria Simulink applicandoli al dato di posizione angolare ottenuto integrando l'accelerazione.
La prima caratteristica è simulata con una saturazione che impedisce all'angolo di superare i limiti imposti: l'*upper* , cioè il superiore ed il *lower*, l'inferiore. Nel nostro caso tali valori sono stati fissati tra 0 e $-\pi/2$ in quanto si suppone che il servo parta dalla sua posizione massima.

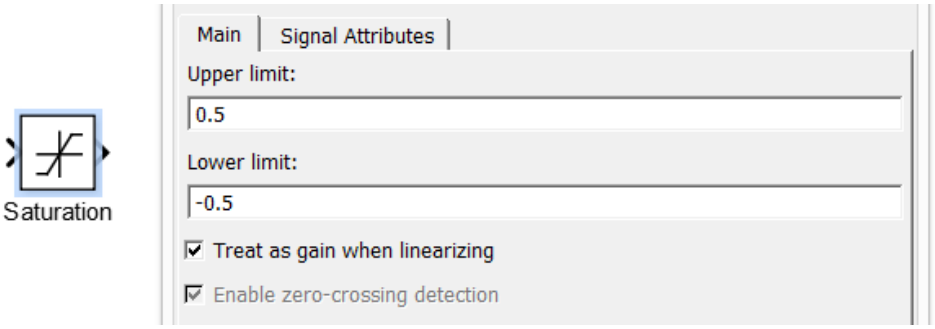


Figura 9 - Saturation

La seconda è riprodotta da un blocco Rate Limiter, utile a definire una velocità angolare massima e minima. La derivata dell'uscita viene infatti limitata superiormente ed inferiormente: se il dato in input non rispetta tali condizioni avremo in output un andamento a derivata massima o minima compatibile con le limitazioni imposte dal blocco.

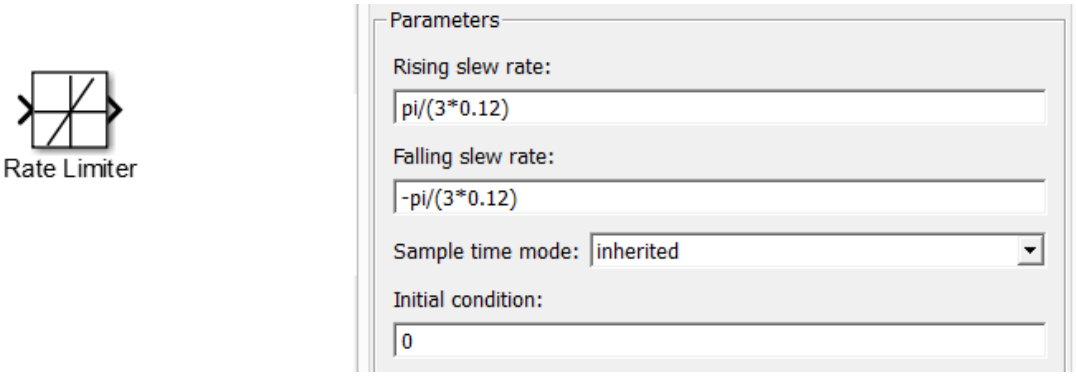


Figura 10 – Rate Limiter

Per quanto riguarda l'aspetto della limitazione legata alla discretizzazione di movimento e velocità sarà necessario un'analisi ulteriore e più approfondita nei paragrafi a seguire sulla base di prove sperimentali effettuate sui motori.

Una osservazione ulteriore valida per entrambe le tipologie di motore, nella riproduzione delle accelerazioni in ingresso, è il ritardo legato ai tempi computazionali del processore della piattaforma che deve leggere i dati, elaborarli e comandare l'uscita per realizzare l'accelerazione richiesta. Il minimo ritardo possibile sarà quindi uguale al tempo di ripetizione del fast loop, che chiameremo G_Dt ossia 0.02 secondi.

Per riprodurre questo comportamento possiamo inserire un blocco ritardante già presente nelle librerie Simulink. Vedremo però che l'andamento descritto, nella specializzazione dello studio ai servomotori stepper è introdotto all'interno dei successivi blocchi. Per cui da questo punto in poi dell'analisi ci concentreremo proprio sulla natura di funzionamento di tale categoria.

Servomotori stepper

Supposti i primi due parametri non limitanti (escursione e velocità massima infinite) possiamo immaginare per un servomotore stepper ideale un andamento della posizione angolare realizzata rispetto a quella registrata e comandata come in figura 11.

Questo si muoverà sempre a velocità angolare massima. Una volta raggiunta la posizione comandata all'intervallo precedente si fermerà e attenderà un nuovo comando. Otteniamo quindi una spezzata, che se interpolata fornisce la curva di posizioni comandate ritardate proprio del G_Dt precedentemente definito.

Il motore a questo punto, portandosi alla posizione ottenutà sottoporrà la piattaforma a una accelerazione come visto uguale a

$$a_y = -a_t = -\ddot{\theta} r$$

Per cui derivando due volte la posizione angolare otterremo l'accelerazione ottenuta.

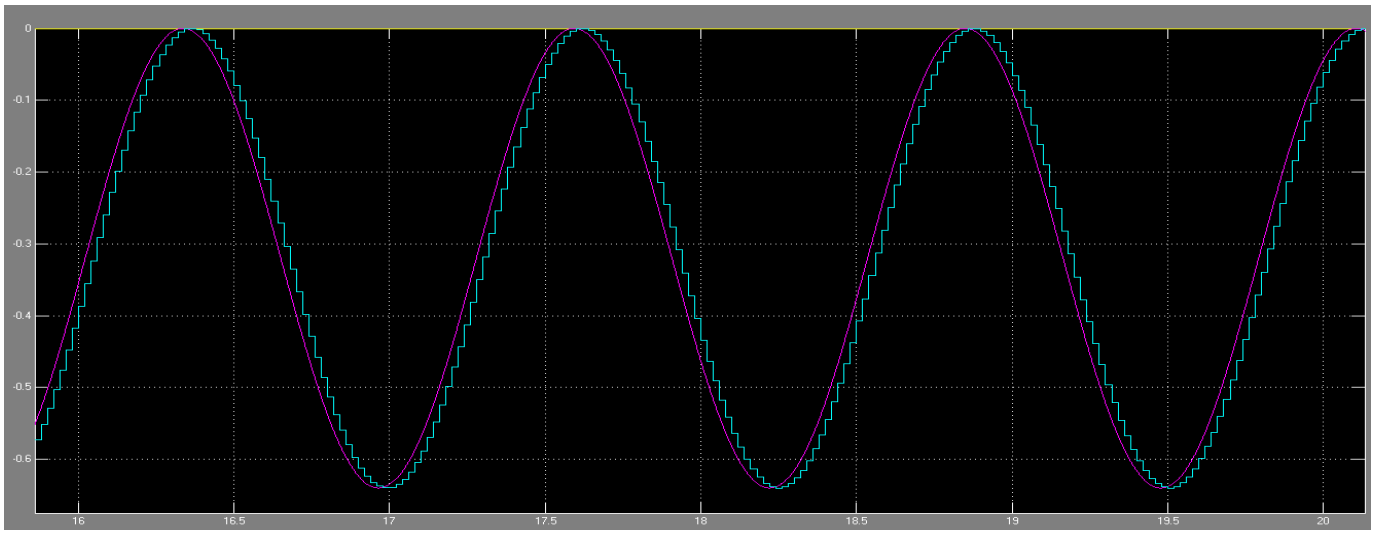


Figura 11 – Stepper ideale

Importante per la controllistica ed l'aspetto computazionale è la realizzazione del derivatore: non essendo una funzione $G(s) = s$ fisicamente sensata dovremo realizzarne una tipo

$$s/(c * s + 1)$$

già previsto nel blocco derivativo di Simulink con $c \to \infty$.

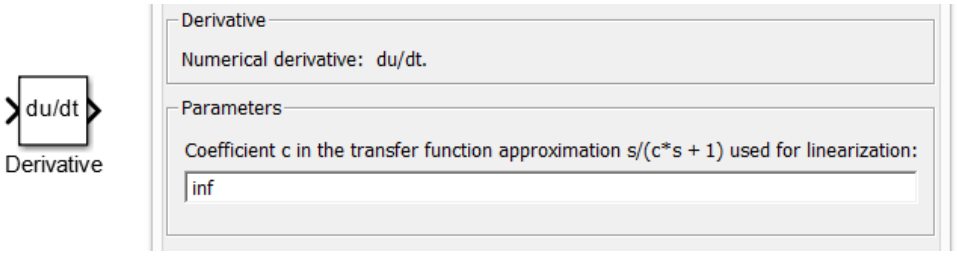


Figura 12 – Blocco Derivativa

Per ottenere una doppia derivazione con una funzione stabile abbiamo la possibilità di costruire un blocco di questo tipo:

$$\frac{-p1 * p2 * p3 * s^2}{(s-p1)(s-p2)(s-p3)}$$

con p1, p2 e p3 minori di zero e poi regolarne il guadagno.
Nel nostro caso, avendo una funzione stabile, possiamo scegliere di mettere due blocchi di derivazione in cascata.

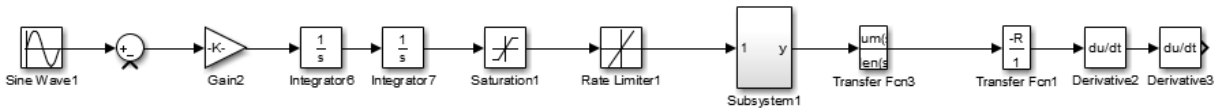


Figura 13 – Sistema Simulink

Il blocco caratterizzante ed il più complesso è il Subsystem mostrato in figura. Questo è stato studiato per replicare, in funzione dell'ingresso e di parametri propri del servo, la posizione realizzata dal motore.
Tale sottosistema è fondamentale per capire a pieno come utilizzare il motore. Infatti i risultati ottenuti rispetto al modello precedente sono parecchio scoraggianti. L'andamento a impulso della velocità provoca una serie di picchi di accelerazione che impediscono di realizzare una forma d'onda pulita come mostrato nelle figure 13 e 14.
Questa risultato non è però sperimentalmente vero: otteniamo valori di accelerazione non realistici in quanto abbiamo supposto la velocità del servo perfettamente impulsiva ed infinita.
Modelliamo allora un inseguimento della posizione a velocità costante ma limitata. Il sottosistema dovrà essere simile a un saturatore dinamico di posizione: raggiunta la posizione comandata il motore si dovrà fermare ed aspettare un nuovo comando.

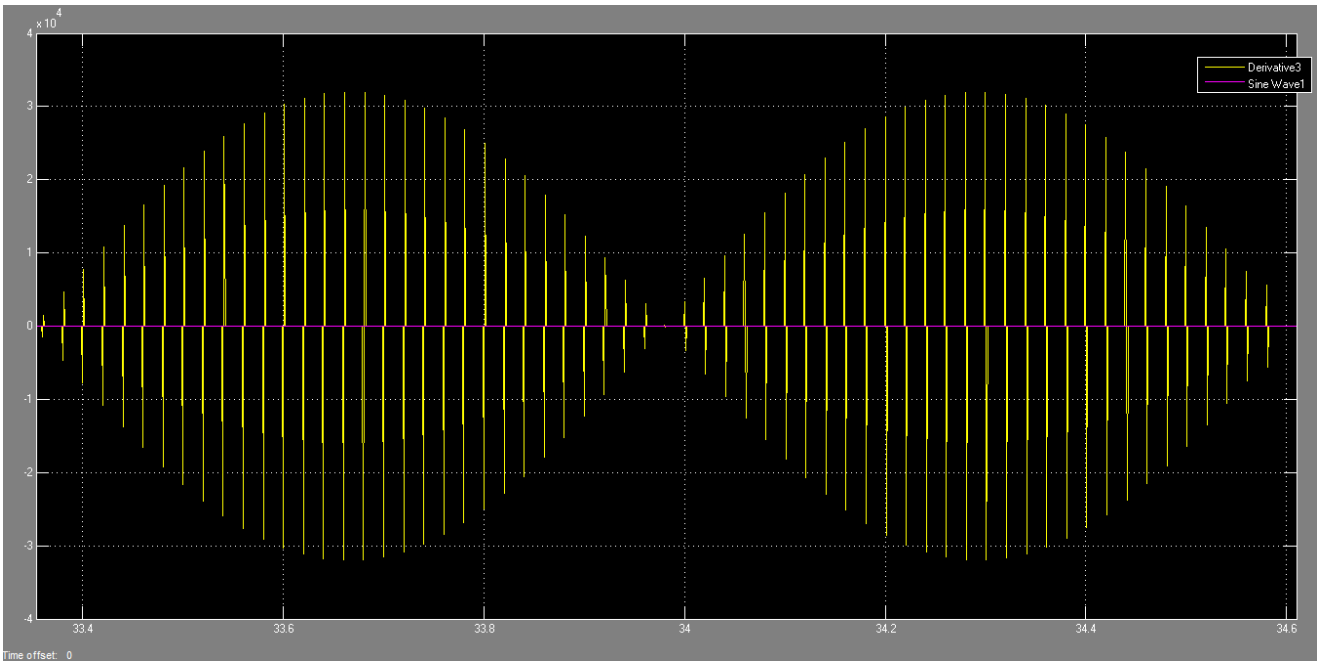


Figura 14 – Accelerazioni 1

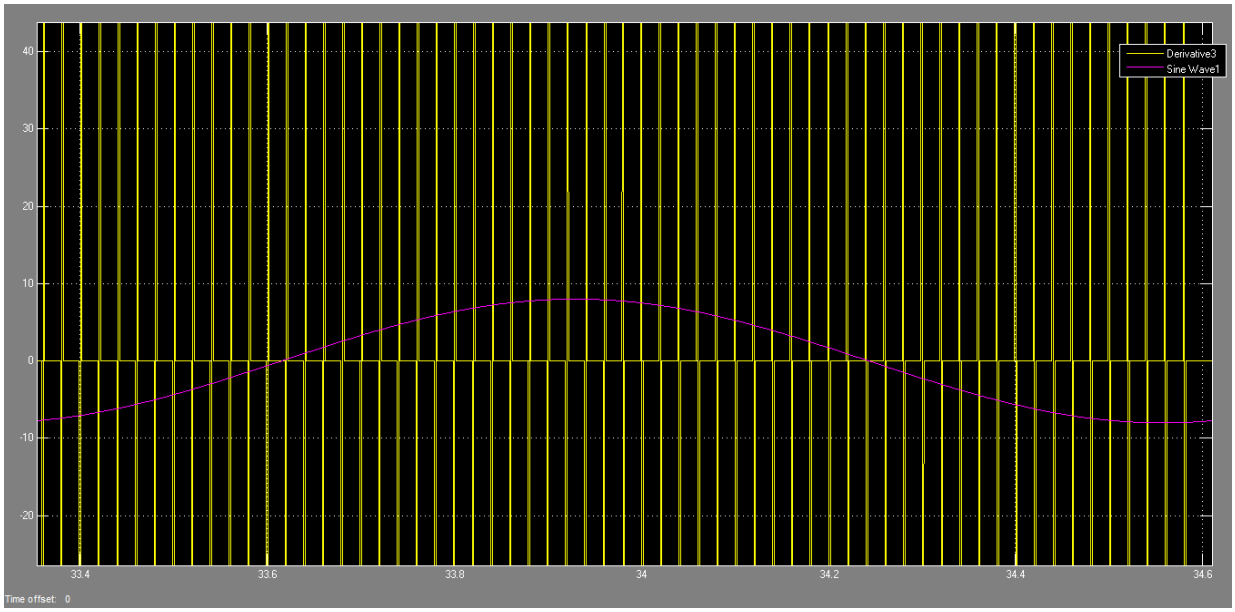


Figura 15 – Accelerazioni 2

Analizziamo la prima versione del sottosistema appena descritto:

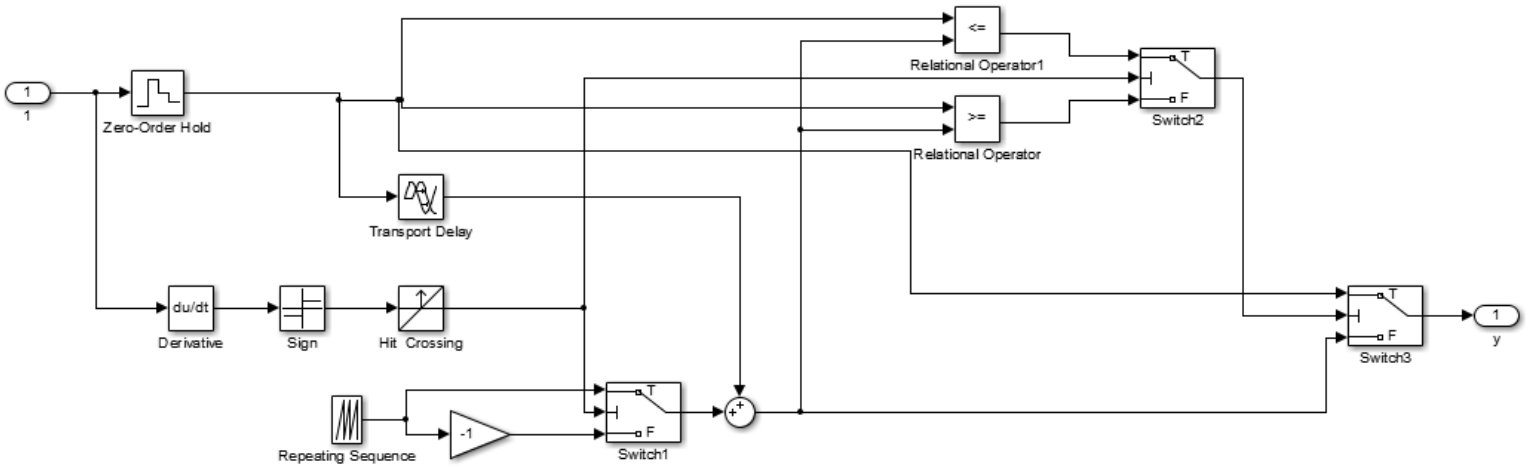


Figura 16 - Subsystem

Dall'input 1 entrano le informazioni di posizioni già limitata alle potenzialità sopradescritte del servo. Questi dati vengono

- discretizzati ad ordine zero
- usati per calcolare il segno della derivata

Il primo punto viene realizzato grazie allo Zero-Order Hold: in uscita il blocco fornisce l'interpolazione dei punti della funzione di input, campionati con frequenza uguale a quella definita nei parametri del blocco, cioè quella di elaborazione del sistema, con una curva di ordine zero, ovvero con linee orizzontali e verticali. E' evidente come tale blocco definisca un *delay* intrinseco nell'output.

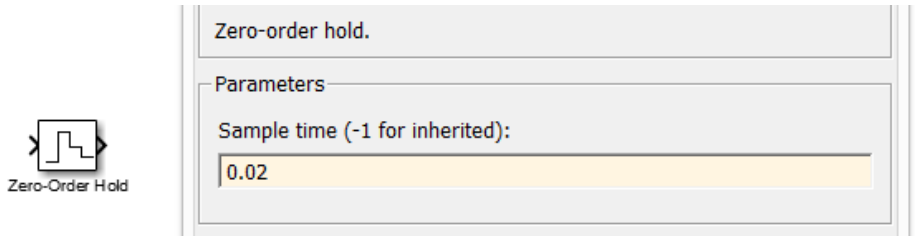


Figura 17 – Zero-Order Hold

La curva ridotta a ordine zero viene ritardata e sommata o sottratta in funzione del segno della derivata a un dente di sega a pendenza uguale alla velocità massima e di frequenza uguale a quella di comando. La funzione che ne risulta rappresenta le posizioni realizzabili dal motore tra i diversi punti di campionamento.

E' importante precisare che la compatibilità del motore con la curva di posizioni da cui tali punti sono stati ricavati è già stata verificata prima dell'ingresso nel sottosistema.

La somiglianza con un saturatore dinamico si sviluppa nell'ultima porzione del blocco, in cui la curva a dente di sega viene limitata superiormente o inferiormente (in base al segno della derivata dell'ingresso), con quella elaborata dal solo Zero-Order Hold: raggiunta la posizione campionata il motore come visto deve fermarsi e aspettare il comando successivo.

In Figura 18 è rappresentato il plot della curva di posizione prima e dopo il sottosistema, per una sinusoide di accelerazione di partenza compatibile con le caratteristiche del sistema.

Anche con questo tipo di andamento però le accelerazioni sono molto lontane dall'essere plausibili: abbiamo abbassato di un ordine di grandezza il modulo ma l'andamento è molto simile a quello visto nella Figura 14. E' necessario introdurre gli elementi che rendono il modello reale: transitori di velocità ed inerzie.

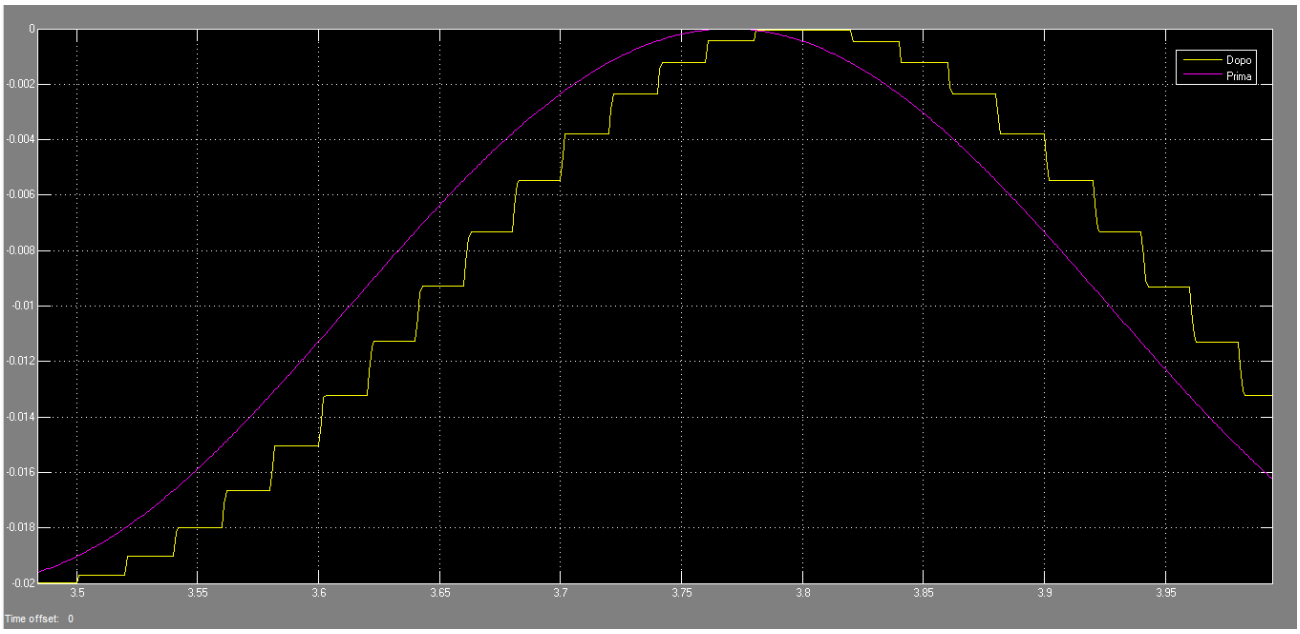


Figura 18 – Servo a velocità limitata

Prove sperimentali

Per avere una idea più precisa sul comportamento del servo e delle uscite è necessario effettuare qualche prova sperimentale. Attraverso i nostri test cerchiamo di valutare i tempi di transitorio della velocità e le oscillazioni derivanti dalle inerzie del sistema.

Nel primo test andiamo a comandare il servo da inizio corsa, posizione di riposo, fino a fine corsa. In questo modo il servo raggiungerà la massima velocità di cui è capace, mantenendola fino in fondo per poi arrestarsi.

E' molto interessante studiare i log di accelerazioni registrate in questa fase.

Osserviamo infatti che l'accelerazione ha una prima fase maggiore di zero, tra il campionamento 506 e 514, in cui la velocità aumenta; la seconda fase è quella di mantenimento della velocità massima, tra i campioni 514 e 517, quindi con

accelerazione costante uguale a zero; la terza fase è quella di stop alla posizione comandata fino alla completa immobilità tra 517 e 545.

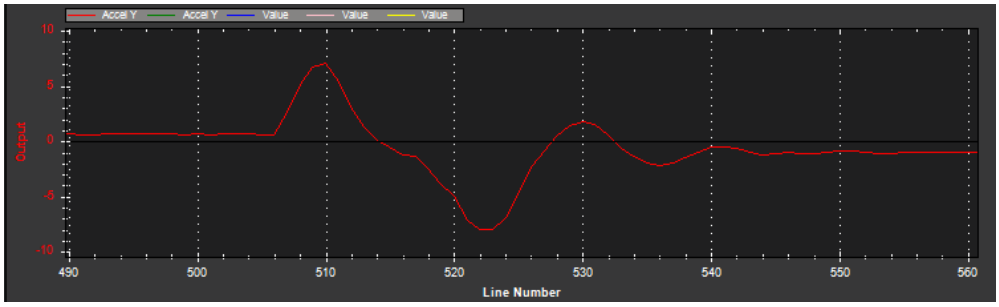


Figura 19 – Profilo di accelerazione

Sapendo che il campionamento avviene con frequenza pari a quella del sistema siamo in grado di calcolare il tempo di raggiungimento della velocità massima:

$$0.02 * (514 - 506) = 0.16\ s$$

Possiamo inoltre, integrando come mostrato nel paragrafo dello Studio generale, ottenere l'angolo percorso durante il transitorio.

Per ora il nostro interesse ricade sul tempo di transitorio T_trans. Volgiamo infatti modificare il dente di sega del precedente sottosistema con un ingresso che replichi in maniera più precisa l'andamento della velocità: vedremo nella successiva sezione come affrontare il problema.

L'altro test che è necessario fare riguarda proprio la possibilità di ottenere profili di accelerazione relativamente puliti nella movimentazione del nostro servo, senza che il continuo arrestarsi e ripartire del meccanismo provochi picchi tali da impedire di utilizzare un motore stepper per i nostri scopi.

Si è pensato allora di movimentare il servo comandandolo in posizioni linearmente crescenti a pendenze diverse.

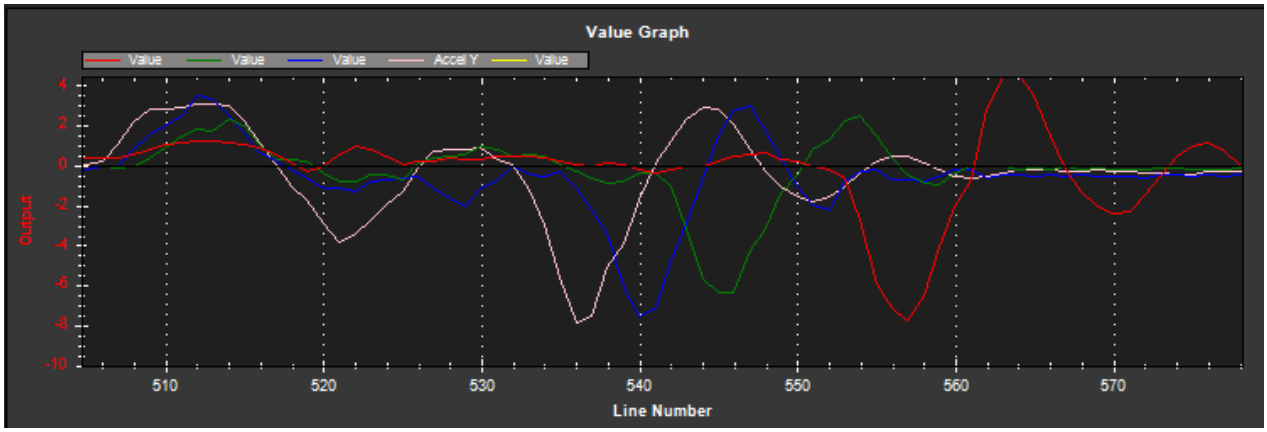


Figura 20 – Profili a diversi stepping

In Figura 20 vengono mostrati i dati registrati dai sensori per movimentazione a posizioni successive di 1, 2, 4 e 8 unità di PWM ogni comando, relativamente in rosso, verde, blu e rosa.

Ciò che troviamo subito positivo è la somiglianza con il profilo già analizzato: le fasi riconoscibili sono le stesse: velocità crescente, velocità costante ed arresto.

Chiaramente più rapido è l'aumento del PWM di comando, prima viene raggiunto il fondo corsa: la fase di arresto è traslata per i diversi test.

La fase iniziale si diversifica per la sola intensità delle accelerazioni percepite, mentre il tempo di realizzazione della velocità costante rimane circa lo stesso.

Nei due test ad aumento di PWM maggiore si nota però che il transitorio si protrae più a lungo che nelle altre due prove effettuate, pur riuscendo comunque a trovare un equilibrio.

Cerchiamo di spiegare questi risultati, cioè come sia possibile movimentare la il nostro braccio meccanico a velocità diversa dalla massima.

Grazie alle inerzie avremo, tutte le volte che la piattaforma viene arrestata, un rallentamento graduale e quindi, anche in questo caso, un tempo diverso da zero per raggiungere velocità uguale a zero. Alla stessa maniera il profilo accelerazione seguirà un andamento influenzato dalle inerzie e dalla velocità iniziale.
In funzione del diverso punto di equilibrio che si instaura in questo processo di accelerazione-decelerazione, relativamente piccole in modulo, otterremo velocità diverse. Riusciamo, secondo questa analisi anche a spiegare il maggior tempo di assestamento della velocità nei test a stepping più alto: nella prima fase di accelerazione in cui le velocità sono ancora basse il servo accumula ritardo rispetto alla posizione comandata e per annullare l'errore si porterà ad una velocità più alta di quella di equilibrio. Sarà allora necessario un rallentamento con conseguente decelerazione (vedi campionamento 520 del test a passo 8 pwm ed il 530 per il passo 4).
Dall'analisi effettuata consegue che un modello per il motore non è affatto semplice come poteva risultare da una prima analisi. Dovremo quindi elaborare il nostro sottosistema per renderlo compatibile con i comportamenti osservati nei test.

Modellizzazione

A livello ideale, la velocità del servomeccanismo dovrebbe essere impulsiva: partiamo allora direttamente da un treno di impulsi in velocità (anzichè usare una funzione in posizione) per poi modificarlo ed integrarlo per ottenere una simulazione più realistica.
Studiamo prima il blocco *Pulse generator* tralasciando dall'analisi la compatibilità dei dati immessi con le impostazioni temporali della simulazione, molto influenzate dalle esigenze e dalle preferenze dell'utente.
Questo blocco dà in uscita un treno di impulsi di cui è possibile definire intensità, periodo ed ampiezza della pulsazione in percentuale dello stesso.

Modificando questi dati potremo definire il punto di funzionamento definito nel paragrafo precedente.

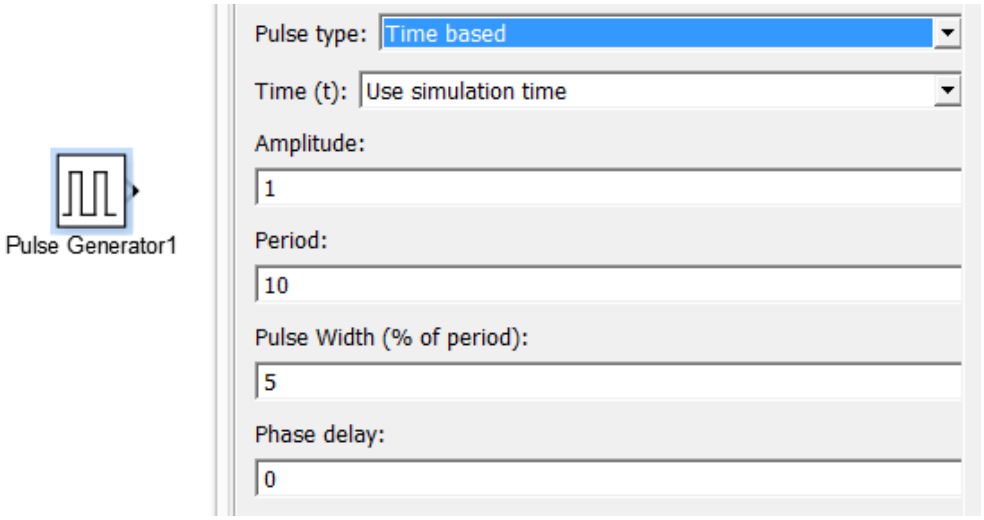


Figura 21 – Pulse Generator

Per introdurre le inerzie si applica al segnale una funzione di trasferimento retroazionata del second'ordine del tipo

$$G(s) = \frac{Wn^2}{(s^2 + 2 * delta * Wn * s + Wn^2)}$$

in cui i valori dei parametri *Wn* e *delta* derivano dalle osservazioni sperimentali presentate nel paragrafo precedente: scegliamo una *Wn* con cui la velocità massima viene raggiunta in tempi simili a quelli registrati (nel nostro caso i valori della variabile sono dell'ordine del 10^1). In figura viene mostrata la funzione prima e dopo il blocco del second'ordine. Osserviamo come la velocità massima si assesti ad un valore più basso della nominale del servo, il che risulta perfettamente

compatibile con i dati registrati nei test: dall'integrale dell'accelerazione della prima prova infatti otteniamo una velocità praticamente identica a quella simulata.

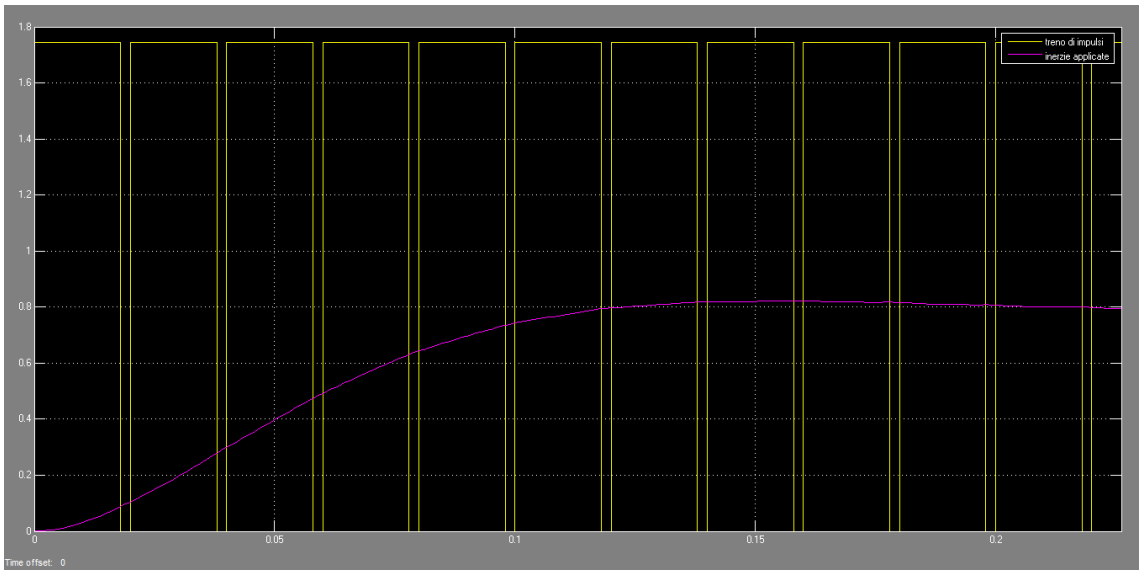


Figura 22 – Velocità di equilibrio

Osserviamo come variando il valore del Pulse Width del blocco vari anche la velocità a cui si stabilizza la funzione: ecco modellato il comportamento che definisce l'equilibrio di cui si è parlato in precedenza.

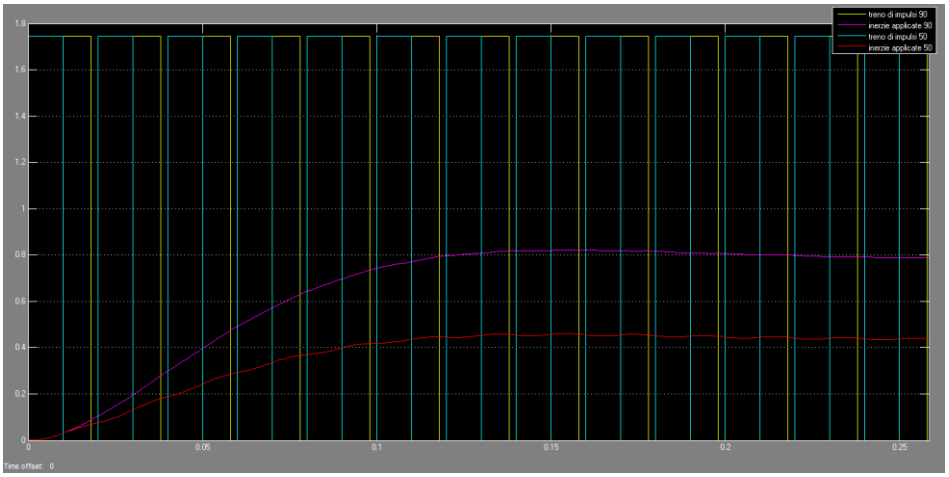


Figura 23 – Velocità di equilibrio 2

Nell' esempio in Figura 23 – Velocità di equilibrio 2 viene visualizzata la funzione dopo il blocco del secondo ordine per impulsi pari al 90% e al 50% del periodo: la prima si stabilizza sul valore di circa $0.8\ m/s$ mentre la seconda a circa $0.45\ m/s$. Per ora utilizzeremo nella simulazione la velocità massima pur essendo non conservativi per quanto riguarda le possibilità del motore.

Per ottenere lo spostamento in ogni intervallo è necessario integrare la funzione velocità elaborata, nell'intervallo considerato, tra i limiti che stiamo considerando, facendo la stessa operazione sulla funzione e sulla stessa ritardata dell'intervallo di integrazione e sottrarre la seconda alla prima svolgendo una operazione simile a:

$$\int_{-\infty}^{t} f(\tau)d\tau - \int_{-\infty}^{t-G\_Dt} f(\tau)d\tau = \int_{t-G\_Dt}^{t} f(\tau)d\tau$$

Otteniamo come risultato una curva che descrive lo spostamento possibile nel precedente intervallo di tempo di campionamento: miglioriamo la modellizzazione ideale utilizziamo questi valori come ampiezze della funzione a dente di sega.

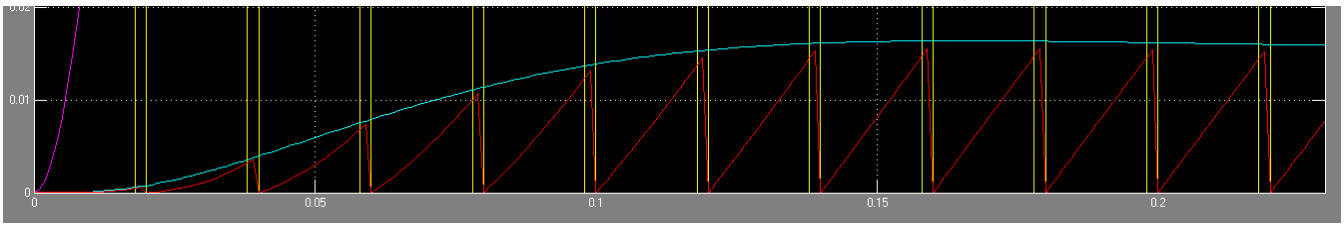


Figura 24 – Escursione angolare possibile

Realizziamo questa operazione moltiplicando un blocco Repeating Sequence di ampiezza uguale a 1 con l'uscita del blocco sommatore come mostrato in Figura 25. Il risultato sostituisce la funzione a dente di sega nello schema della Figura 16 - Subsystem.

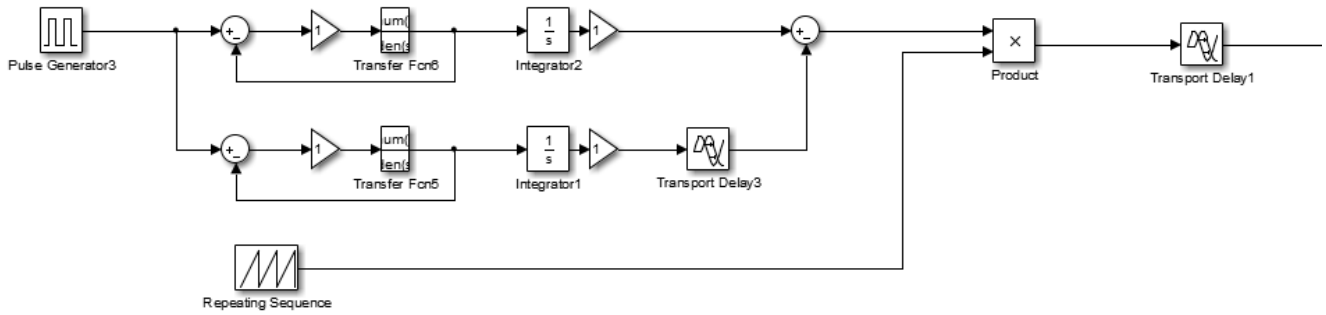


Figura 25 – Subsystem modificato

Il modello non è ancora completamente soddisfacente.

Stiamo infatti approssimando il movimento del servo come a velocità costante e uguale alla massima realizzata nel movimento: essendo l'accelerazione in questo caso una funzione sinusoidale lo dovrà essere anche la velocità (se pur sfasata di $-\pi/4$).

Supponendo ora validi i test e le simulazioni effettuate possiamo affermare che il servo è in grado di mantenere una velocità circa costante e diversa dalla massima.
Risultando inutilmente complesso in simulazione controllare la velocità modificando la durata degli impulsi del blocco *Pulse Generator*, decidiamo di variare l'ampiezza mantenendo costante il primo parametro. Facciamo questo utilizzando il blocco derivatore e lo Zero-Order Hold, ed elaboriamo la sequenza di impulsi di ampiezze diverse (Figura 26) con il solito blocco del secondo ordine.
In Figura 26 e Figura 27 possiamo osservare la velocità derivata dalla posizione in ingresso e quella elaborata dal modello.
In Figura 29 è mostrato la parte modificata del sottosistema che simula il servomotore, nella Figura 30 e Figura 31 le curve di posizione in ingresso ed in Figura 32 – Risultato complessivo in accelerazioneuscita al blocco Subsystem e nella quelle di accelerazione comandata ed ottenuta attraverso il nostro modello.

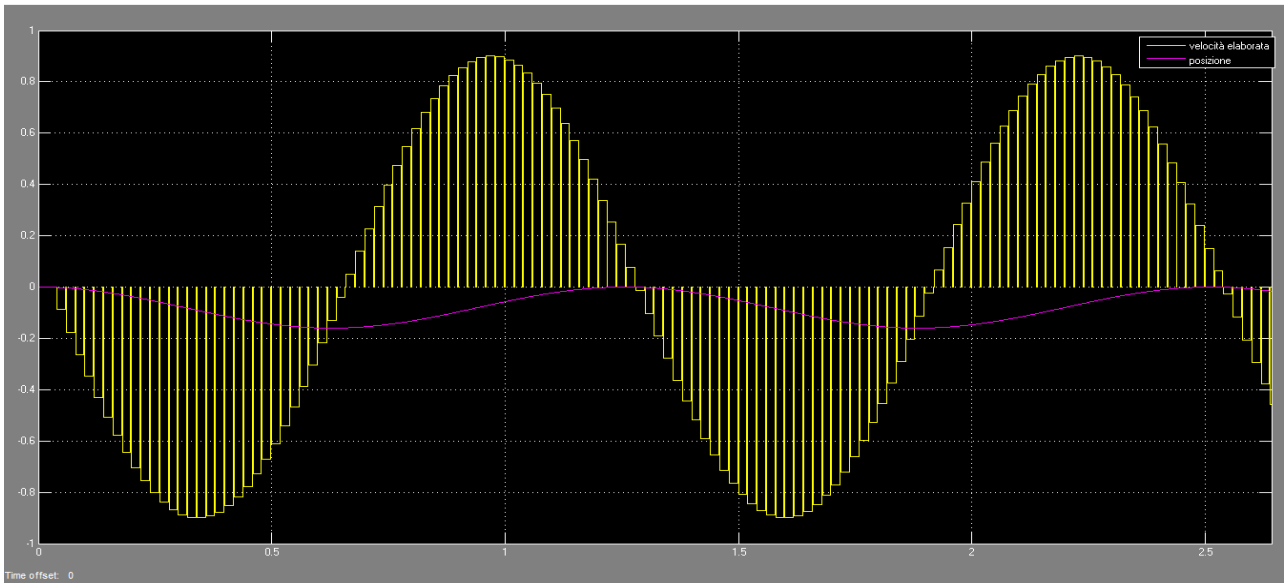


Figura 26 – Controllo sull'ampiezza

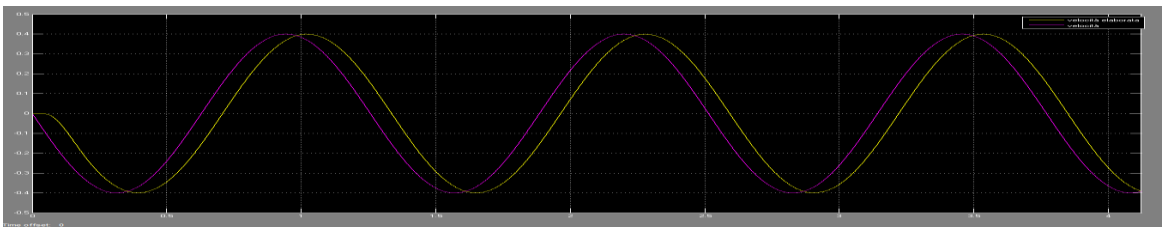


Figura 27 – Risultato complessivo in velocità

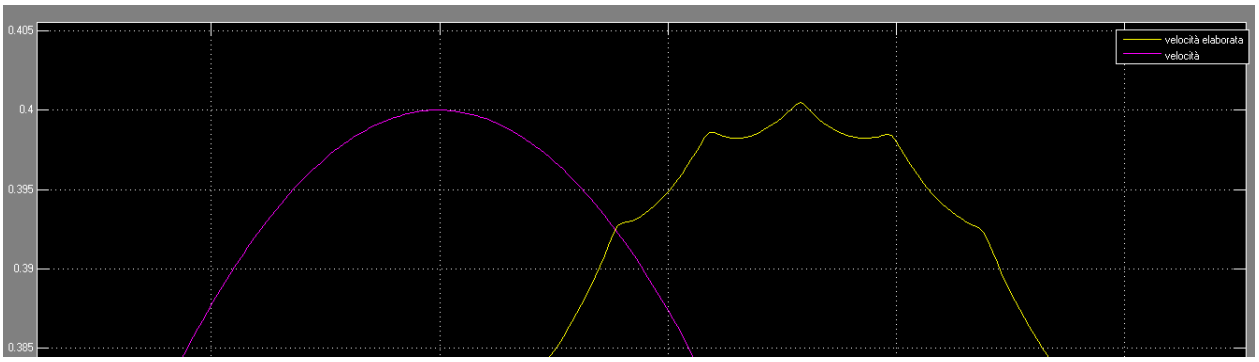


Figura 28 - Particolare

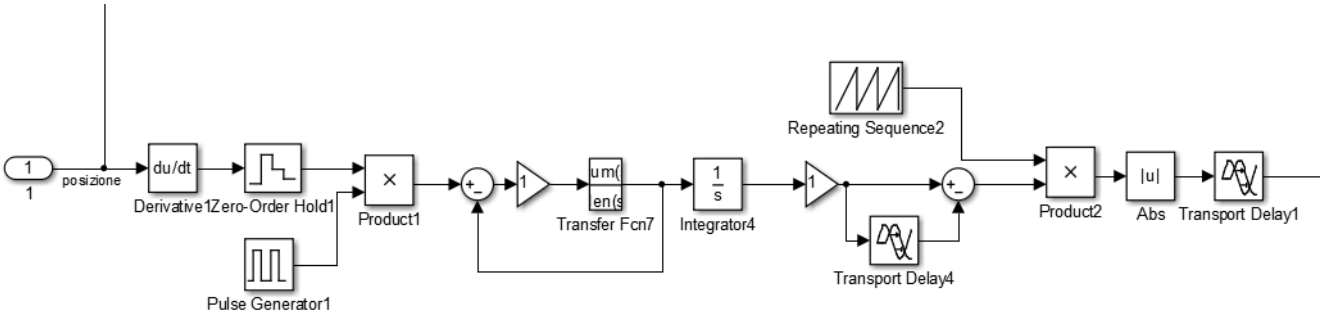


Figura 29 – Porzione di Subsystem modificata

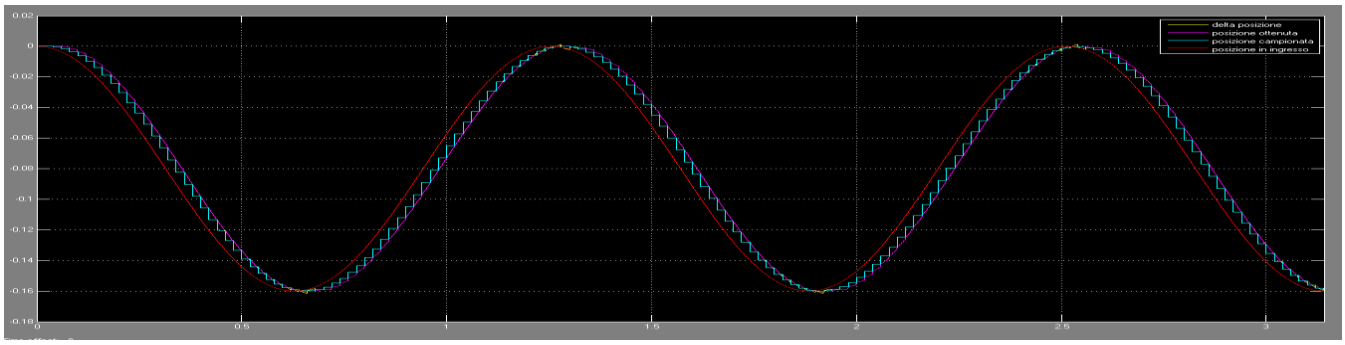


Figura 30 - Risultato complessivo in posizione

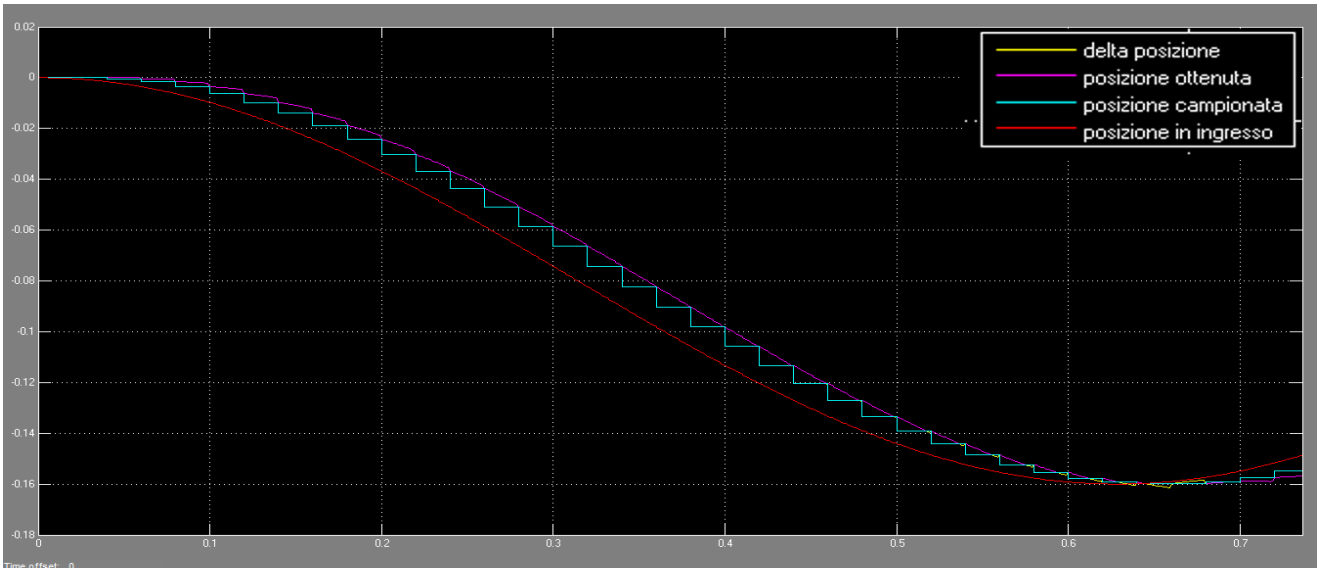


Figura 31 – Particolare 2

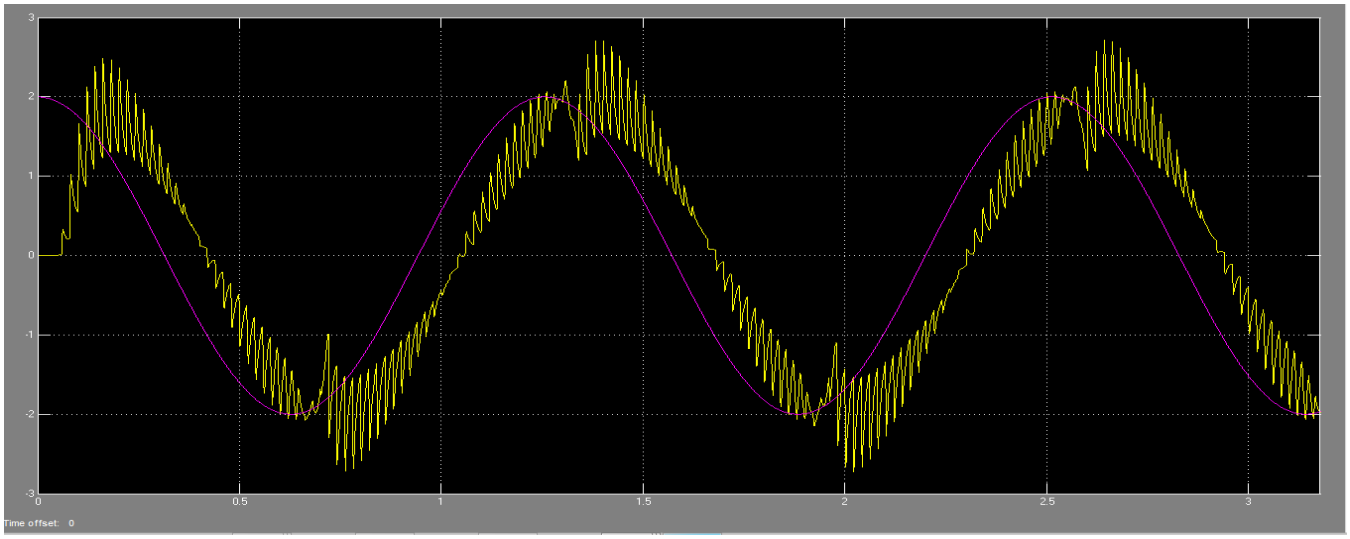


Figura 32 – Risultato complessivo in accelerazione

In Figura 30 - Risultato complessivo in posizioneosserviamo come la posizione ottenuta segua molto bene quella comandata con una curva ritardata rispetto alla prima ma non perfettamente sinusoidale. La tipologia di comando propria del servomotore introduce delle oscillazioni e delle discontinuità nelle derivate che è

causa dell'andamento dell'accelerazione ottenuta che osserviamo in giallo nella Figura 32.

Siamo però estremamente soddisfatti dei risultati ottenuti con questa simulazione: siamo riusciti a costruire una modellizzazione plausibile (le accelerazioni sono limitate e realistiche) e coerente con le osservazioni effettuate sulla base delle prove sperimentali.

Avendo una idea più precisa dello strumento con cui stiamo lavorando iniziamo a elaborare il software necessario a replicare in feedback l'accelerazione registrata.

SOFTWARE

Test rig

Per riuscire a validare il modello e perseguire gli obiettivi sopra descritti è necessario avere la possibilità di registrare un profilo di accelerazione semplice e preciso. L'ideale è proprio quello di avere un log che descriva un andamento sinusoidale: infatti, grazie all' analisi spettrale basata sugli sviluppi di Fourier, siamo in grado di generalizzare lo studio a un campo molto più ampio di profili di accelerazione.
Per avere una tale precisione ci affidiamo ad uno strumento adatto allo scopo:
l' Ideal Aerosmith Model 2102-22 Test Table (Figura 33).

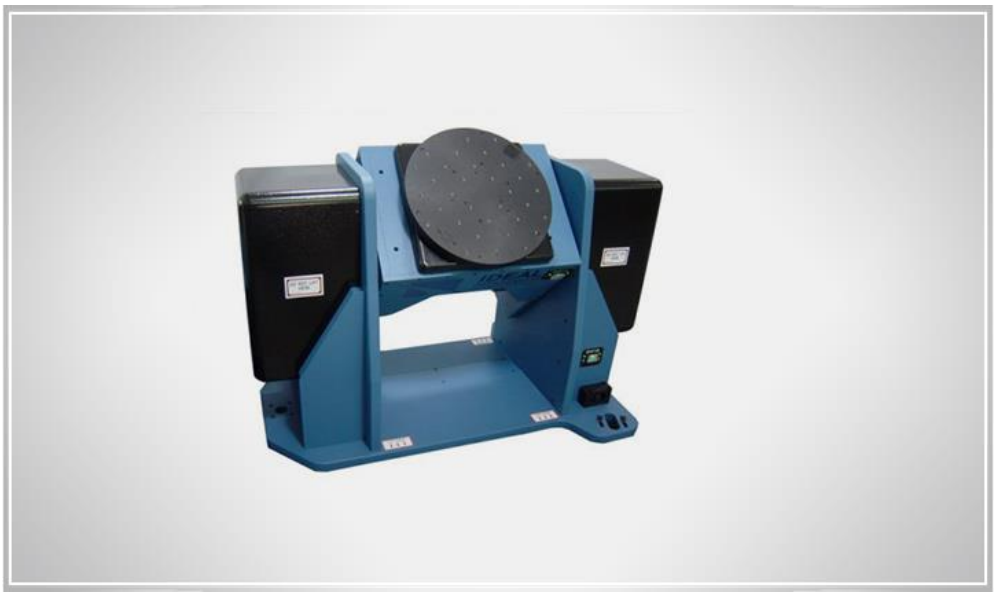


Figura 33 – Test rig

Questo strumento è progettato per sviluppare, collaudare e calibrare sensori di sistemi di navigazione (quali giroscopi, accelerometri e sistemi di navigazione inerziali) attraverso l'attuazione estremamente precisa dei suoi sistemi di

movimentazione in posizione e velocità. Attraverso il banco remoto AERO 900 Test Table Controller siamo infatti in grado di comandare la rotazione del banco di prova (con velocità fino a 300°/s) attorno ai 2 assi mostrati in Figura 34 - Schema di funzionamento Test Rig: l'inner axis con mobilità completa di 360° e l'outer axis limitato a $\pm 90°$.

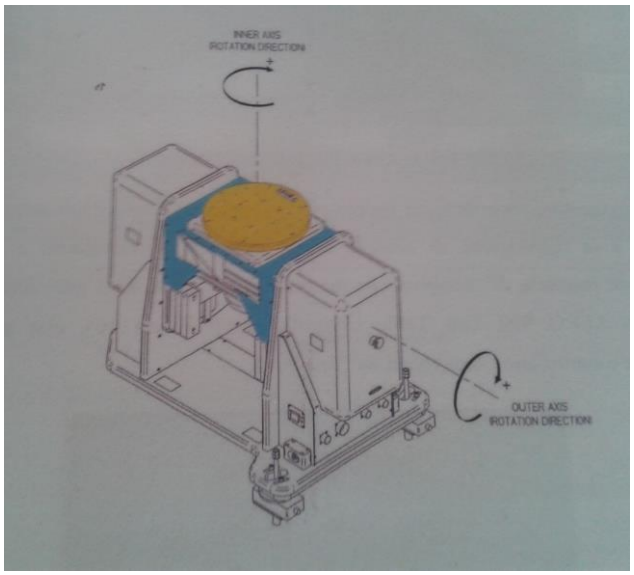


Figura 34 - Schema di funzionamento Test Rig

Risulta perfetto per il nostro scopo il comando già presente nel linguaggio ATL (*Aerosmith Table Language*) che permette di movimentare il banco con moto sinusoidale definendo ampiezza, periodo e numero di cicli della oscillazione.

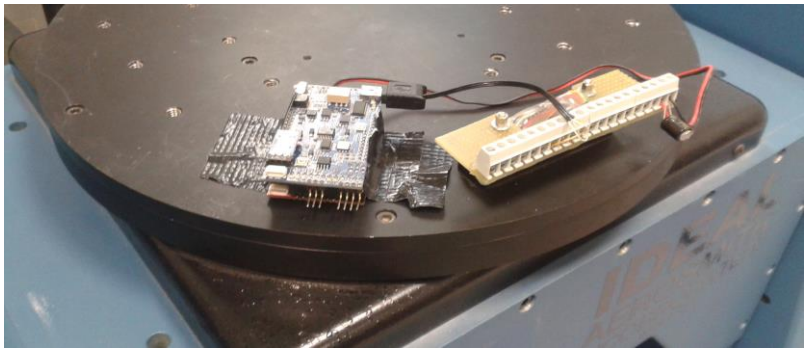


Figura 35 – Acquisizione Log

Posizionando la piattaforma sul disco rotante dell'*inner axis* come mostrato in foto si è acquisito un log abbastanza accurato da poter usare in seguito nel nostro lavoro. Osserviamo dalla visualizzazione dei dati salvati sulla flash che i valori calcolati dagli accelerometri, essendo comunque di tipo *RAW* , cioè non elaborati, risentono di errori: sono facilmente individuabili nella curva dei picchi di accelerazione solamente riconducibili ad imprecisioni dello strumento di misura.

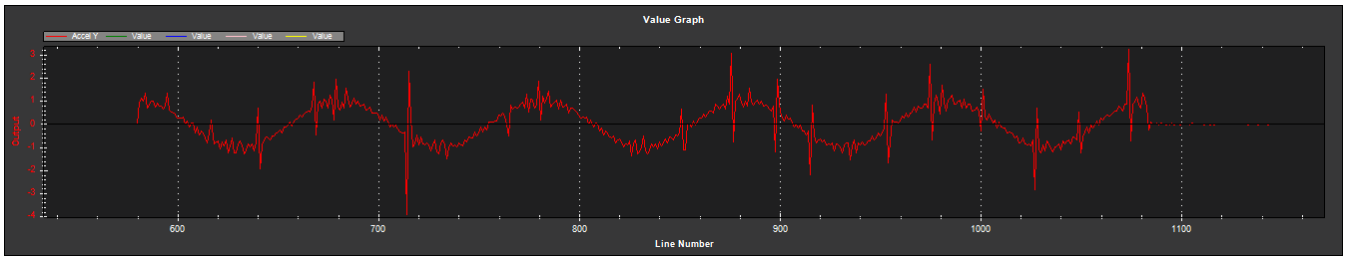


Figura 36 - Log

La possibilità che queste imprecisioni dei sensori influenzino in maniera pesante il software che stiamo sviluppando è alta. E' necessario quindi implementare un filtro che permetta di attenuare gli errori ed avere la registrazione ottimale che desideriamo.

Filtro complementare

Un filtro complementare è un filtro lineare che combina gli effetti di un passo-basso e di un passa-alto che può essere schematizzato come in figura dove:

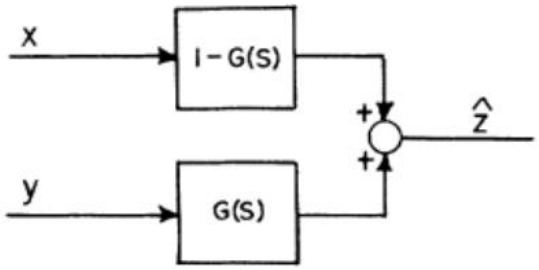


Figura 37 – Schema di un filtro complementare

- x è una misura perturbata del segnale z che è sensibile ad errori a bassa frequenza;
- y è una misura perturbata del segnale z che è sensibile ad errori a alta frequenza;
- $\hat{\mathbf{z}}$ è la stima del segnale z prodotta dal filtro;
- G(s) è la funzione di trasferimento del filtro passa basso.

Nel caso più semplice, al filtro passa alto è associata la funzione di trasferimento H(s)=1-G(s).

La tipologia di funzione che implementa tale filtro è costruita come

$$\hat{z}_t = a * (\hat{z}_{t-1} + y) + (1 - a) * x$$

dove G(s)= $a = \frac{\tau}{\tau + dt}$, dt rappresenta il tempo di loop del sistema e τ è la costante di tempo di sensibilità del segnale (segnali discretamente più lunghi o più corti sono filtrati).

Un esempio concreto di utilizzo del filtro è quello a cui abbiamo già accennato nella descrizione del software Ardupilot per la stima dell'assetto.

Pur essendo il filtro di Kalman quello che minimizza l'errore e che fornisce stima pulita ed accurata dei dati dei sensori prendendo in considerazione proprietà fisiche del sistema, richiede di calcolare coefficienti delle matrici basati sull'errore del processo, sull'errore di misura etc che non sono banali. Risulta quindi essere un vero e proprio algoritmo, in quanto, oltre a filtrare i dati grezzi che riceve in ingresso, li elabora per fornire in uscita una stima della misura. Una caratteristica molto importante di questo filtro è il fatto di essere ricorsivo: ovvero è capace di processare le nuove misure che arrivano in input in tempo reale.

Per evitare di aumentare il tempo di elaborazione del processore è necessario però diminuire il carico computazionale che grava sullo stesso. Un ottima maniera per farlo è proprio quello di utilizzare un filtro complementare.

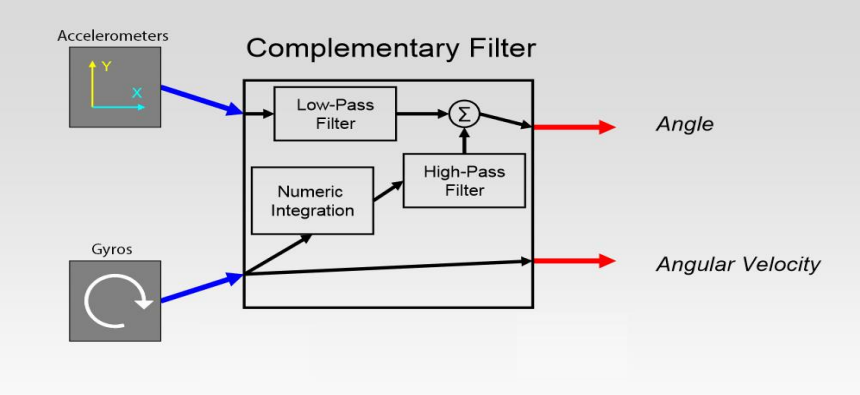


Figura 38 – Filtro complementare

Proprio come descritto in precedenza:

- a causa del processo di integrazione, la stima derivante dal giroscopio è influenzata da errori che si propagano con il tempo e quindi risulta molto instabile per lunghi utilizzi, mentre è affidabile per periodi brevi.
- la stima derivante dall'uso di accelerometri è influenzata dalle vibrazioni ma è utile per poter correggere la posizione statica dopo un lungo periodo di tempo.

La stima dell'assetto avverrà quindi applicando ai dati ottenuti dagli accelerometri un filtro passa-basso ed a quelli calcolati integrando i valori ricavati dai gyros un passa-alto.
In un caso bidimensionale abbiamo ad esempio

$$angle_t = a * (angle_{t-1} + \text{gyro} * \text{dt})) + (1 - a) * acc\_x$$

Nel caso di nostro interesse non avendo a che fare con giroscopi e con errori per cui sia necessario applicare un filtro passa-alto, strutturiamo un passa-basso che ci aiuti a migliorare la qualità dei dati degli accelerometri.

$$acc\_filtr\_y = acc\_filtr\_y * (1 - 1/k) + acc\_\log\_y * (1/k);$$

In Figura 39 e Figura 40 viene mostrato l'effetto del filtro per k=5 e k=20: in rosso i valore RAW dal sensore ed in blu quella filtrata.

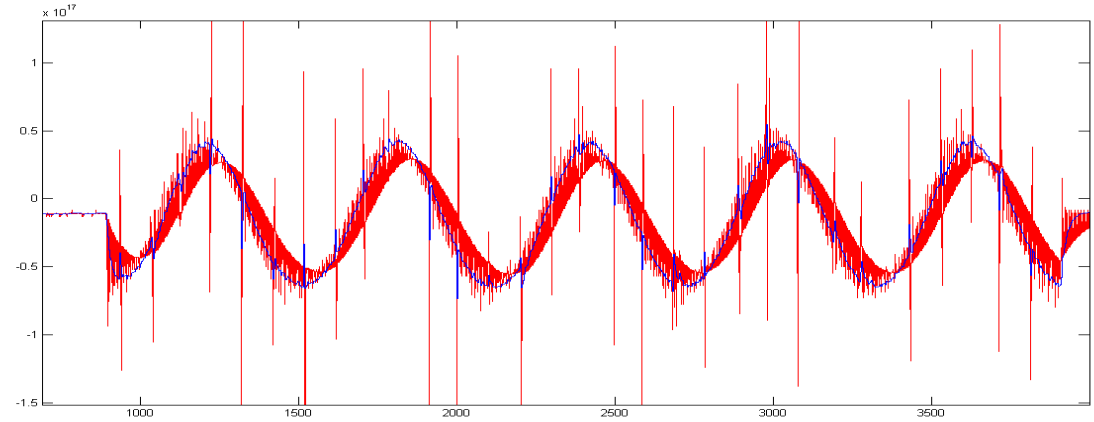


Figura 39 – Filtro 1

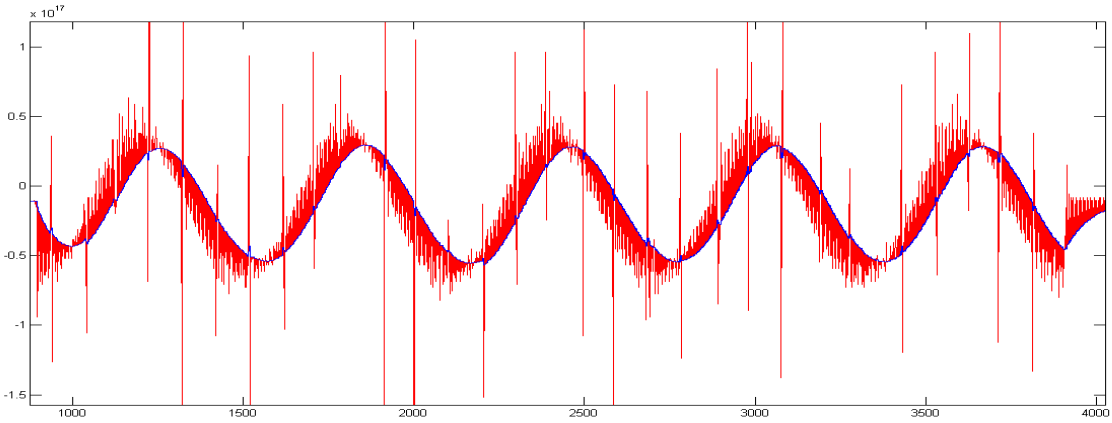


Figura 40 – Filtro 2

Volendo essere ancora più "*smooth*" possiamo aumentare il k di filtraggio andando ad attenuare una ulteriore fascia di frequenze.

Inizializzazione

Abbiamo già ampliamente trattato dei limiti dei servomeccanismi stepper nel capitolo dedicato. Ci saranno quindi accelerazioni che il motore è in grado o non è

in grado di replicare. Come presentato nell'introduzione il nostro obiettivo è quello di generalizzare il software ad un qualunque servo stepper, quindi dalle diverse potenzialità, senza però conoscerne le caratteristiche a priori. La cosa più sensata da fare è quella di implementare una sorta di inizializzazione che vada a valutare il motore e la registrazione del profilo di accelerazione che si ha l'obiettivo di replicare.

Come per il caso della simulazione in cui i parametri dei blocchi limitanti erano ricavati da una porzione della stessa, così dobbiamo fare nella pratica.
Basiamo parte della nostra analisi sul test che valuta i tempi di raggiungimento della velocità nominale. Grazie ad esso siamo in grado di calcolare la massima variazione di posizione attuabile nel tempo di processo, attraverso l'integrazione della curva di accelerazione, così come allo stesso modo la massima accelerazione ottenibile. E' poi necessario analizzare l'intero log per la verifica dei dati presentati e per la valutazione dell'escursione angolare.

Raggiungimento della posizione comandata e limiti di comando

Abbiamo visto come sia condizione assolutamente necessaria al buon funzionamento del servo che la posizione in output sia raggiunta prima del successivo comando. Bisogna allora individuare la condizione critica che definisce un limite di accelerazione per il servo.
Una situazione problematica può essere individuata proprio nella partenza da fermo del servo. In tali intervalli la variazione di posizione è minima e dalle simulazioni questi risultano essere i punti in cui il servo fatica maggiormente a stare dietro alla curva delle posizioni comandate. La variazione di velocità richiesta è infatti massima: dovremo valutare la prontezza del servo.

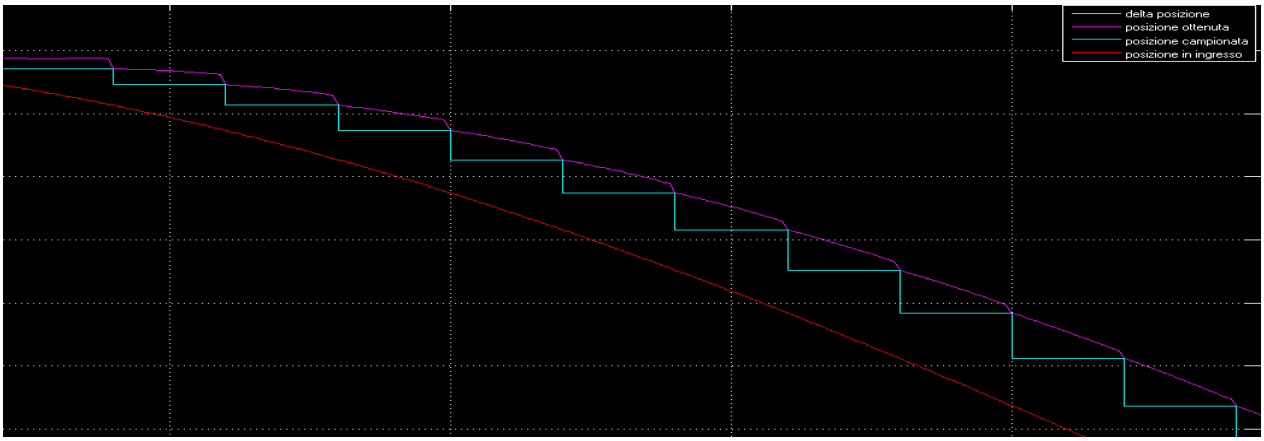


Figura 41 – Particolare della fase critica

Per farlo decidiamo di applicare qualche approssimazione e utilizzare il test menzionato. Attraverso l'elaborazione dei dati del log della prova otteniamo il valore dell'intervallo di tempo necessario a raggiungere la velocità massima (come mostrato nei paragrafi precedenti).

Lavorando in dominio discreto abbiamo che

$$Acc_{max} * T_{loop} = dV_{max}$$

dove dV è la variazione di velocità nel tempo di loop del sistema.

Essendo in condizione di accelerazione massima la velocità uguale a zero, possiamo giustificatamente usare i dati acquisiti. Approssimando l'aumento di velocità come circa costante andiamo a calcolare la sua variazione massima ottenibile in un intervallo di comando con

$$\frac{V_{max}}{T_{trans}} T_{loop} = dV_{max\,ott}$$

Dobbiamo avere che $dV_{maxott} > dV_{max}$.

Anche per calcolare la velocità massima ottenuta dalla prova è necessaria una integrazione dei dati in accelerazione. Supponendo la curva una sinusoide del tipo

$$\mathrm{A}_{\text{test max}} \sin\left(\frac{\pi}{T_{trans}} t\right) = \mathrm{A}_{\text{test max}} \sin(wt)$$

otteniamo dalla sua integrazione tra 0 e π

$$V_{servo\ max} = \int_0^{\pi} \mathrm{A}_{\text{test max}} \sin(wt)\ dt = \frac{2\mathrm{A}_{\text{test max}}}{w}$$

Per cui dovrà essere vero

$$\frac{2\mathrm{A}_{\text{test max}}}{w} \frac{T_{loop}}{T_{trans}} > Acc_{max} * T_{loop}$$

Ovvero

$$\frac{2\mathrm{A}_{\text{test max}}}{\pi} > Acc_{max}$$

Per quanto riguarda il limite sulla velocità nominale del servo il ragionamento è molto più semplice. Attraverso l'integrazione discreta dell'accelerazione del log da replicare, valutiamo le velocità massima che sarebbe necessario raggiungere.
Dovrà chiaramente valere

$$V_{\log max} < V_{servo\ max}$$

Alla stessa maniera integrando due volte abbiamo una curva in PWM, funzione del tempo. Dovrà allora valere

$$PWM_{min} < PWM(t) < PWM_{\max} \quad \forall t$$

Per riuscire a fare questa valutazione è necessario introdurre una ulteriore porzione di inizializzazione atta a calcolare il guadagno K visto nella schematizzazione in Simulink. Questa costante ci è utile a passare da posizione ad angolo e quindi a PWM. Senza tale parametro non ci sarebbe possibile avere una stima sul valore del comando, senza conoscere a priori il raggio del braccio, il sistema, etc.
Anche in questo caso ci è necessaria una sorta di retroazione che compari risultati ottenuti con quelli comandati e modifichi il sistema in funzione dei risultati.

Sfruttiamo l'occasione proprio per introdurre un feedback sul comportamento del sistema prevedendo il salvataggio di un log sinusoidale di inizializzazione la cui replica valuti il k prima della riproduzione del log desiderato.
Riserviamo a tale registrazione una porzione della memoria flash accessibile solo nella prima fase di funzionamento della piattaforma e calcoliamo il K correggendo il fattore moltiplicativo del sistema sulla base del rapporto tra accelerazione ottenuta e accelerazione comandata.

Riproduzione

Sulla base del software comparativo già scritto andiamo ad implementare la porzioni di codice atta ad elaborare i dati registrati per poter comandare il servo e attuare la replica del log.
Riprendendo lo schema Simulink vediamo come sia necessario, dall'accelerazione, moltiplicare per il Gain ed integrare due volte il dato.
Nel nostro caso sarà necessaria una integrazione discreta effettuata nel tempo di loop: otteniamo la variazione di PWM di comando del motore da

$$dPWM = K * Acc_{\mathrm{reg}}(t) * T_{loop}^2$$

e sommiamo al valore precedente del dato.

Codice

Funzioni richiamate nel main loop

```
1.  static void piattaforma_loop()
2.
3.  {
4.
5.
6.   digitalWrite(A_LED_PIN, HIGH);
7.   digitalWrite(C_LED_PIN, LOW);
8.   if (!controllo)
9.    { controllo = inizializzazione( k_uff, Acc_Poss_Max); }
```

```
    else
  { Serial.printf_P(PSTR("\n flag_riproduzione \n"));
  riproduzione(G_Dt, k_uff, Acc_Poss_Max);}


}


 static void acquisizione_accelerazione()

{


  if (erase_log_flag!=0)
      {

        # if CONF_MODE == CONF_MODE_ENABLED
       { b_log =3801;
        t_log =4096;  }
        #else
       { b_log = 1;
        t_log = 3801; }
        #endif
    Serial.printf_P(PSTR("\nErasing log...\n"));
    for(int j = b_log; j <  t_log; j++)
        DataFlash.PageErase(j);
    DataFlash.StartWrite(1);
    DataFlash.WriteByte(HEAD_BYTE1);
    DataFlash.WriteByte(HEAD_BYTE2);
    DataFlash.WriteByte(LOG_INDEX_MSG);
    DataFlash.WriteByte(0);
    DataFlash.WriteByte(END_BYTE);
    DataFlash.FinishWrite();
    Serial.printf_P(PSTR("\nLog erased.\n"));

        erase_log_flag = 0;
      }


   if (!flag_log)
  {
 # if CONF_MODE == CONF_MODE_ENABLED
     DataFlash.StartWrite(3801);
    #endif
  flag_log=1;
}
        if (g.log_bitmask & MASK_LOG_ATTITUDE_FAST)
            Log_Write_Attitude((int)dcm.roll_sensor, (int)dcm.pit
ch_sensor, (uint16_t)dcm.yaw_sensor);

        if (g.log_bitmask & MASK_LOG_RAW)
            Log_Write_Raw();


    digitalWrite(C_LED_PIN, HIGH);
```

```
64.         digitalWrite(A_LED_PIN, LOW);
65.
66. }
```

Nel file di configurazione dei parametri

```
8.  #define CONF_MODE CONF_MODE_DISABLED
```

Nel file Inizializzazione.pde

```
1.  float acc_max=0;
2.  float acc_max_reg=0;
3.  float acc0_iniz;
4.  float acc_iniz;
5.  float T_ing;
6.  float T_amax;
7.  float Ti;
8.  float acc_iniz_log[6] ;
9.  float k=50;
10. float k1 = 50
11. ;
12. float k2;
13. float Var_T=0.02;
14. //float acc_reg ;
15. float  V_Max;
16. float Arco_trans;
17.
18.
19. int valk1;
20. int last_page_iniz;
21. int case_iniz;
22.
23. Vector3f tempaccel;
24.
25. boolean insk=0;
26. boolean control ;
27. boolean iniz_ardu_flag ;
28. boolean boundary=0;
29.
30. static boolean inizializzazione (float *k_replica, float* Acc_Pos
    s_Max)
31. {
32.
33.        if (!iniz_ardu_flag)
34.
35.       {   case_iniz = 0;
```

```
      APM_RC. OutputCh (PAN_SERVO ,1000) ;
      T_ing = millis();
          acc_log_0=0;
          iniz_ardu_flag = 1;
      }
  case_iniz = int ((millis() - T_ing)/3000);
if (case_iniz > 1)
    case_iniz = 2;

switch (case_iniz)
{
    case 0:
            break ;
    case 1:
         APM_RC. OutputCh (PAN_SERVO ,2000) ;
                tempaccel = imu.get_accel();

        acc_iniz = tempaccel.y;

        acc0_iniz = acc0_iniz * (1.0f -
(1.0f/20)) + acc_iniz * (1.0f/20);

        if (acc0_iniz > acc_max)
        {   acc_max = acc0_iniz;
            T_amax = Ti;
        }

              break ;
    case 2:
      if (!boundary)

          {
              V_Max = 2*acc_max*(T_amax - Ti)/PI ;
              Acc_Poss_Max[0]= 2*acc_max/PI;


           Serial.printf_P(PSTR("\n acc max iniz  "));
           Serial.print(acc_max, DEC);
            Serial.printf_P(PSTR("\n acc max possibile  "));
           Serial.print(Acc_Poss_Max[0], DEC);
           acc_max=0;
           last_page_iniz = find_last_loginiz_page();
           Serial.printf_P(PSTR("\n boundary "));
           Serial.print(last_page_iniz, DEC);
           boundary=1;}

          control = Log_Read2 (3801, 3850, acc_iniz_log);
          acc_iniz_log[4] = acc_log_0 * (1.0f -
(1.0f/20)) + acc_iniz_log[4] * (1.0f/20);
          Arch = -(50)*(( acc_iniz_log[4]/t7 )*Var_T)*Var_T;
          Pan = ToDeg( Arch );
          float pwm_pan = Pan * PAN_RATIO ;
          pwm_pan_0 += pwm_pan;

          if (pwm_pan_0 > pwm_pan_max)
```

```
                {pwm_pan_0 = pwm_pan_max;}
                   else  if (pwm_pan_0 < pwm_pan_min)
                          {pwm_pan_0 = pwm_pan_min;}

          APM_RC. OutputCh (PAN_SERVO ,pwm_pan_0 ) ;

         acc_log_0= acc_iniz_log[4];
         tempaccel = imu.get_accel();
         tempaccel = tempaccel*t7;
         acc_reg = acc_reg * (1.0f -
   (1.0f/20)) + tempaccel.y * (1.0f/20);

          if (!insk)
          { if (pwm_pan_0!=2000)
              insk=1;
          }
          else{
          k = acc_log_0 / acc_reg ;
          k1 = k1*(1.0f -
   (1.0f/20)) + (1.0f/20)*(acc_log_0/ acc_reg);
          }
          if (acc_iniz_log[4] > acc_max)
         acc_max = acc_iniz_log[4];
      if (acc_reg > acc_max_reg)
         acc_max_reg = acc_reg;
      if ( control)
                { k2=  acc_max / acc_max_reg;
                  Serial.printf_P(PSTR("\n accmaxlog "));
                  Serial.print(acc_max, DEC);
                  Serial.printf_P(PSTR("\n accmaxrip "));
                  Serial.print(acc_max_reg, DEC);
                  Serial.printf_P(PSTR("\n k2 "));
                  Serial.print(k2, DEC);
                  k_replica[0]= k2;
                }
 Serial.printf_P(PSTR("\n a log "));
    Serial.print(acc_log_0, DEC);
    Serial.printf_P(PSTR("\n a  real "));
    Serial.print(acc_reg, DEC);
    Serial.printf_P(PSTR("\n pwm1 "));
    Serial.print( pwm_pan, DEC);
    Serial.printf_P(PSTR("\n pwm2 "));
     Serial.print( pwm_pan_0, DEC);
    Serial.printf_P(PSTR("\n k "));
    Serial.print(k, DEC);
    Serial.printf_P(PSTR("\n k1 "));
    Serial.print(k1, DEC);


    break ;


}
return control;
}
```

```
142.
143.
144. static int find_last_loginiz_page()
145. {
146.     int top_page = 4096;
147.     int bottom_page= 3801;
148.         int look_page;
149.     long check;
150.
151.     while((top_page - bottom_page) > 1) {
152.         look_page = (top_page + bottom_page) / 2;
153.         DataFlash.StartRead(look_page);
154.         check = DataFlash.ReadLong();
155.         if(check == (long)0xFFFFFFFF)
156.             top_page = look_page;
157.         else
158.             bottom_page = look_page;
159.     }
160.     return top_page;
161. }
```

Nel file Riproduzione.pde

```
1.  #define HEAD_BYTE1 0xA3
2.  #define HEAD_BYTE2 0x95    // Decimal 149
3.  #define END_BYTE 0xBA      // Decimal 186
4.
5.
6.  float Var_Time;
7.  float  acc_log[6] ;
8.  float raggio = 0.2;
9.  float Pan_0 = 0;
10. float  acc_log_0 = 0;
11. float Arch = 0;
12. float Arch0 = 0;
13. float V_Servo_Max ;
14. float acc_max1=0;
15. byte replica_log = 1;
16. float Acc_min;
17. float acc_reg;
18. float dV_Servo;
19. float V_Servo=0;
20.
21.
22. int pwm_pan_max =2000;
```

```
23. int pwm_pan_min=1000;
24. int pwm_pan_0=2000;
25. int dump_log_start2;
26. int inizio_log;
27. int dump_log_end2;
28. int fine_log;
29.
30. boolean ultimo = 0;
31. boolean acc_pox = 0;
32. boolean firstlog = 0;
33. boolean poslog = 0;
34. boolean finito = 0;
35. boolean possible = 0;
36.
37.
38.
39. void riproduzione (float Var_Time, float* k_f, float* Acc_Poss_Ma
    x)
40.   {
41.           if (!limitilog)
42.         {
43.            Arch = ToRad(1/PAN_RATIO);
44.            Acc_min = Arch*t7/(Var_Time * Var_Time * 50*k_f[0] );

45.            V_Servo_Max= 2*Acc_Poss_Max[0]*(T_amax - Ti)/PI ;
46.             Serial.printf_P(PSTR("\n acc min \n"));
47.             Serial.print( Acc_min, DEC);
48.             Serial.print( k_f[0], DEC);
49.             APM_RC. OutputCh (PAN_SERVO , 2000 ) ;
50.             pwm_pan_0 = 2000;
51.            get_log_boundaries(replica_log, dump_log_start2, dump_
    log_end2);
52.            inizio_log = dump_log_start2;
53.            fine_log = dump_log_end2;
54.            page_ext = inizio_log;
55.
56.            poslog = 0;
57.            limitilog = 1 ;
58.         }
59.    if (!possible)
60.     {
61.        Serial.printf_P(PSTR("\n dentro possibile \n"));
62.            ultimo = Log_Read2(inizio_log, fine_log, acc_log);
63.            acc_log[4] = acc_log_0 * (1.0f -
    (1.0f/20)) + acc_log[4] * (1.0f/20);
64.            acc_log_0 = acc_log[4];
65.            dV_servo = ( acc_log[4]/t7 )*Var_Time;
66.            V_Servo+= dV_Servo;
67.             Arch = - k_f[0] *50*((( acc_log[4]/t7 )*Var_Time)*Var
    _Time);
68.
69.             Pan =  Arch*180/PI;
70.             float pwm_pan = Pan * PAN_RATIO ;
71.             pwm_pan_0 += pwm_pan;
72.
```

```
73.
74.             if (acc_log[4] > acc_max1 )
75.              { acc_max1 = acc_log[4] ;
76.                if (acc_max1 > Acc_Poss_Max[0] )
77.         { acc_pox=1;
78.                     Serial.printf_P(PSTR("\n Registrazione non rip
    roducibile col servo a disposizione \n"));
79.                     Serial.print( acc_max1, DEC);
80.                     Serial.printf_P(PSTR(" \n"));
81.                     Serial.print( Acc_Poss_Max[0], DEC);
82.                     digitalWrite(C_LED_PIN, HIGH);
83.                digitalWrite(A_LED_PIN, HIGH);
84.                digitalWrite(B_LED_PIN, HIGH);
85.                     possible = 1;
86.               }
87.              }
88.
89.               if (V_Servo_Max<V_Servo)
90.               { possible = 1;
91.                 acc_pox=1;
92.                     Serial.printf_P(PSTR("\n Registrazione non rip
    roducibile col servo a disposizione \n"));
93.
94.               }
95.               if (pwm_pan_0 > 2300|| pwm_pan_0<700)
96.               { possible = 1;
97.                 acc_pox=1;
98.                     Serial.printf_P(PSTR("\n Registrazione non rip
    roducibile col servo a disposizione \n"));
99.
100.               }
101.               if (ultimo)
102.                { if (acc_max1 < Acc_min )
103.                 { acc_pox=1;
104.                    Serial.printf_P(PSTR("\n Registrazione non ri
    producibile col servo a disposizione \n"));
105.                     digitalWrite(C_LED_PIN, HIGH);
106.                digitalWrite(A_LED_PIN, HIGH);
107.                digitalWrite(B_LED_PIN, HIGH);
108.               }
109.               possible = 1;
110.               poslog = 0;
111.               page_ext = inizio_log;
112.              }
113.
114.     }
115.    else {
116.
117.           if ((page_ext <= fine_log) && (!acc_pox) )
118.         {
119.             Log_Read2(inizio_log, fine_log, acc_log);
120.             acc_log[4] = acc_log_0 * (1.0f -
    (1.0f/20)) + acc_log[4] * (1.0f/20);
121.             Arch = - k_f[0] *50*((( acc_log[4]/t7 )*Var_Time)*Va
    r_Time);
```

```
122.             Pan =  Arch*180/PI;
123.             float pwm_pan = Pan * PAN_RATIO ;
124.             pwm_pan_0 += pwm_pan;
125.
126.          if (pwm_pan_0 > pwm_pan_max)
127.                {pwm_pan_0 = pwm_pan_max;}
128.                   else  if (pwm_pan_0 < pwm_pan_min)
129.                          {pwm_pan_0 = pwm_pan_min;}
130.
131.          APM_RC. OutputCh (PAN_SERVO ,pwm_pan_0 ) ;
132.
133.    acc_log_0= acc_log[4];
134.    Vector3f tempaccel = imu.get_accel();
135.    tempaccel = tempaccel*t7;
136.    acc_reg = acc_reg * (1.0f -
    (1.0f/20)) + tempaccel.y * (1.0f/20);
137.
138.     Serial.printf_P(PSTR("\n a log "));
139.     Serial.print(acc_log_0, DEC);
140.     Serial.printf_P(PSTR("\n a  real "));
141.     Serial.print(acc_reg, DEC);
142.     Serial.printf_P(PSTR("\n pwm "));
143.     Serial.print( pwm_pan, DEC);
144.
145.  }
146. else { flag_piattaforma = 0 ;}
147.
148. }
149.
150. }
151.
152.
153. static boolean Log_Read2(int start_page, int end_page, float * r
    iferimento)
154. {
155.     byte data;
156.
157.
158.      if (!poslog)
159.        {
160.          log_step = 0;
161.      packet_count = 0;
162.          page_ext = start_page ;
163.          DataFlash.StartRead(start_page);
164.          poslog = 1 ;
165.        }
166.        packcount = packet_count;
167.           finito=1;
168.
169.       while (page_ext < end_page && page_ext != -
    1 && packcount == packet_count ){
170.          finito = 0 ;
171.                  data = DataFlash.ReadByte();
172.
```

```
173.                     switch(log_step)         // This is a state
    machine to read the packets
174.             {
175.             case 0:
176.                 if(data == HEAD_BYTE1)  // Head byte 1
177.                 {log_step++;}
178.                                 break;
179.             case 1:
180.                 if(data == HEAD_BYTE2)  // Head byte 2
181.                     log_step++;
182.                         else
183.                     log_step = 0;
184.                 break;
185.             case 2:
186.                                 if(data == LOG_ATTITUDE_MSG){
187.                     Log_Read_Attitude();
188.                     log_step++;
189.
190.                 }else if(data == LOG_MODE_MSG){
191.                     Log_Read_Mode();
192.                     log_step++;
193.
194.                 }else if(data == LOG_CONTROL_TUNING_MSG){
195.                     Log_Read_Control_Tuning();
196.                     log_step++;
197.
198.                 }else if(data == LOG_NAV_TUNING_MSG){
199.                     Log_Read_Nav_Tuning();
200.                     log_step++;
201.
202.                 }else if(data == LOG_PERFORMANCE_MSG){
203.                     Log_Read_Performance();
204.                     log_step++;
205.
206.                 }else if(data == LOG_RAW_MSG){
207.                                         Log_Read_Raw2(riferiment
    o);
208.                     log_step++;
209.
210.                 }else if(data == LOG_CMD_MSG){
211.                     Log_Read_Cmd();
212.                     log_step++;
213.
214.                 }else if(data == LOG_CURRENT_MSG){
215.                     Log_Read_Current();
216.                     log_step++;
217.
218.                 }else if(data == LOG_STARTUP_MSG){
219.                     Log_Read_Startup();
220.                     log_step++;
221.                 }else {
222.                     if(data == LOG_GPS_MSG){
223.                         Log_Read_GPS();
224.                         log_step++;
225.                     }else{
```

```
                        Serial.printf_P(PSTR("Error Reading Packet: %d\n"),packet_count);
                        log_step = 0;    // Restart, we have a problem...
                    }
                }
                break;
            case 3:
                if(data == END_BYTE){
                     packet_count++;
                }else{
                    Serial.printf_P(PSTR("Error Reading END_BYTE: %d\n"),data);
                }
                log_step = 0;           // Restart sequence: new packet...
                break;
            }
                    page_ext = DataFlash.GetPage();
                    return finito;
             }

}



void Log_Read_Raw2(float * riferim)
{
    float logvar;
        for (int y = 0; y < 6; y++)
        {
        riferim[y] = (float)DataFlash.ReadLong() ;

    }

}
```

RISULTATI

Pur essendo, come abbiamo visto, il dominio delle accelerazioni riproducibili dal servo non particolarmente esteso, siamo riusciti ad ottenere, con registrazioni compatibili, degli ottimi risultati.

Come abbiamo visto, sia per escursioni angolari molto basse che per escursioni angolari eccedenti i limiti il servo non segue perfettamente la ragistrazione. Nelle figure che seguono vengono mostrati due risulati in cui in rosso è graficata la curva d'accelerazione registrata ed in blu quella ottenuta. Nella seconda figura il risultato è stato filtrato per visualizzare meglio gli andamenti.

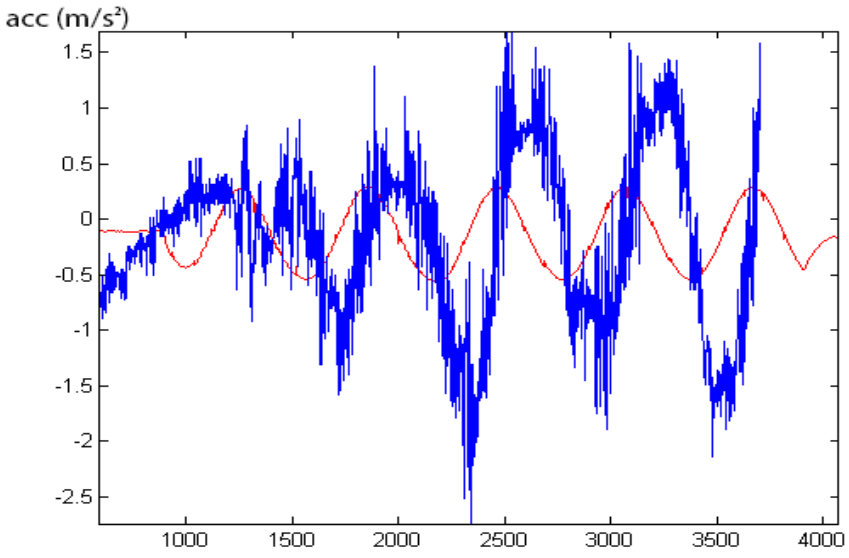


Figura 42 – Risultato non filtrato

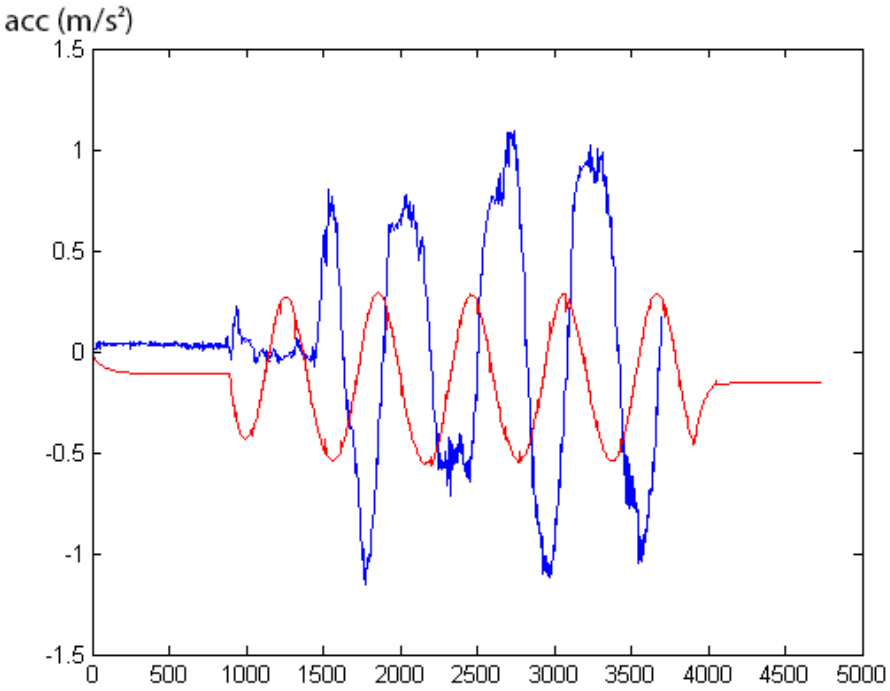


Figura 43 – Risultato filtrato

La Figura 42 e la Figura 43 fanno riferimento ai risultati ottenuti durante l'ultima fase dell'inizializzazione: osserviamo oltre al classico sfasamento una diversa ampiezza della vibrazione legata all'utilizzo di un K di tentativo ancora da correggere.

In Figura 44 è mostrata un test in cui l'escursione angolare del servo non è sufficiente a riprodurre l'accelerazione in ingresso. In figura Figura 45 invece è mostrato come una vibrazione a media non nulla faccia raggiungere al servo le posizioni limite inficiando il buon funzionamento del dispositivo.

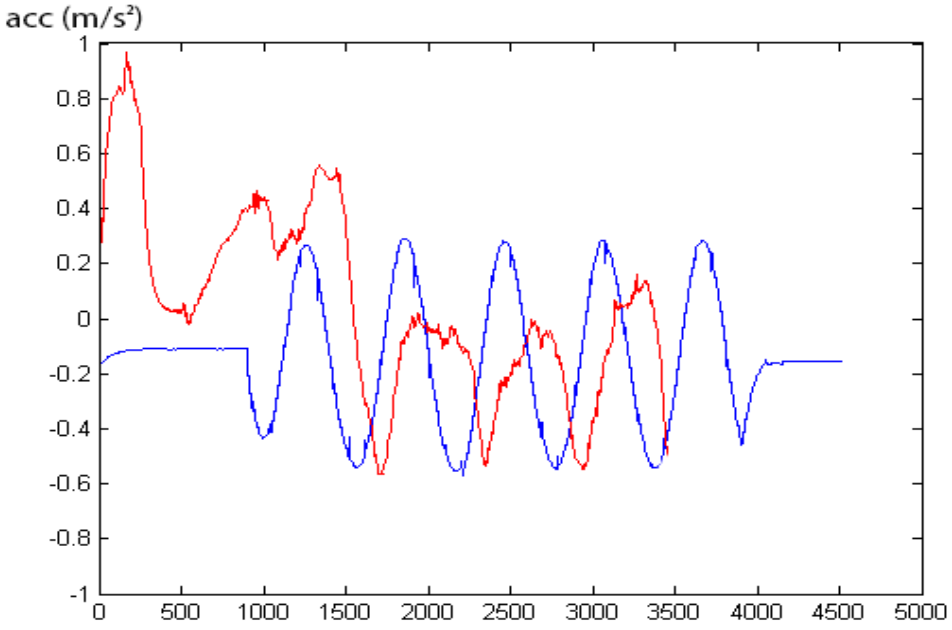


Figura 44 – Test 1

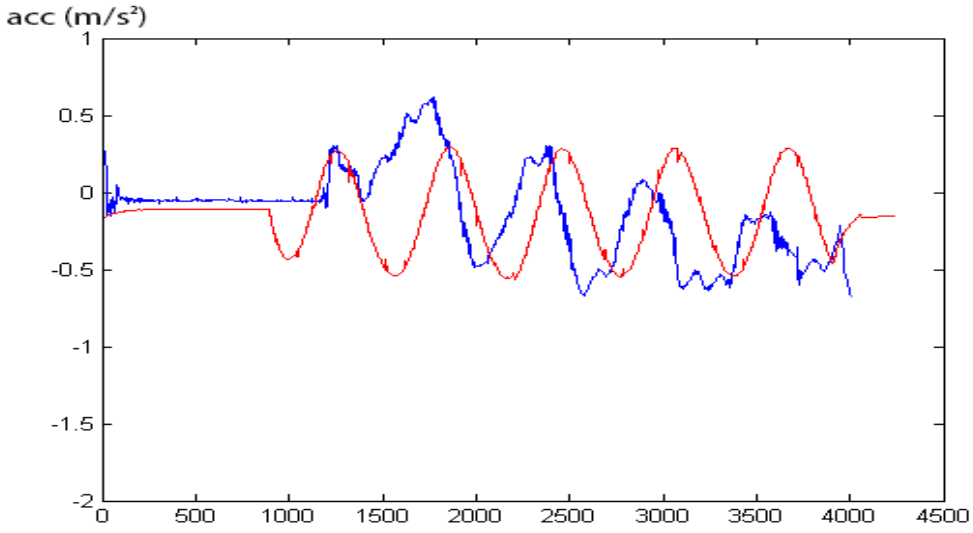


Figura 45 – Test 2

Osserviamo infine in Figura 46 un ottimo risultato di riproduzione del profilo in ingresso.

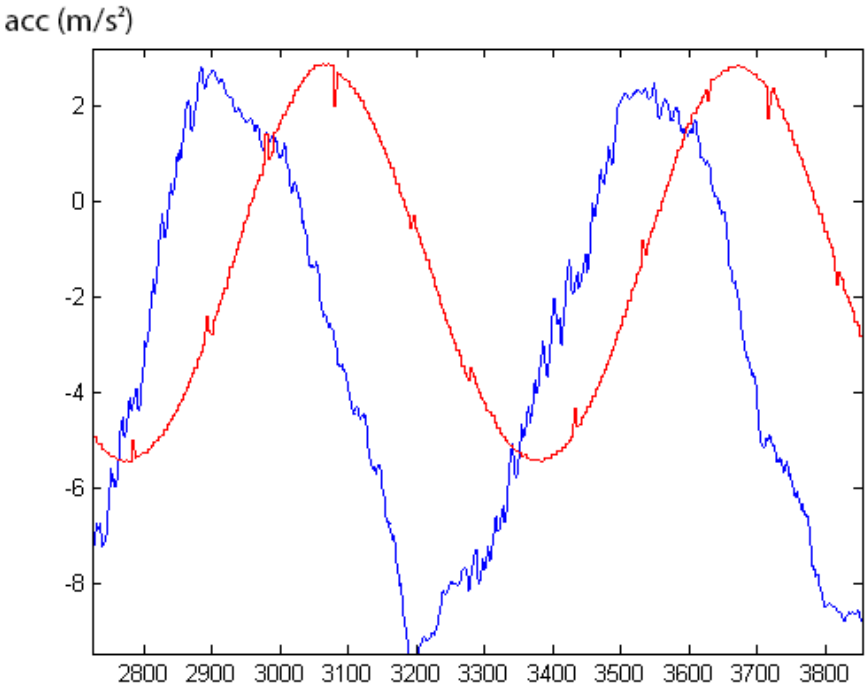


Figura 46 – Test 3

CONCLUSIONI

Siamo complessivamente soddisfatti dei risultati ottenuti attraverso il software scritto e implementato in Arduino. I profili di velocità conseguiti ricalcano molto bene quelli dati in input al sistema con fasi transitorie comunque accettabili.
Il fattori che osserviamo anche nei test essere più limitanti sono l'escursione angolare e la prontezza del servo. Queste problematiche sono però difficilmente risolubili se si rimane nel campo dei motori stepper.

Rimangono da definire i campi di applicazione del nostro progetto ed i possibili sviluppi futuri.

Come già accennato nell'introduzione, il lavoro svolto su un singolo asse può, in fase successiva, essere usato come base per uno sviluppo triassiale del dispositivo: questo riuscirebbe a renderlo più completo ed estenderne i campi d'applicazione.
Altra interessante aggiunta al sistema potrebbe essere quella di dedicare un set di uscite al controllo di motori stepper e un altro a quelli a velocità variabile per riuscire a slegare completamente il dispositivo dalla tipologia di motore.
Chiaramente per fare questo sarebbero necessarie alcune modifiche, ma molto limitate, del firmware e dell'hardware.

Per quanto riguarda i campi applicativi possiamo pensare a diverse soluzioni, ad esempio la registrazione di una accelerazione di interesse in un ambiente complesso, per studiarla e poterla replicare per effettuare test. Gli ambiti di questa applicazione sono moltissimi: infatti una curva in accelerazione non è altro che una vibrazione. Il nostro dispositivo potrà essere utile a effettuare test di fatica, di sollecitazione, di risposta a vibrazioni e molto altro.

Altra condizione plausibile è quella di registrazione e riproduzione di vibrazioni critiche per i dispositivi in studio. Un impiego molto semplice è la riproduzione di accelerazioni attraverso servomeccanismi su un UAV o su un qualsiasi dispositivo che può risentire di problematiche legate alle vibrazioni. Questo è un ambito molto interessante in cui poter lavorare in quanto la piattaforma è predisposta a utilizzare motori stepper già presenti nella maggior dei droni controllabili senza alcuna modifica dal sistema.

Infine pensiamo ad una applicazione per la quale è necessario un ulteriore sviluppo: quella di contrastare vibrazioni istantaneamente. Per fare ciò è necessario sia un servo molto pronto che un sistema operativo a tempo di funzionamento basso, in grado di comandare con frequenza abbastanza alta il motore.

BIBLIOGRAFIA

1)Matteo Zanzi, Antonio Ghetti, Niki Regina, Manuale Zanzi, Università di Bologna

2) Arduino project home page; http://arduino.cc/en/

3) ArduPilot Mega Documentation, http://code.google.com/p/ardupilot-mega/

4) DIYDrones forum; http://www.diydrones.com

5) *Di Bacco Diego*, Tracking di un target mediante telecamera su uav ad ala fissa, Università di Bologna

6) *Matteo Zanzi*, Accelerometri e giroscopi, Università di Bologna

7) *Shane Colton*, The Balance Filter

8) *Mattia Forconi*, Sviluppo e prototipazione di un sistema di stabilizzazione per la fotografia aerea