ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SECONDA FACOLTÁ DI INGEGNERIA CON SEDE A CESENA

CORSO DI LAUREA
in INGEGNERIA AEROSPAZIALE

Sede di Forlí

ELABORATO FINALE DI LAUREA
in Controlli Automatici

TRACKING DI UN TARGET MEDIANTE TELECAMERA SU UAV AD ALA FISSA

Candidato:
Diego Di Bacco

Relatore:
Prof. Matteo Zanzi
Correlatore:
Ing. Niki Regina

Anno Accademico 2012-2013
Sessione III^{a}

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici

Ringraziamenti

Desidero ringraziare il Professore Matteo Zanzi per i suoi preziosi insegnamenti e la sua grande disponibilità per lo svolgimento di questa tesi.
Allo stesso modo ringrazio l'Ingegnere Niki Regina con il quale ho strettamente collaborato e che mi ha supportato in questo periodo di lavoro introducendomi all'utilizzo dei programmi necessari per la realizzazione di questo elaborato.
Un sincero ringraziamento va anche agli Ingegneri Antonio Ghetti e Alessandro Mirri per i loro utili consigli.
Infine, un ringraziamento particolare alla mia famiglia che mi ha sempre sostenuto in questi anni di studio.

Introduzione

Presentazione e obiettivi

Il lavoro di tesi svolto in collaborazione con il CIRI ICT[1] (laboratorio LASIM[2]) s'inserisce in un ambito di ricerca che unisce l'ambiente aeronautico con quello dei controlli automatici. L'idea nasce dall'utilizzo sempre più diffuso di velivoli senza pilota UAV[3] in operazioni di monitoraggio ambientale, telerilevamento e sorveglianza. Il grande vantaggio nell'utilizzo di questi innovativi velivoli risiede nell'economicità con le quali queste operazioni possono essere effettuate rispetto ai velivoli tradizionali. A differenza di quest'ultimi, infatti, i droni possono essere utilizzati in situazioni caratterizzate da un elevato pericolo per la vita umana e nelle aree inaccessibili o impervie. Per questo motivo possono trovare impiego durante le fasi di monitoraggio di aree colpite da calamità naturali o da avvenimenti particolari (terremoti, incidenti stradali, esondazioni, ecc.). Un elemento imprescindibile per lo svolgimento di queste operazioni è l'installazione a bordo dell'UAV di una videocamera in grado di fornire a terra delle immagini, che costituiscono il fattore chiave per la riuscita della missione. L'idea che sta alla base del progetto è dunque quella di creare una legge di controllo di una telecamera montata a bordo di un UAV che segua, dinamicamente, un obiettivo a terra o in volo. Tramite un processo ausiliario di riconoscimento dell'immagine, l'operatore di terra selezionerà il target d'interesse e la telecamera automaticamente seguirà l'obiettivo mantenendolo centrato nell'inquadratura. Il sistema si baserà principalmente sulle posizioni relative tra target ed UAV e sulle trasformazioni per determinare la posizione del target nei pixels del display. La telecamera verrà installata sul velivolo ad ala fissa in dotazione al laboratorio LASIM del CIRI ICT. I vantaggi dell'utilizzo di questo tipo di UAV, rispetto ad un *rotory wing*, sono la maggior adattabilità alle avverse condizioni meteo ed una maggiore autonomia di volo; d'altro canto, ci saranno inevitabilmente degli svantaggi quali, ad esempio, un campo visivo comunque limitato. Nel progetto della legge di controllo,

[1] Centro di ricerca industriale dell'Università di Bologna per le tecnologie dell'informazione e della comunicazione

[2] Laboratorio Sistemi Infomobilità, Viale Seganti 103, 47121 Forlì, Italia

[3] Unmanned Aerial Vehicle

bisognerà anche tener conto di quei fattori che possono influire sulla perdità d'informazioni alla quale potrebbe andare in contro il sistema: presenza di ostacoli sulla linea di vista della telecamera o cambiamenti repentini delle condizioni ambientali in termini di luminosità oppure d'interferenze. É stato anche condotto uno studio della sua robustezza al fine di avere una descrizione più completa del problema.
In questo lavoro, si è partiti dalla derivazione analitica delle leggi matematiche che forniscono l'andamento generale degli angoli con i quali la telecamera deve ruotare. A partire da questo modello, ci si è basati su un sistema di equazioni differenziali di primo grado che, opportunamente risolto, fornisce tali angoli: in particolare, si è utilizzata la tecnica di *input-output linearization*. Il metodo ed i risultati ottenuti saranno poi testati implementandoli su vari software di utilizzo comune ed andranno ad integrare il sistema di autopilota con cui è equipaggiato il velivolo sperimentale.

Organizzazione delle attività

Le attività svolte per ottenere l'obiettivo espresso nella precedente sezione possono essere divise in quattro parti, secondo una sequenza logica ben precisa:

1. sviluppo del modello matematico e delle relative equazioni con il quale è possibile descrivere il problema;
2. sviluppo della legge di controllo per il tracking mediante telecamera;
3. studio della robustezza e dell'efficienza della legge di controllo;
4. simulazioni in ambiente software *MatLab-SimuLink* e verifica dei risultati;
5. implementazione software *ArduPilot Mega* e pianificazione delle prossime attività da effettuare per uno studio più completo del problema in oggetto.

Capitolo 1

Presentazione del velivolo e dei suoi componenti

1.1 Presentazione dell'UAV

Il velivolo utilizzato come piattaforma volante per le attività in oggetto è l'aeromodello in figura 1.1.

Figura 1.1: FPV 168

Si tratta di un velivolo ad ala fissa tipo FPV 168 della Hobby King avente le seguenti caratteristiche:

Caratteristiche	**Valore**
Lunghezza	1.19 m
Diametro supporto motore	0.058 m
Peso a vuoto	1.3 kg
Superfici mobili	elevator, rudder, 2 alettoni, 2 flap

Tabella 1.1: Caratteristiche generali FPV 168

In particolare, le caratteristiche geometriche e aerodinamiche del velivolo sono meglio precisate nella tabella seguente:

	Ala	**Stabilizzatore**	**Deriva**
Apertura	1.66 m	0.53 m	0.17 m
Corda media aerodinamica	0.23 m	0.16 m	0.06 m
Profilo aerodinamico	NACA 4412	NACA 0012	NACA 0012
Incidenza	0°	8.5°	-
Deflessione	±40°	±40°	±40°

Tabella 1.2: Caratteristiche aerodinamiche FPV 168

1.2 Descrizione dei sistemi di bordo

I componenti con i quali è equipaggiato il velivolo sono indicati nelle successive sottosezioni e nello schema hardware di figura 1.15.

1.2.1 Il sistema ArduPilot Mega

Il sistema *ArduPilot Mega* è l'autopilota open-source di cui è equipaggiato l'aereo. E' stato pensato dagli sviluppatori di *DIYDrones.com* per trasformare un qualunque velivolo ad ala fissa in un completo UAV in grado dunque di volare autonomamente e configurabile interattivamente tramite una Ground Control Station di terra (nel nostro caso si utilizza *APM Mission Planner*[4]). Per le specifiche in dettaglio di questo componente, si rimanda al paragrafo 3.1 a pagina 61 .

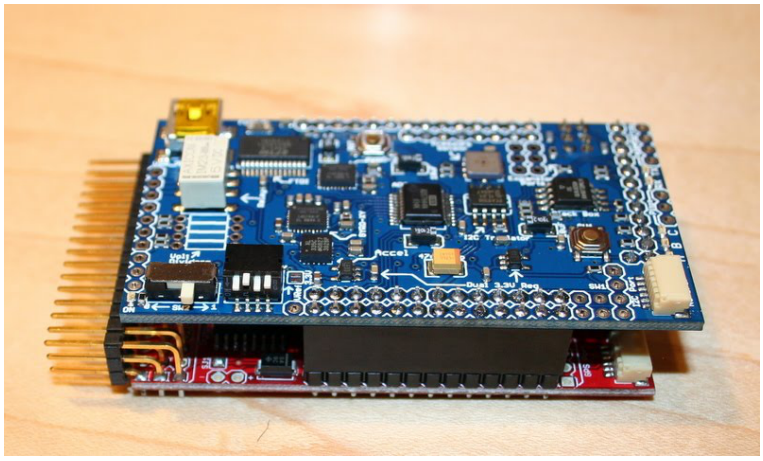


Figura 1.2: ArduPilot Mega

[4]La trasmissione bidirezionale dei dati nel sistema *ArduPilot Mega* avviene tramite il protocollo MAVLink (*Micro Air Vehicle Communication Protocol*): si veda [13].

1.2.2 Motore elettrico brushless

Per il modello UAV FPV 168 è stato utilizzato il motore *NTM Prop Drive 35-36A 1800KV* con peso pari a 120 grammi.

Figura 1.3: Motore brushless NTM Prop Drive 35-36A 1800KV

Corrente massima	65 A
Potenza massima	875 W a 15 V
Giri per Volt	1800 rpm/V
Shaft	4 mm

Tabella 1.3: Caratteristiche Motore

Per le batterie LiPo 4S (4 celle = 14.8 V) le caratteristiche sono:

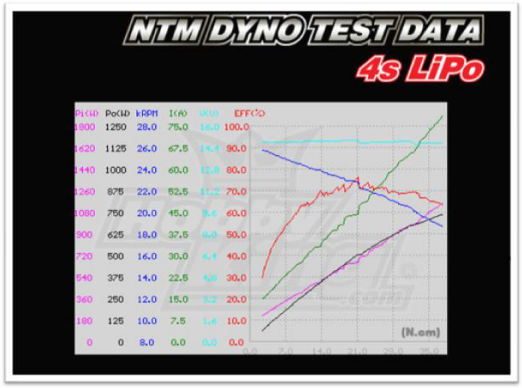


Figura 1.4: Caratteristiche motore NTM Prop Drive 35-36A 1800KV con LiPo 4S

Dalla figura 1.4 si vede che alla massima potenza in ingresso P_i=1100 W corrisponde una potenza resa P_o=750 W. La corrente consumata a potenza massima è di circa 75 A a 14.8 V.

1.2.3 Elica

La scelta dell'elica è in funzione delle caratteristiche del motore. Da prove effettuate in campo, si è preferito utilizzare eliche a due pale, di diametro che varia da 8" a 9" e di passo da 4" a 6". Il diametro massimo di 9" è dovuto alla distanza tra l'albero motore e la fusoliera dell'FPV 168.

Figura 1.5: Elica APC style propeller 9x6 per motore elettrico

Dato che l'albero del motore (*shaft*) è da 4 mm, per poter applicare l'elica (*propeller*) è stato necessario montare un altro supporto per cui lo shaft finale è da 6 mm. In definitiva, l'elica montata ha dunque un foro da 6 mm.

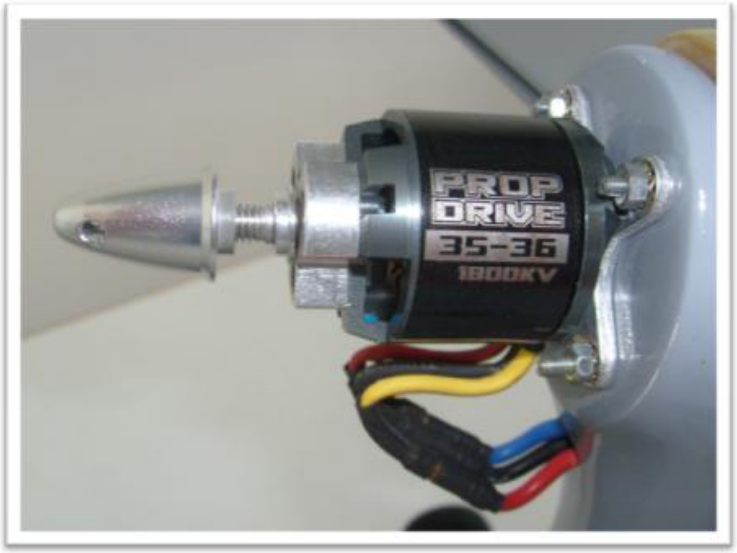


Figura 1.6: Shaft 6 mm

1.2.4 Aircraft ESC

L'ESC (*Electronic Speed Control*) utilizzato a bordo del drone è il modello *ESC 80A-ESC-4A-SBEC* della Hobby King. Esso è in grado di pilotare in maniera continua un motore brushless da 80 A. Inoltre, ha un alimentazione secondaria per ricevente, servi e logica di controllo SBEC da 5.5 V e 4 A massimi. Nel caso in cui la batteria LiPo scenda sotto i 3.0 V per cella, l'ESC attiva in automatico la protezione CUTOFF e la potenza del motore viene ridotta per lasciare potenza all'alimentazione SBEC della logica di controllo.

Figura 1.7: ESC 80A-ESC-4A-SBEC

Corrente sostenuta	80 A
Corrente istantanea massima	100 A
Output BEC	5.5V/4A
Dimensioni	67 x 32 x 23 mm
Peso	90 g

Tabella 1.4: Caratteristiche ESC

1.2.5 Batterie

Il pacco batterie utilizzato è il *4S LiPo* da 8000 mAh ed è alloggiato sulla base anteriore della fusoliera dell'aereo.

Figura 1.8: Batterie Zippy 4S LiPo 8000 mAh

Le caratteristiche del pacco batterie utilizzato sono di seguito riportate:

Scarica nominale	30 C costant - 40 C max
Capacità totale	1800 mAh
Dimensioni	166 x 69 x 35 mm
Peso Complessivo	845 g
Tensione carica	4.2V x 4 = 16.8 V
Tensione nominale	3.7V x 4 = 14.8 V
Tensione scarica	3.0V x 4 =12 V
Tensione morta	$\leq$ 2.7V x 4 =10.8 V

Tabella 1.5: Caratteristiche batterie 4S LiPo

Tenendo conto del solo consumo del motore a potenza massima, si può facilmente calcolare l'autonomia del pacco batterie come segue:

$$\frac{\text{Capacità totale batteria [mAh]}/1000 * 60}{\text{Corrente[A]}} = \text{Autonomia [minuti]} \tag{1.1}$$

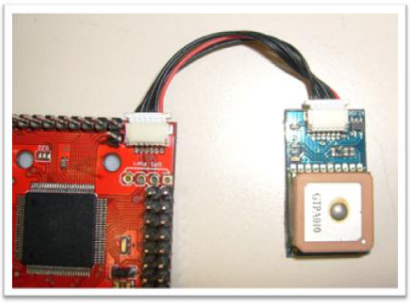
percui:

$$\frac{\frac{8000}{1000} * 60}{75} = 6.4 \tag{1.2}$$

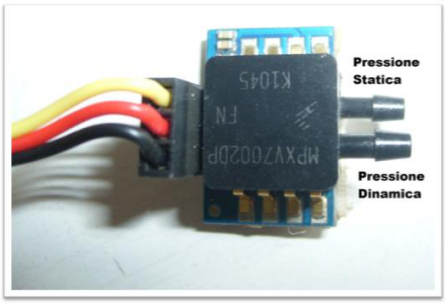
L'autonomia complessiva calcolata a massima potenza sarà pari a circa 6.4 minuti.

1.2.6 Sensori

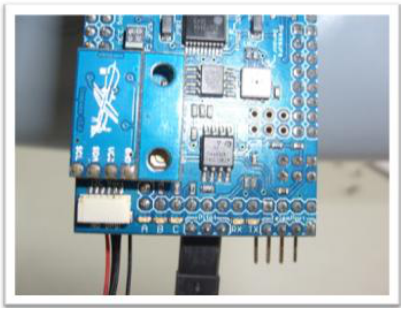
Oltre ad un semplice modulo GPS (figura 1.9(a)), il velivolo è stato equipaggiato dell'*AirSpeed Sensor MPXV7002DP* con presa statica e dinamica (in figura 1.9(b)) e del magnetometro a 3 assi *HMC5883L* (figura 1.9(c)) comunicante con l'autopilota attraverso il protocollo seriale I^2C [5].



(a) Modulo GPS



(b) AirSpeed Sensor



(c) Magnetometro HMC5883L

Figura 1.9: Sensori

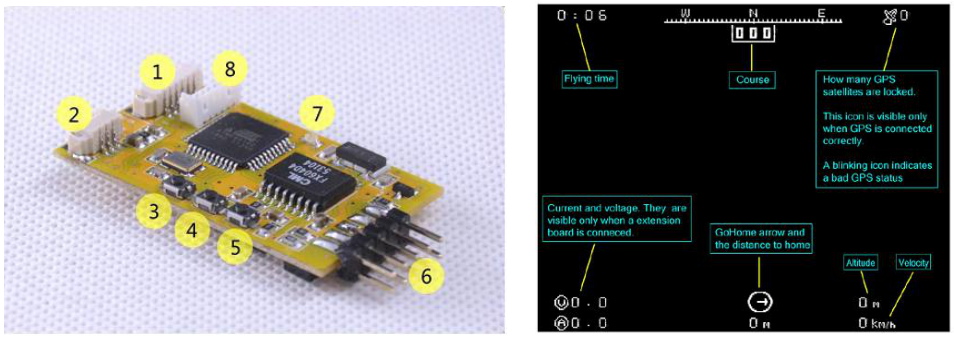
1.2.7 Scheda OSD

L'OSD (*On-Screen-Display*) montata sull'UAV è la *Telefly OSD* e fornisce due tipi di segnali: uno prettamente video, ottenuto dalla sovrapposizione di dati GPS, di batteria e video della telecamera, ed un segnale audio, che consente la trasmissione a terra dei singoli dati GPS e della batteria tramite il collegamento con il trasmettitore.

Peso	9 g
Dimensioni	45 x 25 mm
Voltaggio d'ingresso	7-20V (7-13V se connessa con Extension Board)
Corrente consumata	100 mA

Tabella 1.6: Caratteristiche Telefly OSD

[5] Inter Integrated Circuit. É un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati. Il classico bus I^2C è composto da almeno un *master* ed uno *slave*.



(a) Scheda OSD (b) Legenda video ricevuto a terra

Figura 1.10: Telefly OSD

1.3 Descrizione degli apparati radio

1.3.1 Radiocomando

L'apparato di radiocomando impiegato è il modello *Futaba T9CP* (mostrato in figura 1.12) con ricevente *Futaba R149DP 9 Channel DC 40 MHz* (in figura 1.11) con il quale è possibile comandare il velivolo manualmente.

Figura 1.11: Ricevente Futaba R149DP 9 Channel DC 40 MHz

All'interno del radiocomando è possibile salvare fino a 8 diversi possibili settaggi che, di conseguenza, possono rappresentare 8 diversi tipi di velivoli. La ricevente pilota dei servocomandi proporzionali con dei segnali PWM (*Pulse Width Modulation*), in cui la durata degli impulsi, misurata in μs, è proporzionale alla posizione degli *stick* sul radiocomando. Tuttavia, i segnali PWM in uscita dalla ricevente non vengono direttamente passati ai servocomandi, ma agli appositi *pin* d'ingresso del sistema *ArduPilot Mega*, il quale successivamente guiderà i servocomandi tramite i suoi *pin* d'uscita.

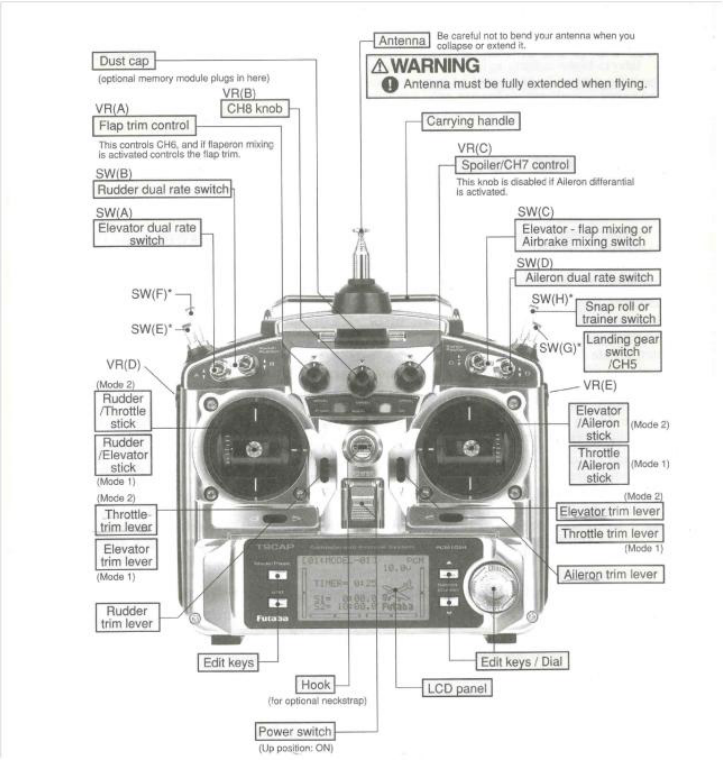


Figura 1.12: Radiocomando Futaba T9CP

1.3.2 Telemetria

Il sistema di telemetria serve ad inviare diversi dati prelevati dai sensori a bordo dell'aereo alla Ground Station a terra. Il canale utilizzato è un canale radio a 900MHz.

Figura 1.13: Xbee Pro Telemetry kit 900MHz

1.3.3 Video

Il trasmettitore montato a bordo del velivolo è il modello *FPV584X Plug-and-Play* della Hobby Wireless. Esso è indispensabile perchè il segnale video della telecamera, unito ai dati GPS sovraimpressi, sia correttamente inviato in diretta alla stazione di terra. L'antenna del trasmettitore è direzionata verso il basso per consentire una migliore trasmissione delle immagini. La frequenza utilizzata è 5.8 GHz.

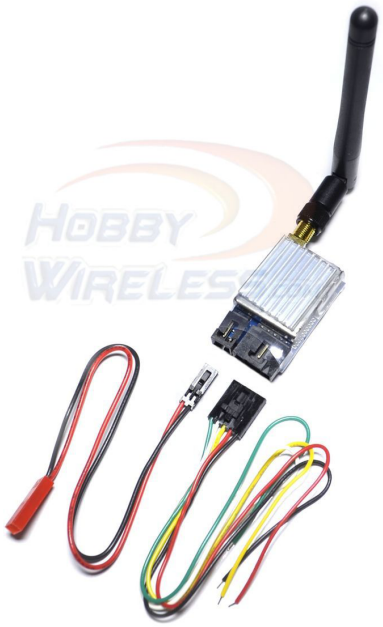


Figura 1.14: Trasmettitore FPV584X Plug-and-Play 5.8 GHz

1.4 Riassunto schema hardware

L'architettura di base del sistema appena descritto in ogni suo componente si basa principalmente sull'utilizzo dell'autopilota *ArduPilot Mega*. Esso è il vero e proprio 'cervello' del velivolo: ha in ingresso una serie di sensori i quali gli forniscono una serie di informazioni permettondogli di agire sul motore, sui servi di movimentazione delle superfici aerodinamiche e sui servi del sistema Pan-Tilt che verrà successivamente descritto. In questo modo, il software legge la posizione e l'orientamento corrente del velivolo e controlla un sistema *fly-by-wire* che lo guida. La movimentazione del Pan-Tilt della telecamera è stata integrata in questo sistema, implementando le sue equazioni fondamentali sul software d'utilizzo. Il segnale video della telecamera viene infine mandato a terra unitamente ai principali dati GPS e di batteria tramite un trasmettitore a 5.8 GHz e ciò fornisce un importante

ausilio all'operatore di terra sia per l'effettiva riuscita degli obiettivi della missione, sia per monitorare in tempo reale lo stato del velivolo.

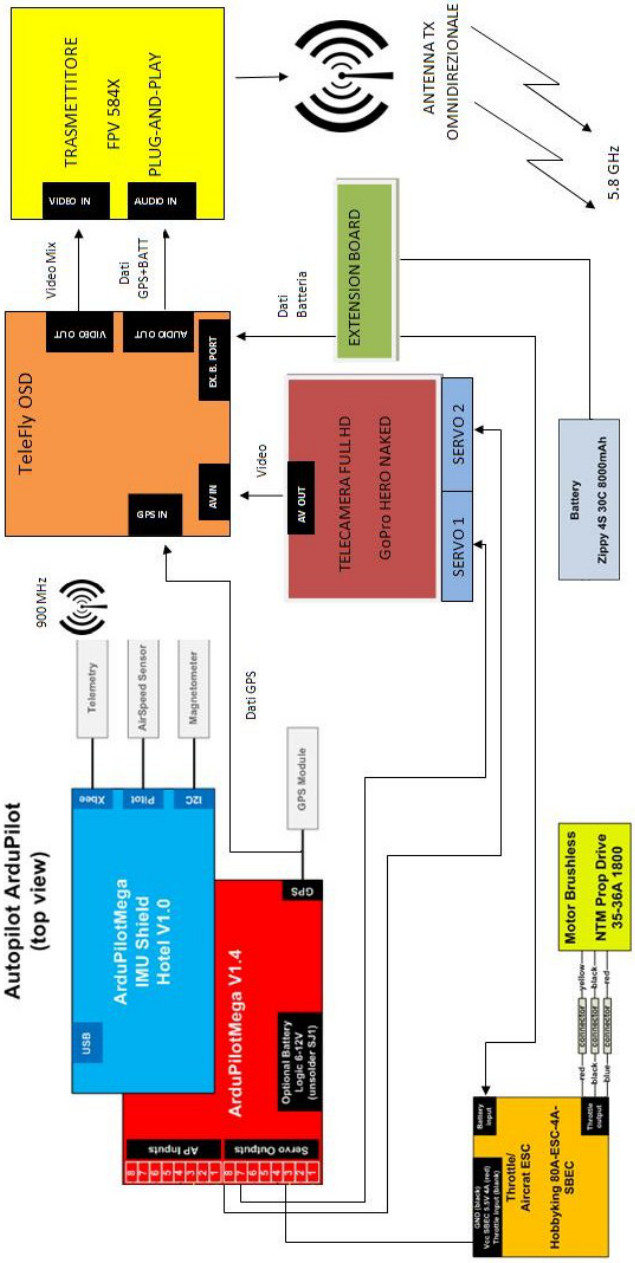


Figura 1.15: Schema Hardware sistema di bordo

1.5 Scelta e descrizione dei componenti d'interesse specifico

1.5.1 Il sistema Pan-Tilt

Elemento di fondamentale importanza per lo scopo della tesi è il sistema di movimentazione della telecamera con rotazioni di Pan e Tilt. Questo sistema è in grado di far ruotare la telecamera attorno l'asse verticale e l'asse trasversale e dunque direzionarla rispetto alla traiettoria dell'UAV. L'angolo di Pan è l'angolo di rotazione tra i piani X-Y dei sistemi di riferimento Camera e Body, solidale con il velivolo, mentre l'angolo di Tilt è il rispettivo angolo di rotazione tra i piani X-Z degli stessi sistemi di riferimento.

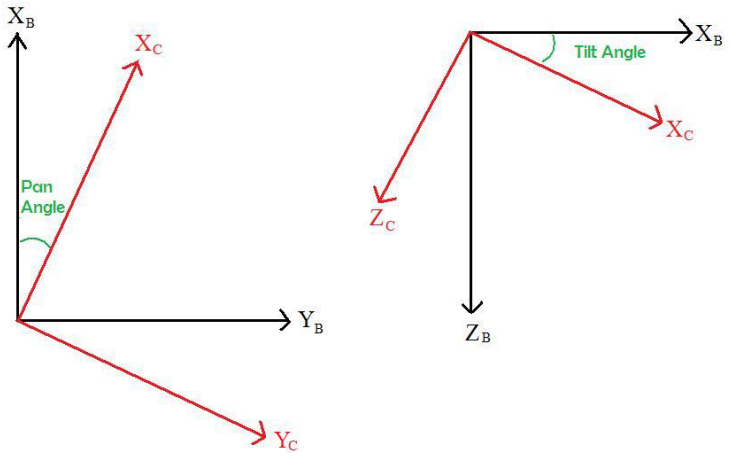


Figura 1.16: Descrizione angoli di Pan e Tilt

Il Pan-Tilt è connesso meccanicamente a due servi che sono collegati ai pin d'uscita dell'autopilota. Inoltre, è alloggiato nella parte inferiore del velivolo, appena davanti al carrello posteriore, in modo da avere il miglior campo visivo possibile e tenere il baricentro del velivolo in avanti. Esso è fissato dentro la fusoliera tramite un foro di dimensioni di circa 41 x 25 mm. La dimensione verticale occupata dal Pan-Tilt in configurazione orizzontale è di circa 5 cm sugli 8 cm totali disponibili. Un altro foro è stato realizzato più avanti per riuscire a collegare mediate cavi AV la telecamera alla scheda OSD, posta all'interno del drone.
Il Pan-Tilt scelto è il sistema *SPT100* della ServoCity, movimentato da dei servi Futaba di dimensioni standard. Ha la piattaforma di montaggio della telecamera delle dimensioni di circa 32 x 45 mm, un peso di 43 grammi e sopporta un carico massimo di 363 grammi. Le escursioni consentite di Pan e Tilt sono rispettivamente di $\pm 90°$ e 0-135°.

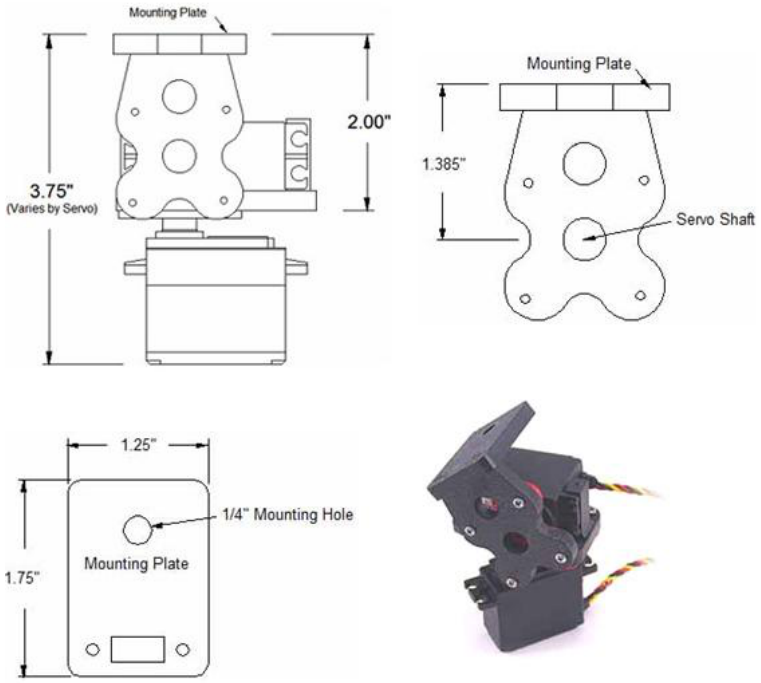


Figura 1.17: Pan-Tilt SPT 100

1.5.2 Telecamera

La telecamera montata sull'UAV è la *GoPro Hero 3* che risponde a varie specifiche tecniche fondamentali per gli obiettivi della missione: definizione FULL HD, formato PAL europeo, 30 *fps* come frequenza minima dei fotogrammi, la miglior ottica possibile per avere un buon angolo di visione, piccole dimensioni e basso peso. La definizione FULL HD (o *1080p*) indica una categoria di risoluzioni video, caratterizzate da una risoluzione verticale di 1080 linee ed una orizzontale di 1920 pixel. La risoluzione di ogni fotogramma è quindi 1920 x 1080 pixel, 2.073.600 in totale. La qualità delle immagini è davvero notevole, anche se purtroppo esse non possono essere trasmesse e visualizzate in diretta a terra con la stessa risoluzione, dato che i normali trasmettitori in commercio tagliano sostanzialmente dei pixel, facendo arrivare l'immagine a terra con una qualità inferiore. Resta comunque la possibilità di registrare le immagini a bordo tramite una scheda di memoria per poi riguardarle a terra con la qualità del FullHD.
Il formato PAL (*Phase Alternating Line*) è un metodo di codifica del colore utilizzato negli schermi analogici. Sostanzialmente differisce dal formato NTSC, usato principalmente in Giappone e Stati Uniti, per la minore velocità di aggiornamento video (50 *Hz* contro i 60 *Hz* del NTSC), ma allo stesso tempo ha una miglior risoluzione video ed una migliore gestione del colore.
La frequenza dei fotogrammi (o anche *frame rate*) è la frequenza di cattura o ripro-

duzione dei fotogrammi che compongono un filmato. Un filmato, o un'animazione al computer, è infatti una sequenza di immagini riprodotte ad una velocità sufficientemente alta da fornire, all'occhio umano, l'illusione del movimento. La frequenza dei fotogrammi viene misurata solitamente in termini di fotogrammi per secondo (*fps*). Il frame rate minimo di un filmato affinché l'apparato visivo umano percepisca il movimento deve essere di 30 *fps*.

Specifiche tecniche in dettaglio:

- Tipo di obiettivo: Lente in vetro asferica ultranitida a 6 elementi;
- Apertura: $f/2.8$(elevate prestazioni in ambienti con poca luce);
- Angolo di campo: 127° grandangolo in modalità 1080p;
- Modalità video ad alta risoluzione: 1080p = 1920 x 1080 pixel (16:9), 30 fps, velocità dati 15 Mbit/s;
- Tipo sensore: 1/2.5" HD CMOS, dimensione pixel 2.2 µm;
- HDTV out: HD NTSC e PAL;
- Tipo batteria: batteria al litio ricaricabile da 1050 mAh;
- Dimensioni (A x L x S): 42mm x 60mm x 30mm;
- Peso: 94 grammi compresa la batteria, 167 grammi compreso la custodia

Figura 1.18: Telecamera GoPro Hero 3

In figura 1.19 viene mostrata l'installazione finale dei componenti appena descritti sul velivolo in oggetto.

Figura 1.19: FPV 168 con telecamera e sistema Pan-Tilt

Capitolo 2

Sviluppo della legge di controllo

2.1 Modello matematico

La modellazione matematica è senza dubbio uno strumento teorico fondamentale perchè si possa pensare di sviluppare una legge di controllo valida. Il modello matematico di riferimento, grazie al quale è possibile descrivere il problema in esame in ogni suo aspetto, è costituito da un sistema di equazioni differenziali del primo ordine nelle quali sono evidenziate le dipendenze dai parametri geometrici, inerziali e di assetto del velivolo.

2.1.1 Sistemi di riferimento

Al fine di avere un modello matematico il più vicino possibile alla realtà, è necessario definire quattro diversi sistemi di riferimento. Le posizioni dell'UAV e del target vengono fornite all'autopilota dal modulo GPS ad esso connesso e sono espresse in termini di latitudine, longitudine e quota.
Il sistema di riferimento in oggetto è dunque il sistema WGS84 (*World Geodetic System 1984*) che è un sistema derivato dal più comune sistema ECEF (*Earth Centered Earth Fixed*), ma in cui la forma della Terra è un ellissoide schiacciato, con piano di simmetria verticale. Le sue caratteristiche sono:

- **Centro**: nel centro di massa della Terra;
- **Asse X**: passante per il meridiano di Greenwich, come definito dal BIH[6] nel 1984;
- **Asse Z**: passante per il polo Nord, come definito dal BIH nel 1984;

[6]Bureau International de l'Heure: organismo internazionale fondato nel 1917 presso l'osservatorio astronomico di Parigi con lo scopo di centralizzare ed analizzare le determinazioni di tempo effettuate presso tutti gli istituti specializzati e di contribuire alla coordinazione delle emissioni di segnali orario.

- **Asse Y**: scelto in modo da dare una terna destrorsa, ovvero tale che un osservatore posto lungo l'asse Z veda l'asse X sovrapporsi a Y con moto antiorario.

A questo sistema è associato l'ellissoide WGS84 che è descritto dai seguenti parametri:

- **Semiasse maggiore**: 6378137 m;
- **Semiasse minore**: 6356752,3142 m;
- **Schiacciamento**: 1/298,257223563;
- **Costante gravitazionale geocentrica**: 3986005 x 108 m^3/s^2.

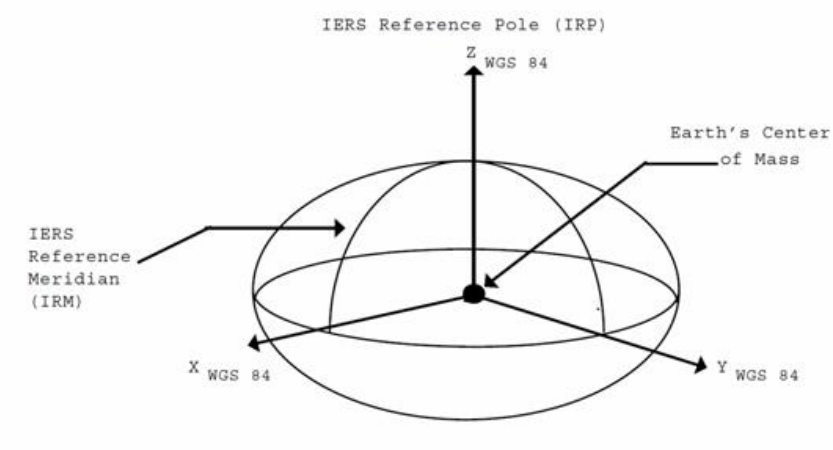


Figura 2.1: Sistema di riferimento inerziale WGS84

Come precedentemente accennato, le coordinate fornite dal GPS sono in termini di:

- **Longitudine**: angolo tra il piano contenente il meridiano di Greenwich ed il piano contenente il meridiano passante per il punto considerato;
- **Latitudine**: angolo che la verticale locale del punto considerato forma con il piano equatoriale;
- **Quota**: distanza tra il punto considerato e la sua proiezione sulla superficie dell'ellissoide rappresentante la Terra.

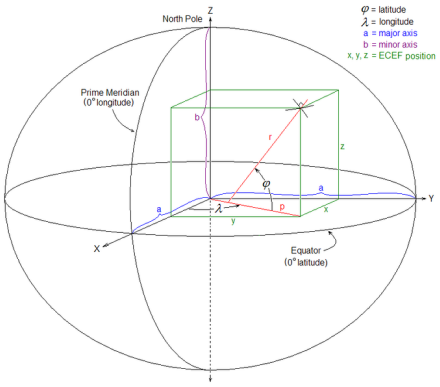


Figura 2.2: Coordinate WGS84

In questo elaborato, per una maggiore chiarezza espositiva, è stato utilizzato il sistema **NED** (***North-East-Down***). Esso 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*, ovvero quella posizione sulla superficie terrestre in cui l'UAV decolla;
- **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*).

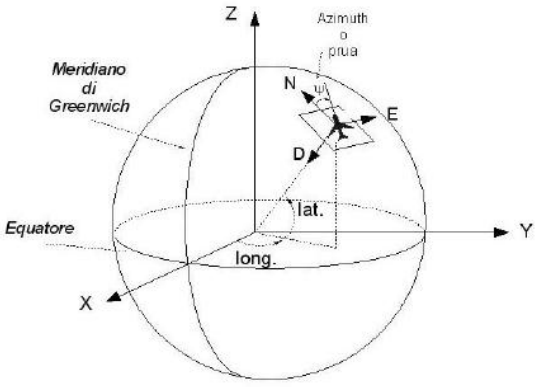


Figura 2.3: Sistema inerziale NED

Un altro sistema di riferimento molto importante è il sistema di **Assi Corpo** (***Body***). Esso è solidale al velivolo ed ha le seguenti caratteristiche:

- **Centro**: nel baricentro dell'UAV;
- **Asse X**: lungo l'asse longitudinale del velivolo;
- **Asse Z**: giace sul piano di simmetria verticale del velivolo ed è perpendicolare all'asse X;
- **Asse Y**: è l'asse trasversale al velivolo (è diretto verso l'ala destra, se si guarda il velivolo dall'alto), perpendicolarmente agli assi X e Z.

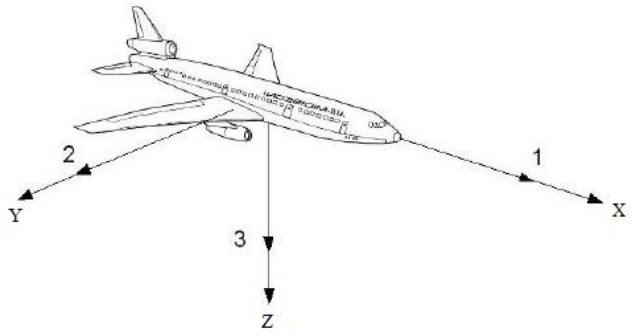


Figura 2.4: Sistema Assi Body

Infine, l'ultimo sistema di riferimento necessario a descrivere in modo corretto il problema è il sistema di riferimento **Camera**. Esso è centrato nel baricentro della telecamera (per semplicità si suppone che il baricentro del velivolo e della telecamera coincidano) e ha le stesse caratteristiche del sistema Body, muovendosi però rispetto ad esso tramite delle rotazioni di Pan e Tilt (si supponga che per angoli di Pan e Tilt nulli i due sistemi coincidano).

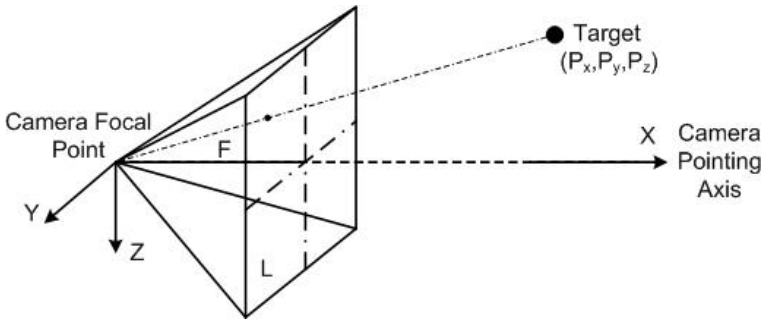


Figura 2.5: Sistema Assi Camera

2.1.2 Modelli cinematici del punto materiale

La cinematica del velivolo è modellata attraverso il seguente sistema di equazioni differenziali:

$$\begin{cases} \dot{x} = V\cos(\gamma)\cos(\chi) \\ \dot{y} = V\cos(\gamma)\sin(\chi) \\ \dot{z} = V\sin(\gamma) \end{cases} \tag{2.1}$$

dove V esprime la velocità dell'UAV, γ è l'angolo di pendenza della traiettoria (negativo per movimenti verso l'alto) e χ è la direzione della prua. Per semplicità, possiamo supporre che l'angolo di prua χ sia all'incirca uguale all'angolo ψ (UAV Heading) e che l'angolo γ corrisponda all'angolo θ (UAV Pitch) secondo la relazione $\theta = \gamma + \alpha$ non avendo sensori che ci diano informazioni sull'angolo d'attacco α (risulta infatti $\theta = \gamma$). Aggiungendo le componenti del vento in direzione x e y rispettivamente W_x e W_y, le equazioni si modificano in questo modo:

$$\begin{cases} \dot{x} = V\cos(\theta)\cos(\psi) + W_x \\ \dot{y} = V\cos(\theta)\sin(\psi) + W_y \\ \dot{z} = V\sin(\theta) \end{cases} \tag{2.2}$$

Le condizioni iniziali da imporre a questo sistema di equazioni differenziali sono le posizioni e l'assetto iniziali del velivolo. Allo stesso modo si può pensare di definire il modello cinematico del target, inteso anch'esso come punto materiale, facendo l'ipotesi aggiuntiva che esso non subisca variazioni di velocità verticale ($\gamma = \theta = 0$):

$$\begin{cases} \dot{x} = V_T\cos(\chi_T) \\ \dot{y} = V_T\sin(\chi_T) \\ \dot{z} = 0 \end{cases} \tag{2.3}$$

Le condizioni iniziali da imporre a questo sistema di equazioni differenziali sono le posizioni e l'assetto iniziali del target.

2.1.3 Cinematica relativa

Per far sì che il modello matematico del problema generale risulti ancora più preciso ed affidabile, si può pensare come prima cosa di integrare il modello (2.2) con una vera e propria legge di guida che consenta all'UAV di rimanere 'attorno' alla posizione corrente del target. Si è dunque ricavato il modello matematico delle matrici relative nelle condizioni di vento nullo $W_x = W_y = 0$.
Guardando la figura 2.6, si definisce il versore $\widehat{LOS}$[7] come:

[7] Linea di vista (Line-of-Sight)

$$\widehat{LOS} = \frac{\vec{R}}{R} = \begin{bmatrix} \cos(\sigma_V)\cos(\sigma_H) \\ \cos(\sigma_V)\sin(\sigma_H) \\ \sin(\sigma_V) \end{bmatrix} \tag{2.4}$$

in cui $\vec{R}$ è il vettore posizione relativa tra UAV e target e $R = \left\|\vec{R}\right\|$ è la rispettiva distanza relativa.

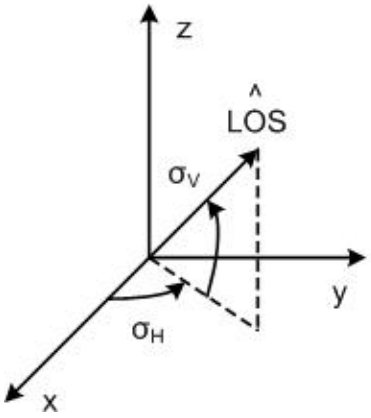


Figura 2.6: Line of Sight

Si definiscono anche le seguenti derivate:

$$\dot{\widehat{LOS}} = \begin{bmatrix} -\dot{\sigma}_V \sin(\sigma_V)\cos(\sigma_H) - \dot{\sigma}_H \cos(\sigma_V)\sin(\sigma_H) \\ -\dot{\sigma}_V \sin(\sigma_V)\sin(\sigma_H) + \dot{\sigma}_H \cos(\sigma_V)\cos(\sigma_H) \\ \dot{\sigma}_V \cos(\sigma_V) \end{bmatrix}$$

$$\dot{\vec{R}} = \vec{V}_T - \vec{V}_{UAV} = \begin{bmatrix} V_T \cos(\chi_T)\cos(\gamma_T) \\ V_T \sin(\chi_T)\cos(\gamma_T) \\ V_T \sin(\gamma_T) \end{bmatrix} - \begin{bmatrix} V\cos(\psi)\cos(\theta) \\ V\sin(\psi)\cos(\theta) \\ V\sin(\theta) \end{bmatrix}$$

percui si ottiene:

$$\dot{R} = \dot{R}\widehat{LOS} + R\dot{\widehat{LOS}} = \vec{V}_T - \vec{V}_{UAV} \tag{2.5}$$

che può essere riscritta per componenti nel seguente modo:

$$\begin{cases} \dot{R}c(\sigma_V)c(\sigma_H) + R(-\dot{\sigma}_V s(\sigma_V)c(\sigma_H) - \dot{\sigma}_H c(\sigma_V)s(\sigma_H)) = V_T c(\chi_T)c(\gamma_T) - Vc(\psi)c(\theta) \\ \dot{R}c(\sigma_V)s(\sigma_H) + R(-\dot{\sigma}_V s(\sigma_V)s(\sigma_H) + \dot{\sigma}_H c(\sigma_V)c(\sigma_H)) = V_T s(\chi_T)c(\gamma_T) - Vs(\psi)c(\theta) \\ \dot{R}s(\sigma_V) + R\dot{\sigma_V}c(\sigma_V) = V_T s(\gamma_T) - Vs(\theta) \end{cases} \tag{2.6}$$

dove $s()$ e $c()$ rappresentano rispettivamente le abbreviazioni di sin() e cos(). Moltiplicando la prima equazione delle (2.6) per $\sin(\sigma_H)$ e la seconda per $\cos(\sigma_H)$ e sottraendo le due equazioni ottenute, si ha:

$$R\dot{\sigma}_H \cos(\sigma_V) = V_T \sin(\chi_T - \sigma_H)\cos(\gamma_T) - V \sin(\psi - \sigma_H)\cos(\theta) \tag{2.7}$$

Invece, moltiplicando la prima equazione delle (2.6) per $\cos(\sigma_H)$ e la seconda per $\sin(\sigma_H)$ e sommando le due equazioni ottenute, si ha:

$$\dot{R}\cos(\sigma_V) - R\dot{\sigma}_V \sin(\sigma_V) = V_T \cos(\chi_T - \sigma_H)\cos(\gamma_T) - V\cos(\psi - \sigma_H)\cos(\theta) \tag{2.8}$$

Infine, moltiplicando la terza equazione delle (2.6) per $\cos(\sigma_V)$ e l'equazione (2.8) per $\sin(\sigma_V)$ e sottraendo le due equazioni ottenute, si ha:

$$R\dot{\sigma}_V = V_T[s(\gamma_T)c(\sigma_V) - c(\chi_T - \sigma_H)c(\gamma_T)s(\sigma_V)] - V[s(\theta)c(\sigma_V) - c(\psi - \sigma_H)c(\theta)s(\sigma_V)] \tag{2.9}$$

Le equazioni (2.7), (2.8) e (2.9) descrivono dunque la cinematica relativa tra l'UAV e il target. Inoltre, supponendo che il target non subisca variazioni di velocità verticale ($\gamma_T = 0$), il modello che si ottiene risulta ancor più semplificato:

$$\begin{cases} R\dot{\sigma}_H \cos(\sigma_V) = V_T \sin(\chi_T - \sigma_H) - V \sin(\psi - \sigma_H) \\ \dot{R}\cos(\sigma_V) - R\dot{\sigma}_V \sin(\sigma_V) = V_T \cos(\chi_T - \sigma_H) - V\cos(\psi - \sigma_H) \\ R\dot{\sigma}_V = -V_T \cos(\chi_T - \sigma_H)\sin(\sigma_V) + V\cos(\psi - \sigma_H)\sin(\sigma_V) \\ \dot{\psi} = \frac{a_n}{V} \end{cases} \tag{2.10}$$

Considerando in aggiunta le relazioni che intercorrono tra le cinematiche singole di target ed UAV e la loro cinematica relativa, risulta anche:

$$\tan(\psi) = \frac{\dot{y}}{\dot{x}} \tag{2.11}$$

$$R = \sqrt{(x_T - x)^2 + (y_T - y)^2 + (z_T - z)^2} \tag{2.12}$$

Una rappresentazione geometrica delle variabili appena descritte è mostrata in figura 2.7.
Agendo inoltre sull'accelerazione laterale a_n del velivolo, il modello fornisce una descrizione completa del problema di Target Tracking. La legge di guida implementata è descritta da:

$$a_n = \dot{\psi} V = K_1(R, \dot{R}) \arctan(K_2(\sigma_H - \psi)) \tag{2.13}$$

dove

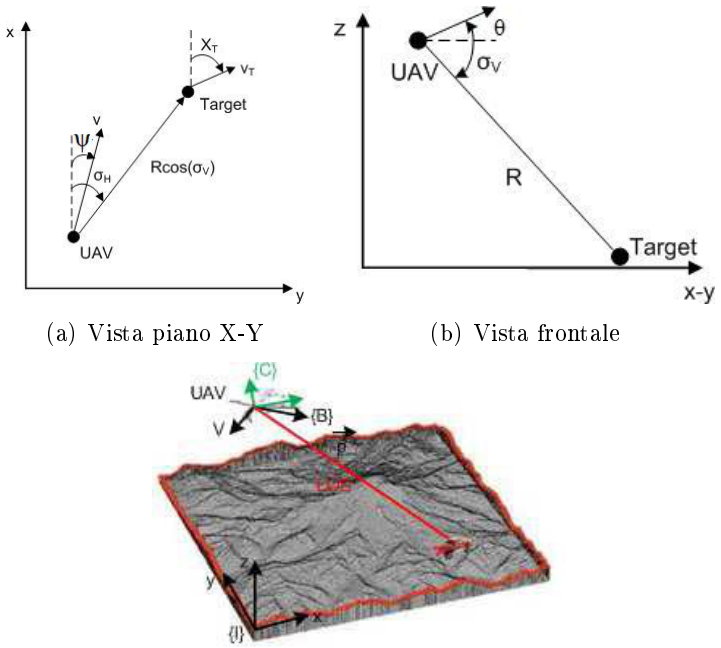
- $K_1() : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}^+$;
- K_2 è considerato positivo nell'intervallo]0,1]

In particolare, la funzione K_1 fornisce informazioni sul modulo dell'accelerazione laterale e deve essere minima quando l'UAV supera il target ma si trova comunque nelle sue vicinanze in un'area circolare specificata tramite un raggio predefinito R_0, massima negli altri casi.

	$\mathbf{R < R_0}$	$\mathbf{R \geq R_0}$
$\mathbf{\dot{R} < 0}$	C	C
$\mathbf{\dot{R} \geq 0}$	0	C

Tabella 2.1: Selezione guadagno K_1 per la legge di guida

Perchè si ottenga un corretto Target-Tracking, sono stati utilizzati i valori di $C = 3.6057$, $R_0 = 57.8112$ m e $K_2 = 5$.



(a) Vista piano X-Y

(b) Vista frontale

(c) Vista 3D

Figura 2.7: Cinematica Relativa

2.1.4 Equazioni per il tracking mediante telecamera

Al fine di progettare una legge di controllo in grado di garantire che la telecamera mantenga al centro del display il target, ogni termine dell'equazione deve essere riferito esclusivamente al sistema di riferimento Camera. Per arrivare a questo, è necessario passare attraverso diverse matrici di trasformazione.
La prima trasformazione che può essere fatta è quella per ottenere le coordinate delle posizioni dell'UAV e del target nel sistema di riferimento inerziale NED, dato che queste informazioni vengono fornite dal sensore GPS in termini di longitudine, latitudine e quota, ovvero rispetto alla terna di assi WGS84. In particolare, questo cambio di coordinate può essere fatto per gradi, ovvero passando prima da coordinate WGS84 a coordinate ECEF e poi da coordinate ECEF a coordinate NED. Infatti, una volta fissati i vettori:

- $\vec{p} = [\lambda, \phi, h]^T$, coordinate di un punto in termini di longitudine, latitudine e quota nella terna WGS84;
- $\vec{O}_{NED} = [\lambda_O, \phi_O, h_O]^T$, posizione dell'origine del sistema inerziale NED.

il cambio di coordinate dal sistema WGS84 al sistema ECEF viene fatto prendendo in considerazione dei dati geometrici riferiti al modello di ellissoide utilizzato per descrivere la forma della Terra. Più precisamente, utilizzando le quantità:

- $a = 6378137$ m, semiasse maggiore dell'ellissoide terrestre;
- $b = 6356752.3142$ m, semiasse minore dell'ellissoide terrestre;
- $e = \sqrt{1 - \frac{b^2}{a^2}}$, eccentricità dell'ellissoide,

la trasformazione di coordinate avviene attraverso le relazioni:

$$\begin{cases} N(\phi) = \frac{a}{\sqrt{1-e^2\sin^2(\phi)}} \\ \vec{p}^{ECEF} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{ECEF} = \begin{bmatrix} (N(\phi)+h)\cos(\lambda)\cos(\phi) \\ (N(\phi)+h)\sin(\lambda)\cos(\phi) \\ (N(\phi)(1-e^2)+h)\sin(\phi) \end{bmatrix} \end{cases} \tag{2.14}$$

dove $N(\phi)$ è la distanza tra la superficie dell'ellissoide e l'asse di rotazione z calcolata lungo la verticale locale.
A questo punto, per ottenere le coordinate del vettore $\vec{p}$ nel sistema di riferimento inerziale NED, basta definire la matrice di trasformazione delle coordinate da ECEF a NED:

$$R_{ECEF}^{NED} = \begin{bmatrix} -\sin(\phi_O)\cos(\lambda_O) & -\sin(\phi_O)\sin(\lambda_O) & \cos(\phi_O) \\ -\sin(\lambda_O) & \cos(\lambda_O) & 0 \\ -\cos(\phi_O)\cos(\lambda_O) & -\cos(\phi_O)\sin(\lambda_O) & -\sin(\phi_O) \end{bmatrix}$$

e moltiplicarla per il vettore posizione $(\vec{p}^{ECEF} - \vec{O}_{NED}^{ECEF})$:

$$\vec{p}^{NED} = R_{ECEF}^{NED}(\vec{p}^{ECEF} - \vec{O}_{NED}^{ECEF}) \tag{2.15}$$

A questo punto, il vettore $\vec{p}^{NED}$ può essere moltiplicato per la matrice di trasformazione R_I^B, definita in base agli angoli di Eulero (ϕ, θ, ψ) che descrivono l'orientamento del sistema Body rispetto a quello inerziale. Il risultato ottenuto è dunque il vettore $\vec{p}^B$ espresso in coordinate Body, secondo la relazione:

$$\vec{p}^B = R_I^B \vec{p}^{NED} \tag{2.16}$$

in cui $R_I^B = (R_B^I)^{-1} = (R_B^I)^T$. In particolare è $R_I^B = R_I^B(\phi, \theta, \psi)$ con

$$R_I^B = \begin{bmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ s(\theta)s(\phi)c(\psi) - c(\phi)s(\psi) & s(\theta)s(\phi)s(\psi) + c(\phi)c(\psi) & s(\phi)c(\theta) \\ s(\theta)c(\phi)c(\psi) + s(\phi)s(\psi) & s(\theta)c(\phi)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\theta) \end{bmatrix}$$

Infine, moltiplicando il vettore ottenuto per la matrice di trasformazione dal sistema di assi Body al sistema di riferimento Camera R_B^C, si ottengono le componenti del vettore $\vec{p}^C$ in assi Camera, secondo la relazione $\vec{p}^C = R_B^C \vec{p}^B$. La matrice R_B^C si può esprimere come:

$$R_B^C = \begin{bmatrix} c(\theta_c)c(\psi_c) & c(\theta_c)s(\psi_c) & -s(\theta_c) \\ -s(\psi_c) & c(\psi_c) & 0 \\ s(\theta_c)c(\psi_c) & s(\theta_c)s(\psi_c) & c(\theta_c) \end{bmatrix}$$

intendendo con la terna $(0, \theta_c, \psi_c)$ gli angoli che definiscono l'orientamento del sistema Camera rispetto agli assi Body (l'angolo θ_c corrisponde all'angolo di Tilt della telecamera, mentre l'angolo ψ_c è relativo al movimento di Pan).
Il vettore di nostro interesse è quello relativo alle posizioni relative tra target e UAV nel sistema di riferimento Camera, ovvero il vettore $\vec{P}_{rel}^C = [P_x, P_y, P_z]^T$, ed è ricavabile ragionando sui corrispettivi vettori velocità del target e del velivolo e sui vettori delle velocità angolari. Infatti, il vettore della velocità del target nel sistema Camera si può esprimere come:

$$\vec{V}_T^C = \vec{V}_{UAV}^C + \vec{V}_{rel/B}^C + \vec{\omega}_B^C \times \vec{P}_{rel}^C \tag{2.17}$$

dove il vettore $\vec{V}_{rel/B}^C$ rappresenta il vettore delle velocità relative tra target e UAV in assi Camera ricavabile dalla formula:

$$\vec{V}^C_{rel/B} = \dot{\vec{P}}^C_{rel} + \vec{\omega}^C_{B/C} \times \vec{P}^C_{rel} \tag{2.18}$$

Mettendo a sistema le due equazioni ed esplicitando il vettore $\dot{\vec{P}}^C_{rel}$ di nostro interesse, otteniamo:

$$\dot{\vec{P}}^C_{rel} = \vec{V}^C_T - \vec{V}^C_{UAV} - (\vec{\omega}^C_B + \vec{\omega}^C_{B/C}) \times \vec{P}^C_{rel} \tag{2.19}$$

In particolare, nell'equazione (2.19) sono stati definiti:

- $\vec{V}^C_{UAV} = R^C_I \cdot \vec{V}^I_{UAV} = R^C_B R^B_I \cdot \vec{V}^I_{UAV}$, ovvero il vettore velocità dell'UAV nel sistema di riferimento Camera è inteso come prodotto del vettore velocità dell'UAV nel sistema di riferimento inerziale secondo il modello cinematico del velivolo per la matrice di trasformazione da assi Inerziali ad assi Camera;

- $\vec{V}^C_T = R^C_I \cdot \vec{V}^I_T = R^C_B R^B_I \cdot \vec{V}^I_T$, ovvero il vettore velocità del target nel sistema di riferimento Camera è inteso come prodotto del vettore velocità del target nel sistema di riferimento inerziale secondo il modello cinematico del target per la matrice di trasformazione da assi Inerziali ad assi Camera;

- $\vec{\omega}^C_B = R^C_B \cdot \vec{\omega}^B$, ovvero il vettore delle velocità angolari di rotazione del velivolo nel sistema di riferimento Camera è inteso come prodotto del vettore delle velocità angolari di rotazione del velivolo $\vec{\omega}^B = [p,\ q,\ r]^T$ nel sistema di riferimento Body per la matrice di trasformazione da assi Body ad assi Camera (in particolare, le componenti del vettore $\vec{\omega}^B$ rappresentano rispettivamente le velocità angolari di rollio, beccheggio ed imbardata);

- $\vec{\omega}^C_{B/C}$, ovvero il vettore delle velocità angolari di rotazione del sistema di riferimento Camera rispetto al sistema di riferimento Body che deriva dalle inverse delle *Equazioni cinematiche di Eulero di propagazione dell'assetto.* Infatti, considerando il vettore velocità angolare in assi Body $\vec{\omega}^B = [p,\ q,\ r]^T$, vale la relazione:

$$\begin{cases} p = \dot{\phi} - \dot{\psi}\sin(\theta) \\ q = \dot{\theta}\cos(\phi) + \dot{\psi}\cos(\theta)\sin(\phi) \\ r = -\dot{\theta}\sin(\phi) + \dot{\psi}\cos(\theta)\cos(\phi) \end{cases} \tag{2.20}$$

  Dunque nel nostro caso, ponendo l'angolo di bank ϕ_c e la corrispettiva velocità angolare $\dot{\phi}_c$ nulli, diventa:

$$\vec{\omega}_{B/C}^{C} = \begin{bmatrix} -\dot{\psi}_c \sin(\theta_c) \\ \dot{\theta}_c \\ \dot{\psi}_c \cos(\theta_c) \end{bmatrix} \tag{2.21}$$

Il modello (2.19) costituisce un sistema di equazioni differenziali del primo ordine nell'incognita $\vec{P}_{rel}^{C}$ che descrive dinamicamente l'andamento delle posizioni relative tra target e UAV nel sistema di riferimento Camera.
Inoltre, si può pensare di trovare un legame tra vettore $\vec{P}_{rel}^{C}$ ed i pixels visualizzati sullo schermo $\vec{P}_{im}^{C} = [f, y_{im}, z_{im}]^T$ in modo da poter visualizzare il corretto centraggio nell'inquadratura del target. Per far questo, intervengono delle relazioni tra le posizioni del target in assi Camera, le caratteristiche proprie della lente della telecamera e la posizione dei pixels visualizzati a partire dal centro dello schermo y_{ip} e z_{ip}. In particolare definendo f come la lunghezza focale della lente, S_y e S_z i fattori di conversione pixel/metro, 0_y e 0_z i fattori di traslazione del sistema di riferimento dello schermo, si può scrivere:

$$\begin{cases} y_{im} = (y_{ip} - 0_y) \cdot S_y \\ z_{im} = (z_{ip} + 0_z) \cdot S_z \end{cases} \tag{2.22}$$

e inoltre

$$\begin{cases} y_{im} = f \cdot \frac{P_y}{P_x} \\ \\ z_{im} = f \cdot \frac{P_z}{P_x} \end{cases} \tag{2.23}$$

In definitiva, risulta:

$$\begin{cases} y_{ip} = \frac{f \cdot P_y}{S_y \cdot P_x} + 0_y \\ \\ z_{ip} = \frac{f \cdot P_z}{S_z \cdot P_x} - 0_z \end{cases} \tag{2.24}$$

Le unità di misura di y_{ip} e z_{ip} sono pixels, mentre le unità di misura di y_{im} e z_{im} sono metri. Nel nostro caso, $S_y = S_z = 2.64583 \cdot 10^{-4}$ metri ed $f = 6 \cdot 10^{-3}$ metri.

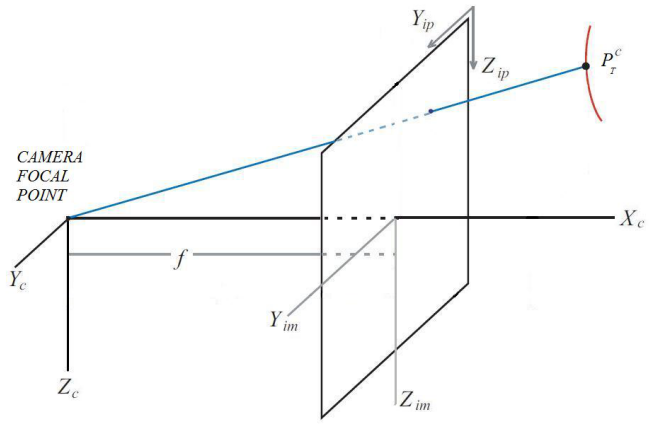


Figura 2.8: Visualizzazione Pixel su schermo

2.2 Legge di controllo

Per ottenere un corretto ed affidabile tracking mediante telecamera, è necessario che si progetti una legge di controllo che abbia come ingressi le velocità angolari di Pan e Tilt della telecamera in modo che il target sia costantemente centrato nello schermo. La legge di controllo è non lineare e la sua descrizione è basata su una tecnica di linearizzazione input-output che permette di guidare i pixels della posizione del target nel sistema di riferimento Camera verso la sua origine in maniera esponenzialmente veloce. In particolare, la tecnica in questione si chiama *Feedback Linearization*. É una tecnica molto semplice per rendere stabile un sistema non lineare, ma può risultare inutile per varie ragioni quali le incertezze dei parametri in gioco, per la semplificazione del modello e per gli errori computazionali. Si rendono dunque necessari, una volta descritta la legge di controllo, degli studi sulla robustezza e sull'efficienza del controllo stesso e la valutazione dei risultati delle simulazioni effettuate.

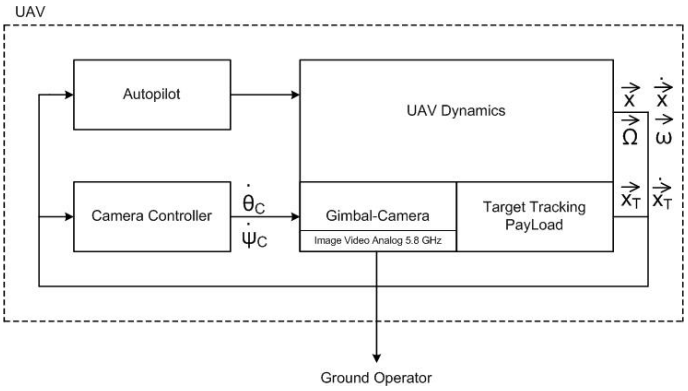


Figura 2.9: Schema a blocchi legge di controllo

2.2.1 Feedback Linearization

Il controllo in feedback linearization è una tecnica di base utilizzata nel controllo di sistemi non lineari. Quest'approccio consiste nella trasformazione di un sistema non lineare in un equivalente sistema lineare, grazie a un cambio di variabili e ad un ingresso appositamente scelto. La feedback linearization può essere applicata a quei sistemi non lineari che possono essere scritti nella seguente forma:

$$\begin{cases} \dot{\vec{x}} = f(\vec{x}, t) + g(\vec{x}, t)\vec{u} \\ \vec{y} = h(\vec{x}) \end{cases} \tag{2.25}$$

in cui $\vec{x}$ è il vettore di stato, $\vec{u}$ è il vettore d'ingresso e $\vec{y}$ il vettore d'uscita. Lo scopo è quello di realizzare un vettore d'ingresso tale che la funzione ingresso-uscita tra il nuovo ingresso e l'uscita sia lineare. A questo punto può essere applicata una normale strategia di controllo di sistemi lineari. Definendo le quantità:

- $\dot{x}^C_{rel} = \dot{x}^C_T - \dot{x}^C_{UAV}$
- $\dot{y}^C_{rel} = \dot{y}^C_T - \dot{y}^C_{UAV}$
- $\dot{z}^C_{rel} = \dot{z}^C_T - \dot{z}^C_{UAV}$

il modello matematico (2.19) può essere riscritto per componenti:

$$\begin{cases} \dot{P}_x = \dot{x}^C_{rel} - P_z q_c + P_y r_c - P_z \dot{\theta}_c + P_y \dot{\psi}_c \cos(\theta_c) \\ \dot{P}_y = \dot{y}^C_{rel} - P_x r_c + P_z p_c - P_x \dot{\psi}_c \cos(\theta_c) - P_z \dot{\psi}_c \sin(\theta_c) \\ \dot{P}_z = \dot{z}^C_{rel} - P_y p_c + P_x q_c + P_y \dot{\psi}_c \sin(\theta_c) + P_x \dot{\theta}_c \end{cases} \tag{2.26}$$

in cui $\vec{\omega}^C_B = R^C_B \cdot \vec{\omega}^B = [p_c, q_c, r_c]^T$ e, tenendo conto anche delle relazioni (2.24) (si considerino gli offset $0_y = 0_z = 0$), può essere posto nella forma dell'equazione (2.25) in cui:

$$f(\vec{x}, t) = \begin{bmatrix} \dot{x}^C_{rel} - P_z q_c + P_y r_c \\ \dot{y}^C_{rel} - P_x r_c + P_z p_c \\ \dot{z}^C_{rel} - P_y p_c + P_x q_c \\ 0 \\ 0 \end{bmatrix} \qquad g(\vec{x}, t) = \begin{bmatrix} -P_z & P_y \cos(\theta_c) \\ 0 & -P_x \cos(\theta_c) - P_z \sin(\theta_c) \\ P_x & P_y \sin(\theta_c) \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

e

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_c \\ \dot{\psi}_c \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y_{ip} \\ z_{ip} \end{bmatrix} \qquad h(\vec{x}) = \frac{f}{P_x} \begin{bmatrix} \frac{P_y}{S_y} \\ \frac{P_z}{S_z} \end{bmatrix}$$

Come si evince dalle equazioni, si è tenuto conto anche della dipendenza dal tempo t in quanto sono presenti all'interno del modello matematico delle variabili (ad

esempio, le velocità di target e UAV o l'assetto del velivolo) che rendono il sistema non stazionario.
Nella teoria, un sistema descritto nella forma del (2.25) è detto *input-output linearizzabile* se esiste un diffeomorfismo[8] $T : D \to \mathbb{R}^5$ tale che $D_z = T(D)$ contiene l'origine ed il cambiamento di variabile $\vec{z} = T(\vec{x})$ trasforma il sistema nella forma:

$$\dot{\vec{z}} = A\vec{z} + B\gamma(\vec{x})(\vec{u} - \alpha(\vec{x})) \tag{2.27}$$

con (A, B) controllabili e $\gamma(\vec{x})$ non singolare per tutte le $\vec{x} \in D$.
In questo caso, la derivata $\dot{\vec{y}}$ è data dalla forma:

$$\dot{\vec{y}} = L_f h(\vec{x}, t) + L_g h(\vec{x}, t)\vec{u} \tag{2.28}$$

dove

$$L_f h(\vec{x}, t) = \frac{\partial h}{\partial \vec{x}} f(\vec{x}, t) = \frac{f}{P_x} \begin{bmatrix} \frac{1}{S_y}(\dot{y}^C_{rel} - P_x r_c + P_z p_c) - \frac{P_y}{S_y P_x}(\dot{x}^C_{rel} - P_z q_c + P_y r_c) \\ \frac{1}{S_z}(\dot{z}^C_{rel} - P_y p_c + P_x q_c) - \frac{P_z}{S_z P_x}(\dot{x}^C_{rel} - P_z q_c + P_y r_c) \end{bmatrix} \tag{2.29}$$

e

$$L_g h(\vec{x}, t) = \frac{\partial h}{\partial \vec{x}} g(\vec{x}, t) = \frac{f}{P_x} \begin{bmatrix} \frac{P_y P_z}{S_y P_x} & \frac{-P_y^2 \cos(\theta_C)}{S_y P_x} - \frac{P_x \cos(\theta_C) + P_z \sin(\theta_C)}{S_y} \\ \frac{P_z^2}{S_z P_x} + \frac{P_x}{S_z} & \frac{-P_y P_z \cos(\theta_C)}{S_z P_x} + \frac{P_y \sin(\theta_C)}{S_z} \end{bmatrix} \tag{2.30}$$

$L_f h(\vec{x}, t)$ e $L_g h(\vec{x}, t)$ rappresentano rispettivamente le *derivate di Lie*[9] di h lungo f e di h lungo g. Quando $L_g h(\vec{x}, t)$ è non singolare, gli ingressi possono essere scritti come:

$$\vec{u} = \alpha(\vec{x}, t) + K\beta(\vec{x}, t)\vec{\xi} \tag{2.31}$$

in cui

$$\begin{aligned} \alpha(\vec{x}, t) &= -L_g^{-1} h(\vec{x}, t) L_f h(\vec{x}, t) \\ \beta(\vec{x}, t) &= L_g^{-1} h(\vec{x}, t) \end{aligned} \tag{2.32}$$

[8] Un diffeomorfismo è una funzione tra due varietà differenziabili con la proprietà di essere differenziabile, invertibile e di avere l'inversa differenziabile.

[9] La *derivata di Lie* calcola la variazione di un campo vettoriale, più in generale di un campo tensoriale, lungo il flusso di un altro campo vettoriale. L'idea alla base della derivata di Lie è quella di confrontare due tensori lungo una stessa curva che è soluzione di un opportuno campo vettoriale, facendo il limite per lo spostamento infinitesimale.

quindi la nuova funzione ingresso-uscita si riduce alla forma:

$$\dot{\vec{y}} = K\vec{\xi} \tag{2.33}$$

dove $\vec{\xi} = [P_y, P_z]^T$ ed il parametro K è una matrice di dimensioni 2 x 2 che contiene dei parametri derivanti dallo studio di robustezza (si veda 2.3).

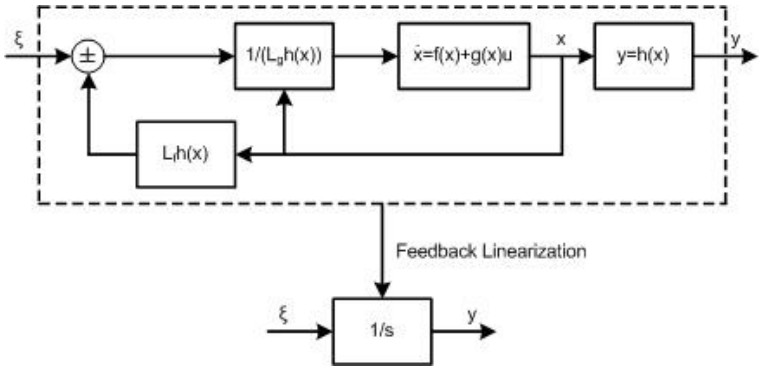


Figura 2.10: Feedback Linearization

É importante analizzare la stabilità interna del sistema applicando la legge di controllo, considerando anche il cambiamento di variabili:

$$\vec{z} = T(\vec{x}) = \begin{bmatrix} \phi_1(\vec{x}) \\ \phi_2(\vec{x}) \\ \phi_3(\vec{x}) \\ -- \\ h_1(\vec{x}) \\ h_2(\vec{x}) \end{bmatrix} = \begin{bmatrix} \vec{\lambda} \\ -- \\ \vec{\xi} \end{bmatrix} \tag{2.34}$$

in cui $\vec{\phi}(\vec{x}) = [\phi_1, \phi_2, \phi_3]^T$ è scelto tenendo conto del fatto che $T(\vec{x})$ è un diffeomorfismo di dominio D tale che

$$\frac{\partial \vec{\phi}}{\partial \vec{x}} g(\vec{x}, t) = 0 \qquad\qquad \forall \vec{x} \in \mathbb{R}^5 \tag{2.35}$$

Il sottosistema derivato dall'equazione (2.25) è, dunque, il seguente:

$$\begin{cases} \dot{\vec{\lambda}} = f_0(\vec{\lambda}, \vec{\xi}) \\ \dot{\vec{\xi}} = \gamma(\vec{x}, t)(\vec{u} - \alpha(\vec{x}, t)) \\ \vec{y} = \vec{\xi} \end{cases} \tag{2.36}$$

dove

$$\gamma(\vec{x},t) = \beta^{-1}(\vec{x},t) = L_g h(\vec{x},t) \tag{2.37}$$

In particolare, l'equazione (2.36) può essere decomposta in una parte interna $\vec{\lambda} = [P_x, \theta_c, \psi_c]^T \in \mathbb{R}^3$ ed una parte esterna $\vec{\xi} \in \mathbb{R}^2$. La prima non è osservabile dal controllore descritto, ma è comunque importante sottolineare che $\alpha(\vec{x},t)$ e $\beta(\vec{x},t)$ sono univocamente determinate dalle funzioni f, g e h (non c'è dunque dipendenza dalla scelta di ϕ).
La funzione $\vec{\phi}(\vec{x})$ può essere scritta, invece, separando le variabili:

$$\begin{aligned} \phi_1(\vec{x}) &= P_x + P_z\theta_c - P_y \sin(\theta_c) \\ \phi_2(\vec{x}) &= \theta_c - \psi_c \\ \phi_3(\vec{x}) &= -\theta_c + \psi_c \end{aligned} \tag{2.38}$$

in modo che si verifichi la condizione $\vec{\phi}(0) = 0$. La dinamica interna del controllo è descritta dalla prima equazione delle (2.36) imponendo $\vec{\xi} = 0$:

$$\dot{\vec{\lambda}} = f_0(\vec{\lambda}, 0) \tag{2.39}$$

ed è chiamata *dinamica zero*. Essa è caratterizzata dal fatto che nelle coordinate d'origine vale

$$\vec{y} \equiv 0 \rightarrow \vec{\xi} \equiv 0 \rightarrow \vec{u} \equiv \alpha(\vec{x}) \tag{2.40}$$

Se l'uscita è identicamente nulla, la soluzione dell'equazione degli stati deve essere compresa nel seguente insieme:

$$Z = \left\{ \vec{x} \in \mathbb{R}^5 : P_y = P_z = 0 \right\} \tag{2.41}$$

e il vettore d'ingresso deve dunque essere:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} q_c - \frac{\dot{z}^C_{rel}}{P_x} \\ \frac{1}{\cos(\theta_c)}(\frac{\dot{y}^C_{rel}}{P_x} - r_c) \end{bmatrix} \tag{2.42}$$

In conclusione, il controllore è descritto dalle seguenti equazioni:

$$\begin{cases} \dot{P}_x = \dot{x}^C_{rel} \\ \dot{\theta}_c = q_c - \frac{\dot{z}^C_{rel}}{P_x} \\ \dot{\psi}_c = \frac{1}{\cos(\theta_c)}(\frac{\dot{y}^C_{rel}}{P_x} - r_c) \end{cases} \tag{2.43}$$

Come si può evincere anche da uno studio empirico del problema, non essendo un sistema a fase minima, non ha un punto di equilibrio asintoticamente stabile nel dominio d'interesse. Il sistema (2.43) fornisce quindi l'andamento delle velocità angolari di Pan e Tilt con le quali si ottiene un corretto tracking automatico con la telecamera.

2.3 Robustezza

Per dare una più generale descrizione del problema in tutti i suoi aspetti, è stata fatta anche un'analisi sulla robustezza. Il controllo robusto è una strategia di controllo automatico dei sistemi dinamici il cui scopo è il controllo del sistema interessato anche quando ci sono delle ambiguità, ovvero quando non si ha una conoscenza completa delle variabili che intervengono nel sistema stesso. Il controllo automatico classico di un sistema dinamico conosciuto completamente e descritto in forma di stato o dalla sua funzione di trasferimento genera, tramite opportuni algoritmi scelti dal progettista, un controllore adatto per quel particolare sistema. In pratica ciò non è possibile perché il sistema che si va a controllare non corrisponde completamente al sistema reale che ne è appunto un modello, cioè un'approssimazione. Un controllo robusto riesce a risolvere questo problema e garantisce la stabilità asintotica per un insieme di sistemi e non solo per quello nominale.
Nel nostro caso, la tecnica di feedback linearization si basa principalmente sull'esatta semplificazione matematica dei termini non lineari α e γ quindi c'è bisogno di una loro perfetta conoscenza, cosa improbabile nella maggior parte dei casi per varie ragioni, come ad esempio gli errori computazionali o l'incertezza dei parametri dai quali dipendono. Infatti, supponendo di avere delle approssimazioni di α e β, si può scrivere:

$$\vec{u} = \hat{\alpha}(\vec{x}, t) + K\hat{\beta}(\vec{x}, t)\vec{\xi} \tag{2.44}$$

Sostituendo alle equazioni (2.36), si ottiene:

$$\begin{cases} \dot{\vec{\lambda}} = f_0(\vec{\lambda}, \vec{\xi}) \\ \dot{\vec{\xi}} = \gamma(\vec{x}, t)(\hat{\alpha}(\vec{x}, t) + K\hat{\beta}(\vec{x}, t)\vec{\xi} - \alpha(\vec{x}, t)) \end{cases} \tag{2.45}$$

Aggiungendo e sottraendo il termine $K\vec{\xi}$ alla seconda delle (2.45) e definendo la perturbazione come:

$$\begin{aligned} \delta(\vec{x}, t) &= \gamma(\vec{x}, t)[\hat{\alpha}(\vec{x}, t) - \alpha(\vec{x}, t) + (\hat{\beta}(\vec{x}, t) - \beta(\vec{x}, t))K\vec{\xi}] = \\ &= \Delta(\vec{\alpha}) + \Delta(\vec{\beta})K\vec{\xi} \end{aligned} \tag{2.46}$$

in cui $\Delta(\vec{\alpha}) = \gamma(\vec{x},t)[\hat{\alpha}(\vec{x},t) - \alpha(\vec{x},t)]$ e $\Delta(\vec{\beta}) = [\hat{\beta}(\vec{x},t) - \beta(\vec{x},t)]$, le equazioni (2.45) diventano:

$$\begin{cases} \dot{\vec{\lambda}} = f_0(\vec{\lambda}, \vec{\xi}) \\ \dot{\vec{\xi}} = K\vec{\xi} + \delta(\vec{x},t) \end{cases} \tag{2.47}$$

Si nota che il sistema dipende da dei parametri che forniscono informazioni sull'influenza delle incertezze sul modello. In particolare, se si verificano le condizioni:

$$\begin{aligned} \|\Delta(\vec{\alpha})\| &\leq \epsilon_1 \\ \left\|\Delta(\vec{\beta})\right\| &\leq \epsilon_2 \end{aligned} \tag{2.48}$$

dove ϵ_1 e ϵ_2 sono determinati in base al modello ed ai sensori di bordo dell'UAV, tramite uno studio dell'*equazione di Lyapunov* associata, perchè si ottenga una determinata regione asintoticamente stabile R_0 dovrà essere:

$$\|K\| \geq \frac{\epsilon_1}{R_0(1-\epsilon_2)} \tag{2.49}$$

Quindi, se le incertezze del modello sono alte e si vuole una regione di instabilità piccola, il guadagno K deve essere alto altrimenti, con piccole incertezze, la regione d'instabilità si mantiene limitata anche con valori piccoli di K, come si può vedere anche dalla figura 2.11 (nell'esempio, $\epsilon_1 = [0,10]$ e $\epsilon_2 = [0,1]$). In base a queste considerazioni, dunque, la legge di controllo si può definire abbastanza robusta anche in presenza di ambiguità, ad esempio sul valore della velocità del target (a questo proposito si vedano le simulazioni effettuate in 2.4.3).

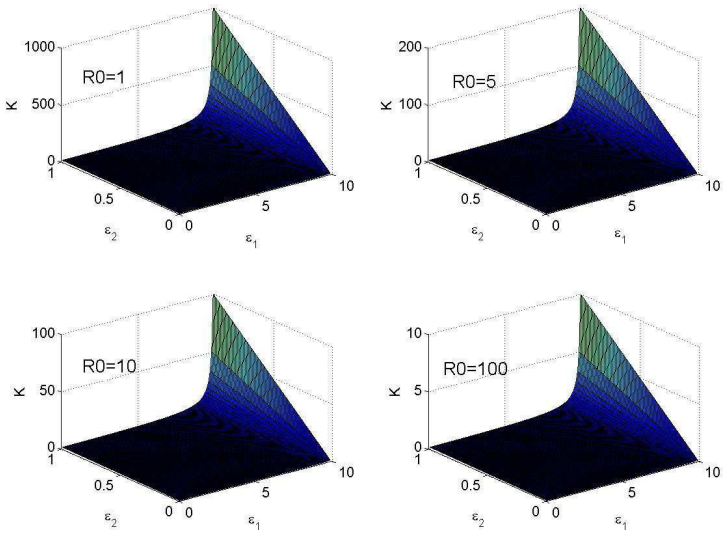


Figura 2.11: Guadagno K

2.4 Simulazioni MatLab-Simulink

Dopo aver chiarito dunque il problema ed essere arrivati ad una sua completa descrizione, sono state fatte delle simulazioni al riguardo per verificare il corretto comportamento della strategia risolutiva adottata. La piattaforma di simulazione utilizzata è *Simulink*, un software integrato con *MatLab* (*Matrix Laboratory*). Quest'ultimo è un ambiente per il calcolo numerico e l'analisi statistica che comprende anche l'omonimo linguaggio di programmazione creato da *MathWorks*. Matlab consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente e interfacciarsi con altri programmi. Simulink è un software integrato in Matlab ideato per modellare, simulare e analizzare sistemi dinamici di qualsiasi tipo (lineari e non lineari). La sua interfaccia grafica principale è uno strumento grafico di diagrammi a blocchi, raggruppati in librerie, risultando così molto intuitiva. Come primo passo, è stato dunque creato un file MatLab d'inizializzazione *File_ Ini.m* dove vengono definite tutte le condizioni iniziali del problema (si veda l'Appendice). Successivamente sono stati creati in Simulink i vari blocchi che permettono di descrivere il problema in tutti i suoi aspetti, sono stati opportunamente connessi tra di loro ed, infine, sono state eseguite varie simulazioni cambiando le condizioni iniziali per verificare che la risoluzione del problema sia effettivamente rigorosa e sempre valida. La strategia risolutiva utilizzata dal software è ovviamente numerica, in particolare attraverso il metodo di Runge-Kutta del 4°ordine. Nel nostro caso, si è considerato un tempo di simulazione pari a 100 s ad una frequenza di 100 Hz.

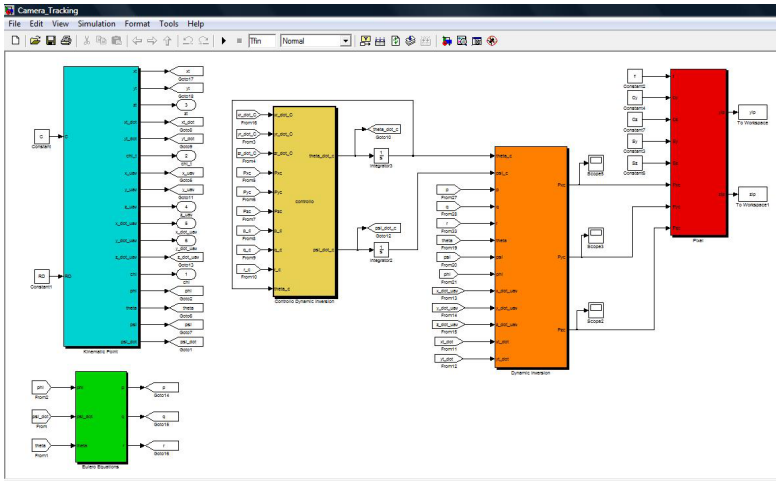


Figura 2.12: Schema Simulink

Le varie simulazioni sono state realizzate soprattutto cambiando la posizione e l'assetto dell'UAV e del target e considerando anche l'azione del vento. Si riportano di seguito alcune di queste.

2.4.1 Simulazioni con target fisso

Come prima simulazione effettuata, si è considerato il target fermo nell'origine delle coordinate mentre l'UAV partiva dalla posizione [100,100,-100] del sistema NED con una velocità pari a 15 m/s.

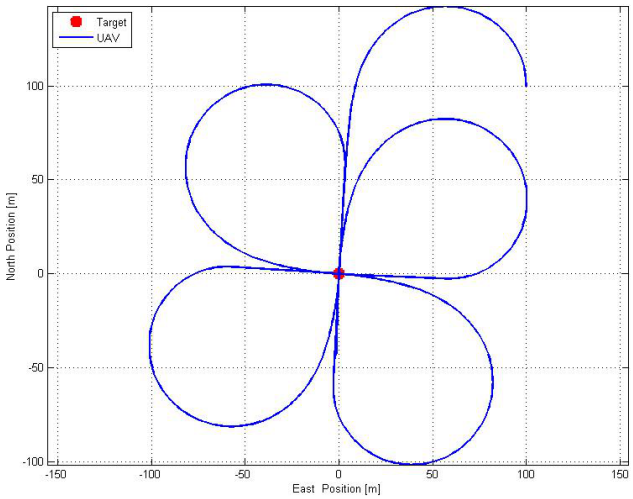


Figura 2.13: Traiettoria simulata con target fermo

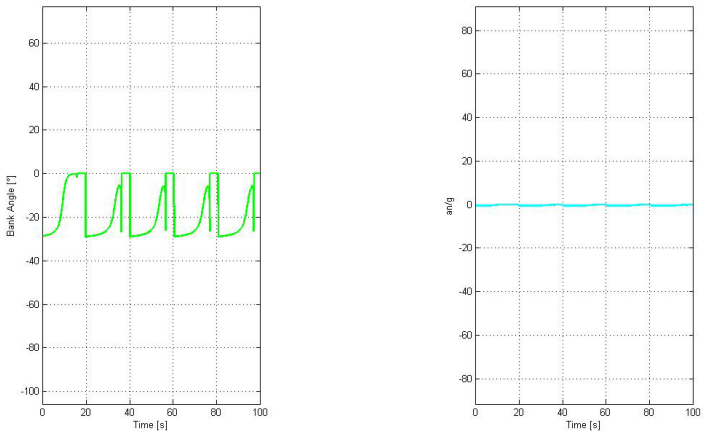
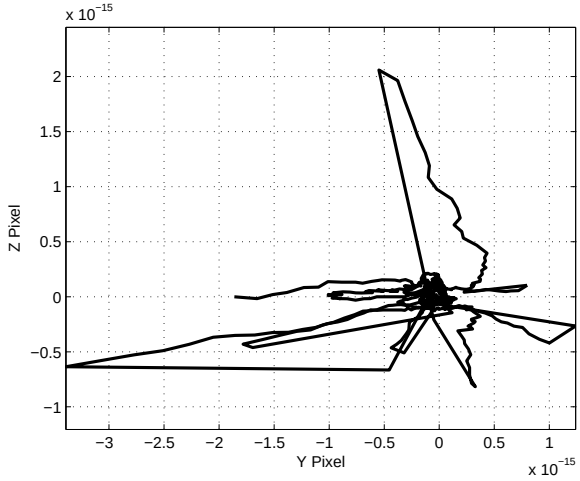
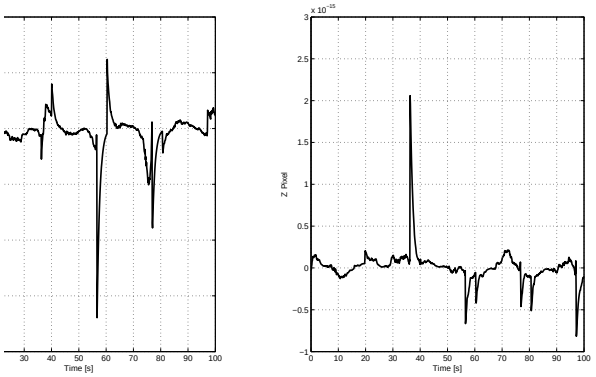


Figura 2.14: Andamento angolo di rollio e accelerazione adimensionalizzata

Come si può notare dalla figura 2.15, l'andamento dei pixels nello schermo testimonia il corretto funzionamento della legge di controllo in quanto effettivamente che questi si mantengono al centro del display. In figura 2.16, invece, si mostrano gli angoli di Pan e Tilt assunti dai servi nel tempo di simulazione.



(a) Pixels Y-Z



(b) Valori separati dei Pixels

Figura 2.15: Andamento Pixels

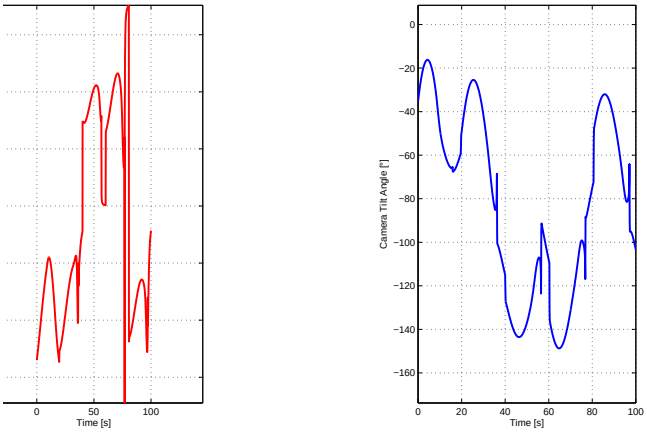


Figura 2.16: Angoli Pan-Tilt

In seguito, è stata effettuata sempre una simulazione con target fermo e con le stesse condizioni iniziali dell'UAV, ma con l'aggiunta di componenti di vento sia in direzione x che in direzione y per valutare il comportamento della legge di controllo anche in queste condizioni. Tuttavia, i risultati che si ottengono sono ancora una volta molto buoni, come si evince anche dalla figura 2.18.

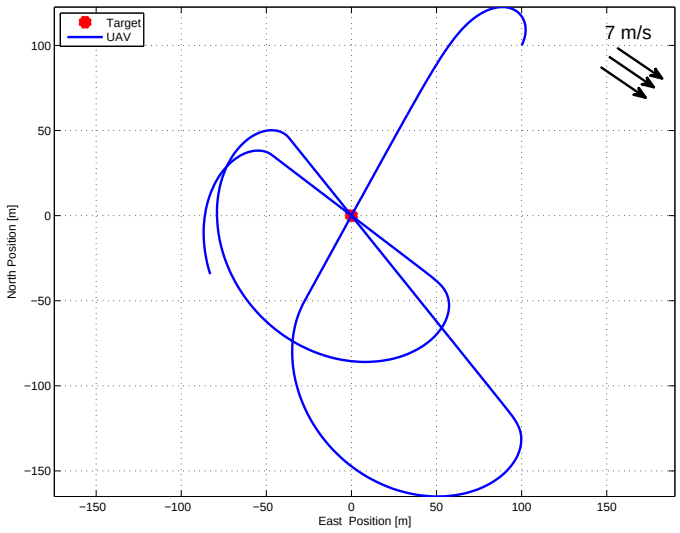
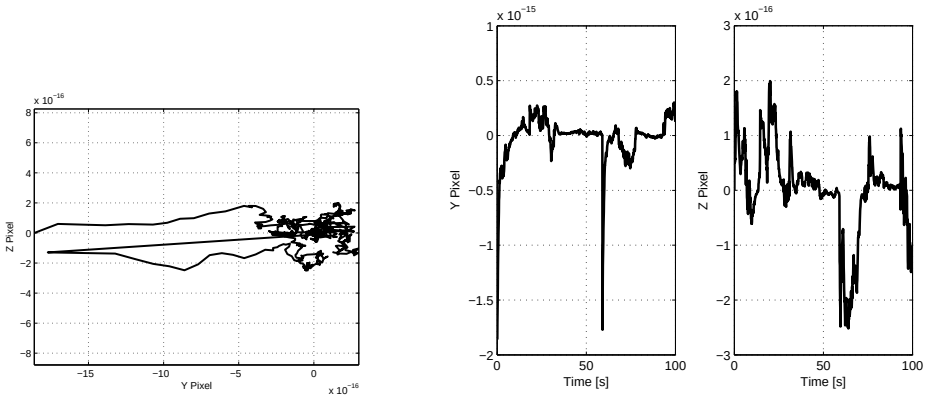


Figura 2.17: Traiettoria simulata con target fermo e componenti di vento



(a) Pixels Y-Z

(b) Valori separati dei Pixels

Figura 2.18: Andamento Pixels (con vento)

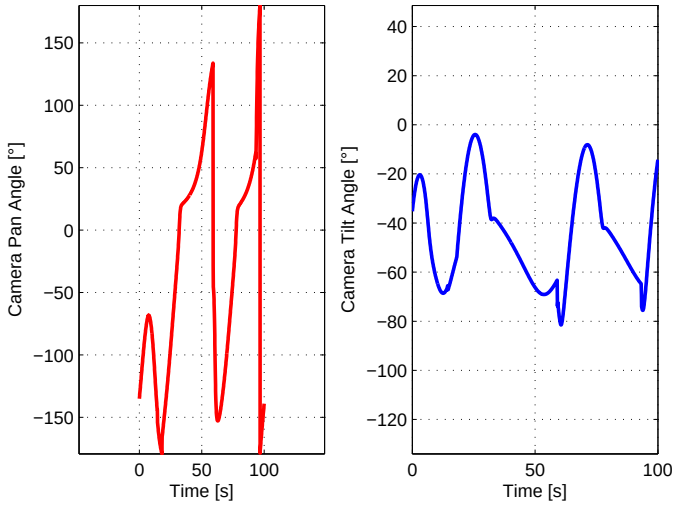


Figura 2.19: Angoli Pan-Tilt (con vento)

2.4.2 Simulazioni con target in movimento

Delle altre simulazioni sono state effettuate considerando un movimento rettilineo del target alla velocità di 10 m/s dalla posizione [-100,100,0] ed una posizione iniziale dell'UAV pari alle coordinate [100,0,-100] con una velocità di 15 m/s.

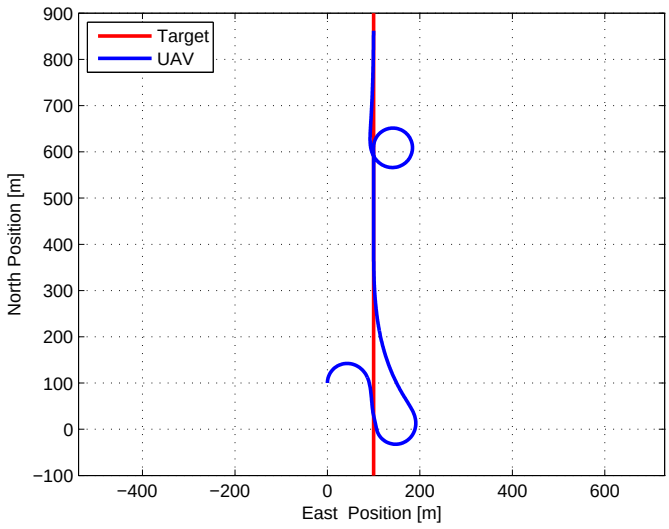


Figura 2.20: Traiettoria simulata con target in movimento

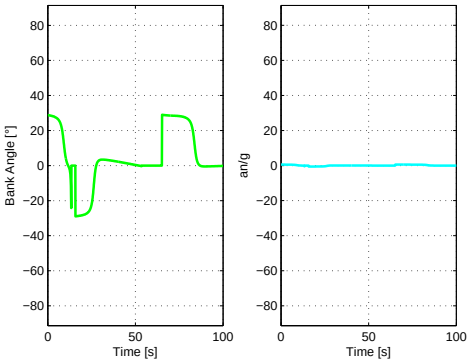
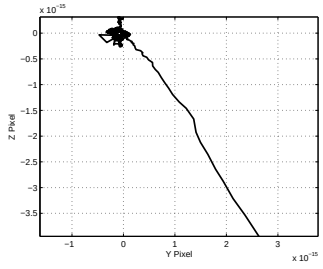
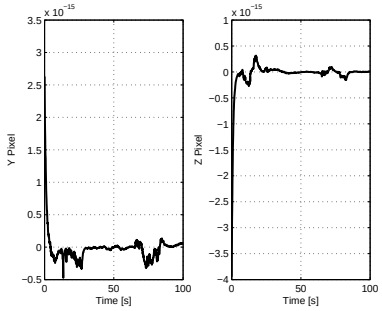


Figura 2.21: Andamento angolo di rollio e accelerazione adimensionalizzata con target in movimento

Anche quì i risultati che si ottengono sono più che accettabili a testimonianza del fatto che la legge di controllo progettata si comporta bene anche per differenti condizioni d'utilizzo, come si può vedere dalla figura 2.22.



(a) Pixels Y-Z



(b) Valori separati dei Pixels

Figura 2.22: Andamento Pixels con target in movimento

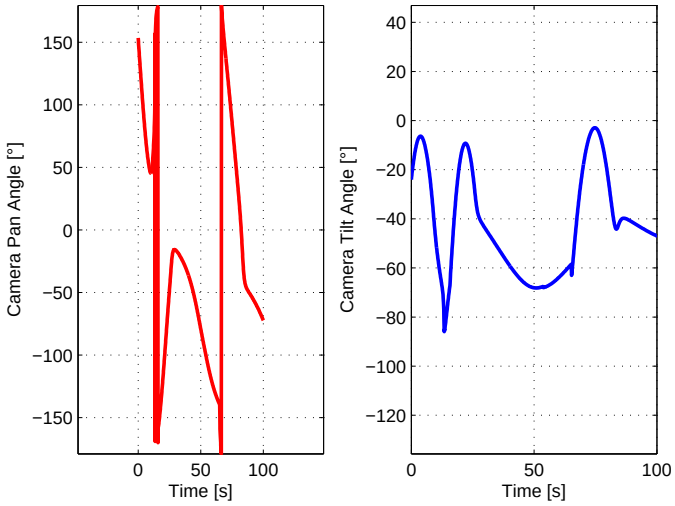


Figura 2.23: Angoli Pan-Tilt (target in movimento)

Successivamente, è stata effettuata una simulazione con target in movimento e presenza di vento, mantenendo le condizioni iniziali di posizione e velocità di target ed UAV precedenti ed inserendo un angolo di prua iniziale del target pari a $\chi_T = 45°$. I risultati ottenuti mostrano di nuovo un buon comportamento della legge di controllo, come si evince dalla figura 2.25.

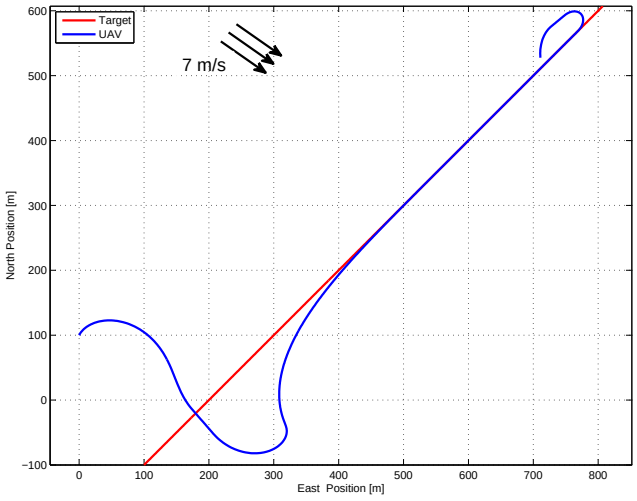
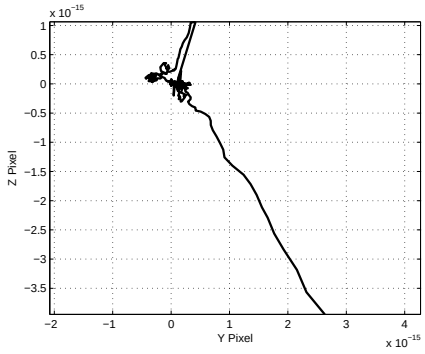
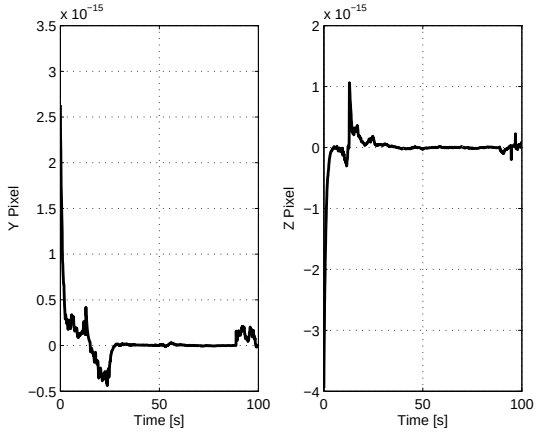


Figura 2.24: Traiettoria simulata con target in movimento e componenti di vento



(a) Pixels Y-Z



(b) Valori separati dei Pixels

Figura 2.25: Andamento Pixels (target in movimento e vento)

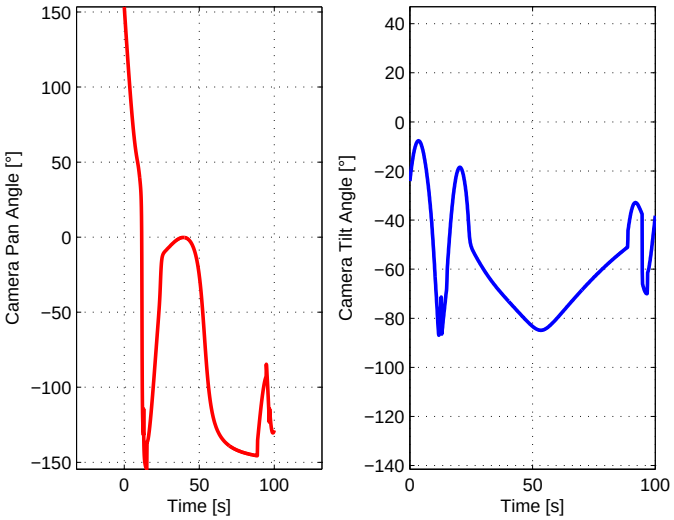


Figura 2.26: Angoli Pan-Tilt (target in movimento e vento)

2.4.3 Simulazioni senza informazioni sulla velocità del target

A verifica dello studio fatto sulla robustezza della legge di controllo, è stata effettuata una simulazione che tenesse conto di alcune ambiguità sullo stato del target, in particolare sulla sua velocità. Infatti, essa è stata considerata costante a 5 m/s per la descrizione della traiettoria, ma non è stata inserita come parametro nella legge di controllo quindi al controllore progettato non è stato fornito questo dato. Come altre condizioni al contorno, inoltre, sono state considerate una posizione iniziale del target nell'origine delle coordinate con una traiettoria variabile, una posizione iniziale dell'UAV nelle coordinate [100,100,-100] ed una velocità di 15 m/s. Inoltre sono stati inseriti degli errori intrinseci di misurazione sui pixels pari a $y_{ip} = z_{ip} = 20$ e si è considerato un tempo di simulazione pari a 30 s con una frequenza di 100 Hz. Nonostante ciò, come si può evincere dalla figura 2.28, il comportamento della legge di controllo è accettabile, dato che i pixels rimangono centrati nell'origine del display, e questo fornisce un'ulteriore prova della buona robustezza della legge di controllo sviluppata.

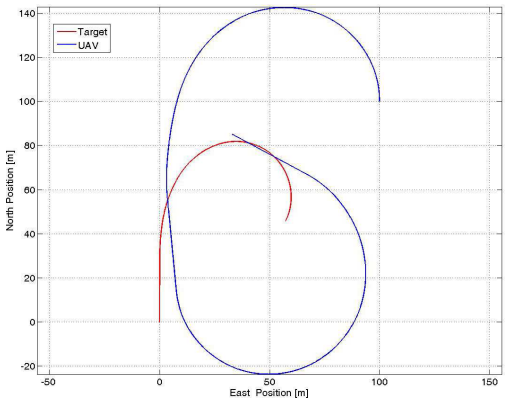
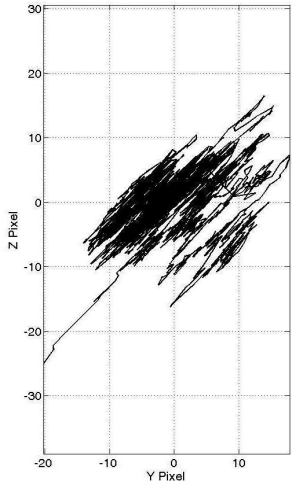
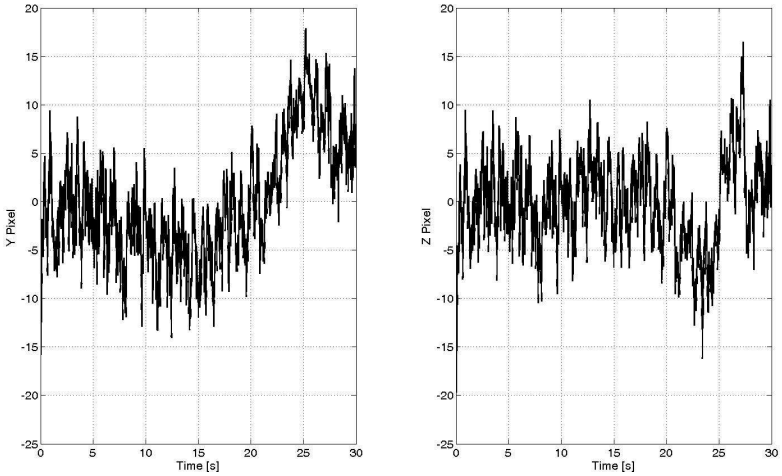


Figura 2.27: Traiettoria simulata senza informazioni sul target

La legge di controllo sviluppata è tutto sommato quindi di grande qualità, anche se non si dispongono di tutte le informazioni necessarie a descrivere il problema in ogni suo aspetto. Come ulteriore approfondimento perché il sistema risulti nella sua totalità più completo, si potrebbe pensare di sviluppare un filtro che stimi la posizione e la velocità del target quando queste informazioni non sono disponibili per il controllore, cioè il target non è collaborativo. Ciò potrebbe essere necessario sia perché velocità e posizione del target intervengono non solo nella legge di controllo, ma anche nella legge di guida del velivolo, sia perché i movimenti dei servi di Pan e Tilt sono, nella maggior parte dei casi, meccanicamente limitati a determinati angoli di saturazione per cui quando il target non è inquadrabile dalla telecamera, pur avendo una legge di controllo funzionante, si rende necessaria una stima di queste informazioni.



(a) Pixels Y-Z)



(b) Valori separati dei Pixels

Figura 2.28: Andamento Pixels (senza informazioni sul target)

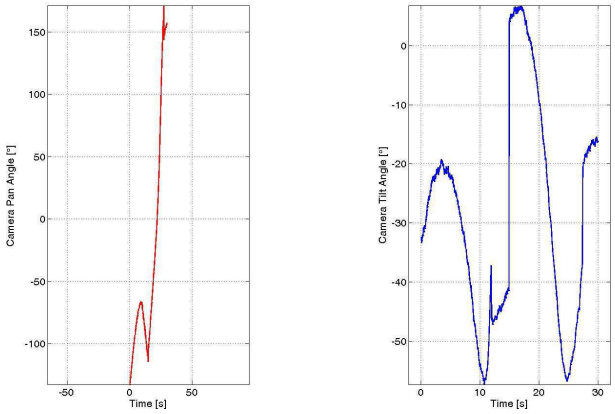


Figura 2.29: Angoli Pan-Tilt (senza informazioni sul target)

Capitolo 3

Implementazione su ArduPilot Mega

3.1 Presentazione ArduPilot Mega

Il sistema *ArduPilot Mega* è un autopilota *open-source IMU-based*, *Arduino* compatibile. Esso si può suddividere principalmente in 2 componenti hardware:

- **Main board:** si tratta del microcontrollore Atmel®ATMega2560 con frequenza massima di 16 MHz contenente processore e memorie *flash*, SRAM (*Static Random Access Memory*) e EEPROM (*Electrically Erasable Programmable Read-Only Memory*). Il componente può essere connesso in ingresso con il radiocomando e con i sensori accessori ed in uscita con i servocomandi che movimentano sia le superfici del velivolo che il sistema di Pan-Tilt della telecamera, tramite un totale di 8 canali. Il sistema inoltre è dotato di un chip secondario AT328 che, nel caso si verifichi un *crash* del codice sul processore principale, di fatto cortocircuita ingressi ed uscite dei quattro canali principali.

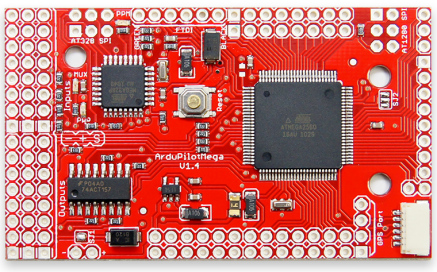


Figura 3.1: ArduPilot Mega Main Board con microcontrollore ATMega 2560

- **IMU shield:** questa scheda è collegata alla precedente main board tramite dei connettori *pin*. Essa serve principalmente a fornire al velivolo una serie di informazioni per la determinazione dell'assetto e della traiettoria tramite dei sensori quali accelerometri e giroscopi che formano la cosiddetta IMU (*Inertial Measurement Unit*). Per far sì che si ricostruisca in tempo reale il corretto orientamento del velivolo rispetto alla terna cartesiana di riferimento terrestre NED (*North-East-Down*), viene utilizzata una procedura chiamata *DCM (Direction Cosine Matrix) Estimator* in cui le tre componenti di velocità angolare fornite dal giroscopio vengono integrate per ottenere i tre angoli di assetto di *roll, pitch* e *yaw*, tenendo conto anche dell'informazione proveniente dal magnetometro montato sulla IMU shield stessa.

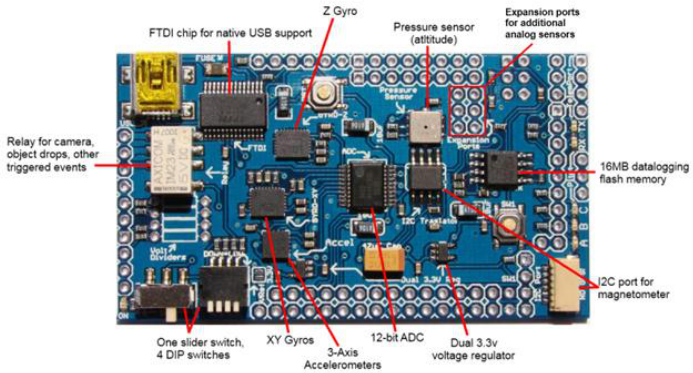


Figura 3.2: ArduPilot Mega IMU shield

Il sistema ArduPilot Mega impiega per la sua compilazione un ambiente di sviluppo denominato *Arduino IDE*, al cui interno è possibile trovare un terminale *CLI* (*Command Line Interface*). Il codice sviluppato viene *flashato* sopra il microcontrollore tramite l'interfaccia USB presente sulla IMU Shield. Il sistema di autopilota implementa sette tipi differenti di modalità (*flight-mode*):

1. *MANUAL*: controlli manuali tramite radiocomando;
2. *STABILIZE*: il sistema porta il velivolo in volo livellato con *roll* e *pitch* nulli;
3. *CIRCLE*: il sistema porta il velivolo con un angolo di rollio costante e mantenuto in virata;
4. *FLY BY WIRE A e B*: il sistema lascia la gestione del motore al pilota, mentre i comandi sui canali di alettoni ed equilibratore derivano da angoli di *roll* e *pitch* voluti;

5. *LOITER*: il sistema porta il velivolo in una modalità di traiettoria circolare sopra una determinata posizione geografica;

6. *RTL (Return To Launch)*: il sistema porta il velivolo sulle coordinate geografiche dove è stato acceso (home).

7. *AUTO*: il sistema porta il velivolo a seguire automaticamente una traiettoria determinata da vari *waypoints* preimpostati in fase di pianificazione di volo.

3.2 Sviluppo del codice

Come è stato già anticipato, dopo aver progettato la legge di controllo della telecamera in Simulink ed aver fatto delle simulazioni che testimoniano le ottime performance di questa, resta da implementare il controllo dei servi che movimentano il sistema Pan-Tilt della videocamera sul software di ArduPilot Mega, quindi mediante delle funzioni di *Arduino*. L'implementazione seguirà generalmente i passi descritti nella modellazione matematica, con qualche leggera modifica: in particolare, disponendo dei dati provenienti dai sensori connessi all'autopilota, la determinazione delle posizioni relative tra target ed UAV sarà molto più immediata, essendo necessarie soltanto delle trasformazioni tra sistemi di riferimento mediante le matrici di rotazione. Per quanto riguarda la legge di controllo, invece, essa sarà implementata semplicemente nei termini con i quali è stata precedentemente descritta e fornirà, mediante due canali d'uscita dell'autopilota, dei valori angolari che verranno trasformati elettricamente per comandare i servi che gestiscono i movimenti di Pan e di Tilt della telecamera. Si è agito, principalmente, aggiungendo a quello che è il listato del software di base di ArduPilot Mega delle funzioni significative (per tutte le funzioni di seguito riportate si veda l'Appendice).
Come primo passo, sono state fissate le posizioni a riposo del sistema Pan-Tilt, cioè quando il controllo non è attivo. In particolare, nel *setup()* di Arduino presente nel file *ArduPilotMega.pde* viene chiamata la funzione *init_ardupilot()* che gestisce l'inizializzazione generale dell'autopilota. In questa funzione, esplicitata nel file *system.pde*, vengono quindi assegnate le posizioni in cui deve portarsi il sistema Pan-Tilt quando il controllo è spento (nel nostro caso, si è preferito fissare un angolo di Pan pari 0° ed un angolo di Tilt pari a 60°, in modo da non rischiare il contatto a terra della telecamera in fase di decollo o atterraggio).

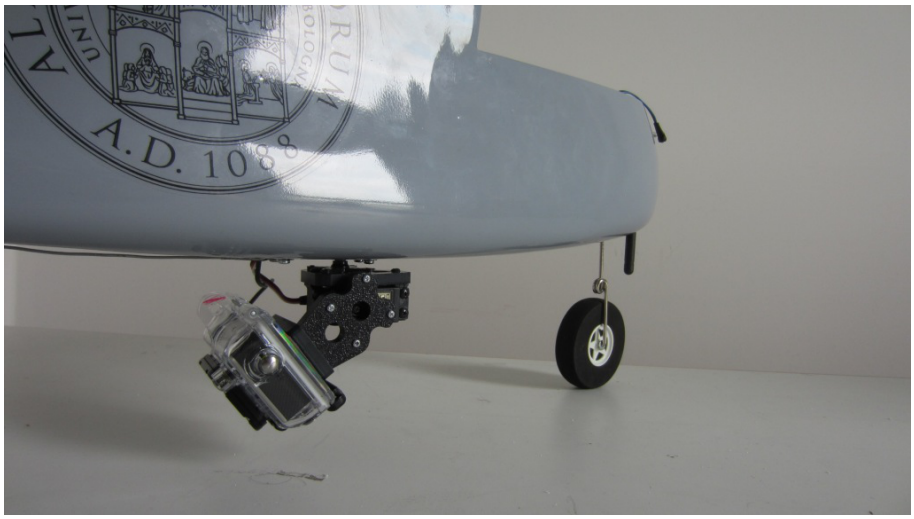


Figura 3.3: Configurazione telecamera decollo/atterraggio

Successivamente, è stata creata una funzione *camera_ loop()*, nel file *ArduPilot-Mega.pde*, richiamata all'interno del *fast_ loop()* di Arduino, che è sostanzialmente l'anello principale del programma ciclicamente ripetuto alla frequenza di circa 50 Hz con dati provenienti dai sensori sempre aggiornati. É importante sottolineare il fatto che il controllo è attivo solo nel caso in cui la differenza di quota tra UAV e *home* è superiore ai 30 metri perciò viene fatta una verifica tramite un *if* prima di chiamare effettivamente *camera_ loop()*. In caso contrario, il controllo risulta disattivato quindi il sistema Pan-Tilt si porta nella posizione iniziale ed è pronto ad un eventuale decollo o atterraggio senza rischiare che la telecamera tocchi terra. All'interno della funzione *camera_ loop()*, dunque, vengono a loro volta chiamate delle funzioni che gestiscono direttamente l'inizializzazione e il tracking della telecamera, rispettivamente *camera_ initialization()* e *camera_ track()* esplicitate nel file *Cam_ Movement.pde*. In particolare, la prima fornisce un'inizializzazione del procedimento di tracking in base alla posizione iniziale della telecamera: definiti, infatti, gli angoli iniziali di Pan e Tilt in cui essa si trova, vengono calcolate le posizioni relative tra target ed UAV nel sistema di riferimento Camera, trasformando le loro coordinate dal sistema WGS84 fornito dal sensore GPS mediante le rispettive matrici di rotazione. Per semplicità, è stato fissato come target un punto fisso a terra nella posizione di *home*. La funzione *camera_ track()*, invece, gestisce il vero e proprio tracking mediante telecamera. In essa viene chiamata un'ulteriore funzione *camera_ geometric()*, specificata anch'essa nel file *Cam_ Movement.pde*, che calcola le posizioni relative tra target ed UAV geometricamente in quanto le matrici di rotazione utilizzate nel cambio di sistemi di riferimento dipendono direttamente dagli angoli di Pan e Tilt dati dal controllo e ciò consente di avere le coordinate sempre

aggiornate. Successivamente, *camera_track()* richiama al suo interno la funzione *runge4()*, specificata nel file *runge_kutta.pde*, la quale non è altro che l'implementazione del metodo di Runge Kutta del 4° ordine. In questo caso, la tecnica di Runge Kutta viene utilizzata per l'integrazione numerica delle velocità angolari di Pan e Tilt della telecamera provenienti dal controllo. Infatti, *runge4()* chiama al suo interno le funzioni *controllo_Tilt()* e *controllo_Pan()*, specificate nel file *camera_control.pde*, che implementano la legge di controllo precedentemente descritta e forniscono in uscita l'andamento degli angoli di Pan e Tilt utili ad effettuare un corretto tracking. Una volta integrati questi valori, si ottengono gli angoli con i quali la telecamera deve ruotare: questi vengono direttamente passati ai servi del sistema Pan-Tilt. Tutta l'implementazione del programma richiede una memoria di 115204 bytes su un totale di 258048 bytes disponibili.
L'implementazione sul software dell'autopilota ArduPilot Mega è dunque di grande importanza perché, una volta inserito sulla main board tramite l'interfaccia USB, l'autopilota stesso ha un completamento molto significativo ed il velivolo può essere finalmente utilizzato operativamente per gli scopi percui è stato progettato.

3.3 Test HIL

La possibilità di effettuare un test HIL (*Hardware-in-the-Loop*) è uno dei grandi vantaggi che offre il sistema ArduPilot Mega. Sostanzialmente, si tratta di una simulazione software in grado di testare gli algoritmi programmati sul microcontrollore. L'architettura generale del processo è la seguente:

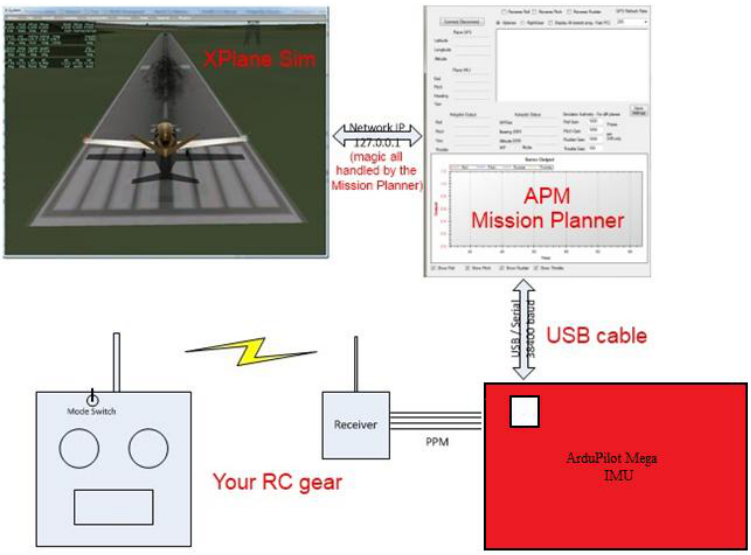


Figura 3.4: Architettura Test HIL

Il software di simulazione di volo usato è *X-Plane 9*, mentre la Ground Control Station di terra è l'*APM Mission Planner*. Utilizzando opportunamente i due software, si può pensare di verificare l'efficacia dei controlli progettati prima di effettuare il primo volo reale dell'UAV. Per il controllo della telecamera non è stato possibile effettuare un test HIL in quanto c'è bisogno di disporre di alcuni file sorgente che riguardano la comunicazione tra la Ground Control Station e il simulatore X-Plane 9. In particolare, tali file permettono, modificandoli opportunamente, di cambiare la visuale del simulatore di volo, predisponendo una determinata posizione della camera ed il suo movimento. Essendo dei file di proprietà della società creatrice del software di GCS, questi sono di difficile reperimento. Non si esclude, però, come un possibile sviluppo futuro, la possibilità di ottenerli e riuscire ad effettuare un test HIL anche della legge di controllo della telecamera progettata. D'altra parte, sono stati effettuati test HIL riguardanti il comportamento generale del velivolo con le altre funzioni dell'autopilota. Di seguito è spiegata in dettaglio la modalità con la quale questi test vengono effettuati.

3.3.1 APM Mission Planner

Grazie all'apparato di telemetria connesso sia alla board *ArduPilotMega IMU Shield* che al PC, è possibile utilizzare il software di GCS (*Ground Control Station*) che, in esecuzione su un PC a terra, permette di visualizzare in tempo reale le numerose informazioni e misure di volo fornite dall'autopilota, sia sottoforma di strumentazione virtuale ispirata a quella presente nei cockpit dei velivoli pilotati, sia come database in tempo reale; è inoltre possibile, attraverso la telemetria, interagire direttamente con il software di bordo per la modifica sul campo di alcuni parametri di funzionamento. La comunicazione tra telemetria del modulo *X-Bee* e Ground Control Station segue uno specifico protocollo denominato *MavLink*. La struttura di questo protocollo di comunicazione è presente nel programma che si carica sul microcontrollore presente su ArduPilotMega (libreria *Mavlink_ Common.h*). Una delle caratteristiche più usate è la nota *Point-and-Click Mission*. Una volta implementata la nostra missione, l'operatore alla Ground Control Station può cliccare sulla mappa il punto dove vuole mandare l'UAV. Per poter accedere a questa funzione, basta cliccare con il tasto destro del mouse sulla mappa e selezionare il comando *Fly to Here*. Il velivolo volerà fino al punto desiderato e s'inserirà in modalità *Loiter* fino a quando non gli sarà impartito un nuovo comando.

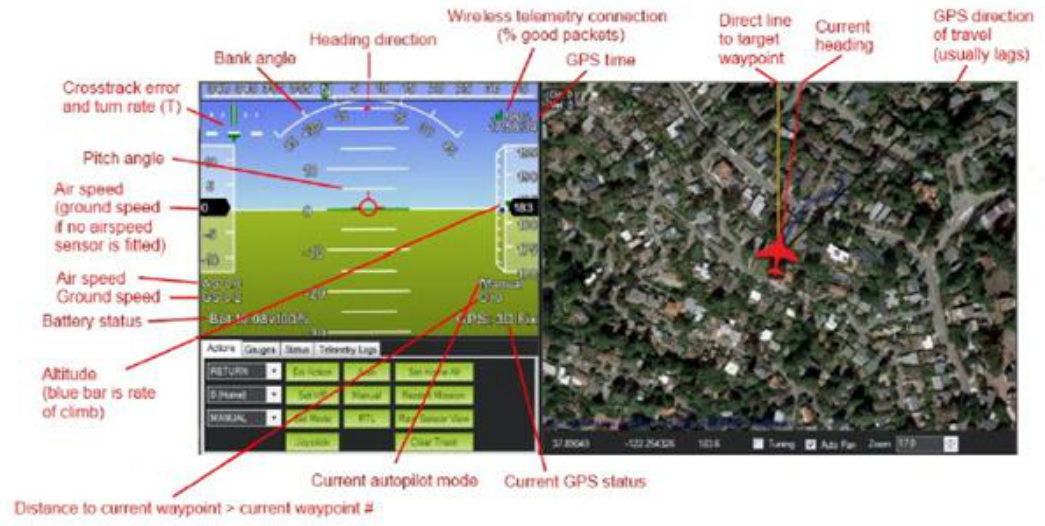


Figura 3.5: Schema principale GCS

E' possibile pianificare un piano di volo attraverso il Flight Planner, determinando la *home location* ed i *waypoints*, come mostrato in figura 3.6.

Figura 3.6: GCS Mission Planning

In figura 3.7 viene mostrato come è stata equipaggiata la stazione di terra. Essa è composta da un treppiedi sul quale è montata l'antenna direttiva che ruota e si direziona costantemente verso l'UAV. In aggiunta, si è installato un PC dove si comanderà il velivolo tramite l'*APM Mission Planner* ed uno schermo dove si visualizzeranno le immagini provenienti dalla telecamera.

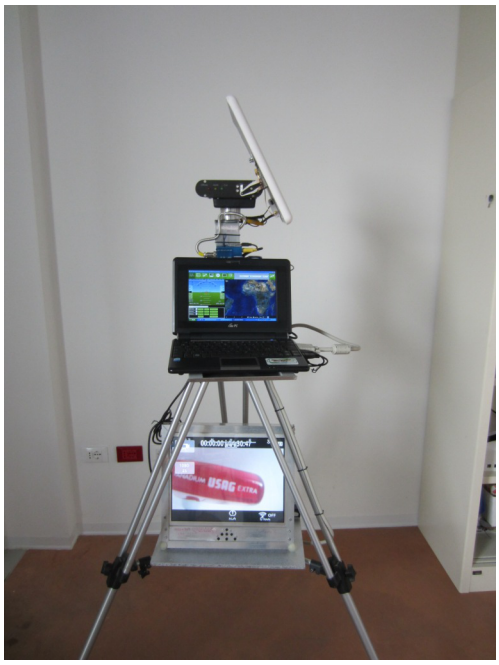


Figura 3.7: Stazione di terra

3.3.2 Interfacciamento ArduPilot Mega - GCS - X-Plane 9

Al fine di ottenere una corretta simulazione HIL è necessario capire come avviene la comunicazione tra i software utilizzati. Premesso che, come si è già accennato, l'interfacciamento tra l'APM Mission Planner e X-Plane 9 avviene tramite il protocollo seriale *MAVLink*[10], il simulatore di volo prevede come impostazione la possibilità di ricevere ed inviare una serie di dati *input/output* da e per un determinato utilizzatore. Una volta selezionati quali dati ricevere o inviare, s'instaura il corretto collegamento tra ArduPilot Mega e X-Plane 9 mediante la GCS e si può procedere ai test HIL, seguendo alcuni semplici passi:

1. Impostare X-Plane 9 al fine di comunicare con il Flight Planner mediante le *Net Connections* (figura 3.8) e le impostazioni del menù *Settings/Data Input Output* (figura 3.9);

2. Lanciare la GCS ed impostare e salvare un profilo di missione nel Flight Planner;

3. Collegare la presa USB assicurandosi che lo *slider-switch* (SW2) di ArduPilot-Mega IMU Shield sia verso il connettore dei servi (posizione 0);

4. Aprire il software dal programma *Arduino-022* e in *APM_ Config.h* impostare i parametri per effettuare una simulazione software:

[10] si rimanda a [13]

```
# define HIL_PROTOCOL          HIL_PROTOCOL_MAVLINK
# define HIL_MODE              HIL_MODE_ATTITUDE
# define HIL_PORT              0
# define SERIAL0_BAUD          115200
```

5. Eseguire l'upload del software;
6. Seguire la procedure per il settaggio dei comandi radio e dei modi impostati dall'autopilota;
7. Dopo aver chiuso il programma *Arduino-022*, aprire il Mission Planner, collegare la seriale a 115200 e, sempre dal Flight-Planner, caricare il profilo di missione salvato al passo 3;
8. Scollegare prima la seriale e successivamente staccare la presa USB. Muovere lo *slider switch* (SW2) di ArduPilotMega IMU Shield verso il *dip switch* (posizione 1) e ricollegare la presa USB;
9. Aprire nuovamente il Mission Planner e collegare nuovamente la seriale;
10. Lanciare X-Plane 9 scegliendo il modello di UAV e, nel Mission Planner, cliccare su *Simulation/Link SIM Avvia*.

In questo modo, il modello in X-Plane 9 verrà comandato attraverso i comandi dati con il radiocomando. E' possibile quindi, oltre che testare i comandi delle superfici mobili, osservare come il velivolo si comporta nelle modalità di autopilota scelte.

Figura 3.8: X-Plane 9 Net Connections

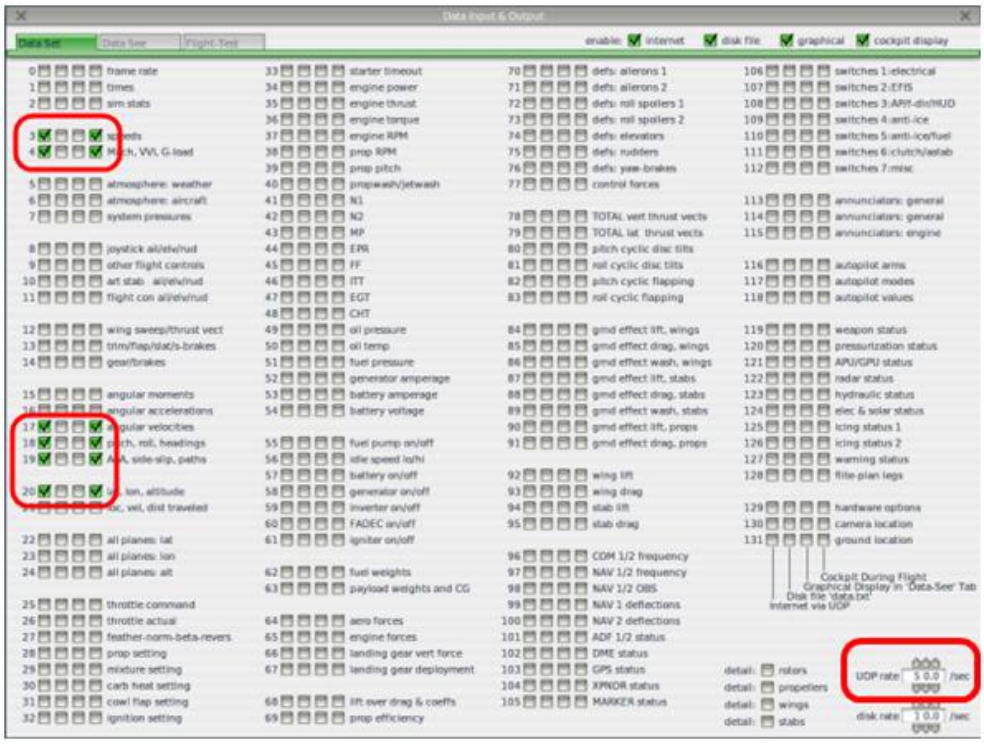


Figura 3.9: X-Plane 9 Data input/output

3.4 Applicazioni successive e miglioramenti

In questa sezione, si cerca di delineare quali possono essere i passi successivi volti al miglioramento del codice e alle basi da porre perché si possa pensare di effettuare un test HIL della legge di controllo sviluppata.
Il sistema studiato si configura in base ad una comunicazione bidirezionale esistente tra l'autopilota e la telecamera. Infatti, l'autopilota fornisce, tramite il controllo sviluppato, le velocità angolari di Pan e Tilt alla telecamera, in particolare ai suoi servi, con i quali essa può ruotare e direzionarsi verso il target d'interesse. Allo stesso tempo, la telecamera, tramite il movimento dei pixels nel display, manda le informazioni di posizione e velocità del target all'autopilota il quale le usa sia per la legge di guida che lo porta ad inseguire il target, sia per la legge di controllo della telecamera stessa. Quest'ultimo collegamento, però, necessita di un processo ausiliario di riconoscimento dell'immagine. In ogni caso, per testare il controllo, si può pensare di avere già delle informazioni sul target senza quindi disporre di questa importante integrazione complementare.
Innanzitutto, nello sviluppo del codice precedentemente descritto, si è pensato, per semplicità, di imporre un target fisso a terra nella posizione di *home*. Per testare la legge di controllo in modo di più generale, dunque, si potrebbe considerare il target

in movimento e, per far ciò, si potrebbe agire in due modi: il primo è ipotizzare di conoscere solo determinate coordinate di posizione considerandolo dunque come fisso a terra, ma variabile nei rispettivi punti; il secondo, invece, è conoscere la sua traiettoria reale mediante informazioni provenienti dalla GCS.
Per quanto riguarda la prima ipotesi, sarebbe conveniente utilizzare le coordinate dei *waypoints* come target variabili. Per far questo, il codice andrebbe modificato nella parte relativa alle funzioni *camera_ initialization()* e *camera_ geometric()*, presenti nel file *Cam_ Movement.pde*, imponendo il vettore di coordinate del target come **float** P_TARGET_GEO[3]={GPS_mark.lng,GPS_mark.lat,GPS_mark.alt}, cioè corrispondente ai waypoints scelti per la missione.
La seconda opzione, invece, sarebbe quella di modificare i file sorgente del software *APM Mission Planner* della GCS in modo da poter inserire all'interno di esso la descrizione della traiettoria del target e quindi riuscire a fornire questa informazione all'autopilota non più tramite una lettura dei pixels del display della telecamera, ma direttamente (figura 3.10).
Avendo, dunque, fatto queste considerazioni e modificate le rispettive parti dei file precedentemente descritti, si potrà procedere ad effettuare un test HIL che provi l'effettivo funzionamento della legge di controllo della telecamera. Una volta che il test darà buoni risultati, si potrà effettivamente procedere ad una verifica sperimentale del tracking con telecamera mediante un volo reale.

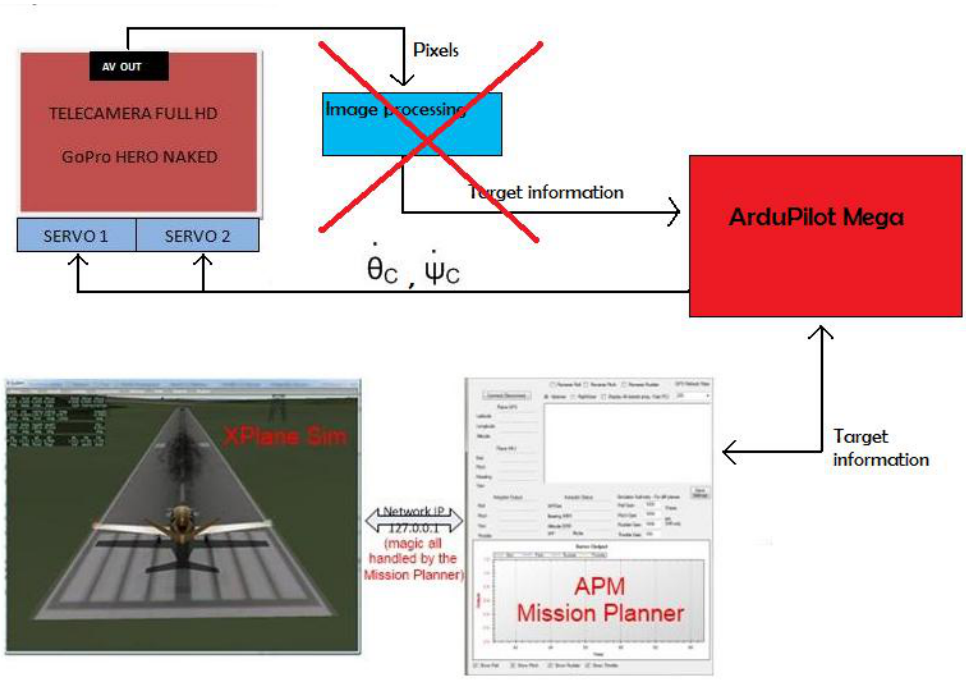


Figura 3.10: Test HIL Camera senza image processing

Conclusioni

Alla fine della presente tesi viene presentata una breve valutazione generale del lavoro svolto in relazione agli obiettivi che erano stati fissati per il progetto ed agli effettivi risultati raggiunti.
L'obiettivo principale del presente lavoro era la creazione di una valida ed efficace legge di controllo di una telecamera montata su un UAV ad ala fissa utile per applicazioni di target tracking, in particolare operazioni di monitoraggio ambientale, telerilevamento e sorveglianza. L'economicità e la versatilità di questi tipi velivoli costituiscono, infatti, i fattori chiave per la loro scelta e messa a punto. L'installazione a bordo dell'UAV di una videocamera si è resa dunque necessaria per la riuscita della missione ed ha costituito la base per lo sviluppo della legge di controllo della telecamera, che automaticamente inquadra l'obiettivo mantenendolo centrato nel display.
La legge di controllo si è basata su una tecnica di *input-output linearization* che, di fatto, linearizza un sistema di equazioni differenziali in grado di descrivere completamente il problema in oggetto e nelle quali si è esplicitata la dipendenza da parametri geometrici, aerodinamici ed inerziali, tenendo conto anche dei principali sistemi di riferimento adottati e delle caratteristiche tecniche specifiche della videocamera stessa. Nel progetto della legge di controllo, si è anche tenuto conto della possibile perdita d'informazioni o dell'influenza di errori di misurazione degli apparati per cui si è valutata la robustezza del sistema progettato, ottenendo comunque degli ottimi risultati anche se sono presenti delle ambiguità sui parametri in gioco.
Infatti, l'implementazione di questa legge di controllo in Simulink e la verifica del suo corretto comportamento, anche in particolari situazioni, mediante prove e simulazioni ha testimoniato la bontà del procedimento seguito e, di conseguenza, del controllore progettato. Le simulazioni effettuate hanno inoltre dimostrato che la legge di controllo non risente particolarmente dell'incertezza sulla velocità del target, fornendo un'ulteriore prova della robustezza del controllore.
Infine, il presente metodo è stato implementato sul software di bordo di ArduPilot Mega e costituisce un'importante integrazione del sistema di autopilota con il quale è stato equipaggiato il velivolo.

Possibili sviluppi futuri

Il lavoro effettuato nella presente tesi costituisce una parte integrante di un progetto più ampio per la messa a punto di un UAV in grado di adempiere correttamente agli scopi già citati. Sulla base di quanto discusso, è possibile individuare una serie di sviluppi futuri che permetterebbero sia una rivisitazione di alcune delle attività svolte finora che nuovi impieghi in altre attività di ricerca.
Un ampliamento della legge di controllo potrebbe essere costituito da un approccio mediante quaternioni invece che la descrizione in base agli angoli di Eulero. Essi forniscono una notazione matematica conveniente per la rappresentazione di orientamenti e rotazioni di oggetti in tre dimensioni. In confronto agli angoli di Eulero, i quaternioni presentano funzioni più semplici da comporre ed evitano il problema del *gimbal lock*, cioè dei problemi di singolarità dovuti a tali angoli. Confrontati con le matrici di rotazione, essi sono più stabili numericamente e quindi più efficienti perciò la legge di controllo avrebbe una validità più generale.
Lo sviluppo di un processo ausiliario di riconoscimento dell'immagine potrebbe costituire un'importante integrazione di tutto il sistema in quanto sarebbe molto più facile determinare alcune caratteristiche del target, ad esempio la sua posizione e velocità, in base ai pixels visualizzati nello schermo, soprattutto secondo la loro variazione nel tempo. La determinazione di queste grandezze potrebbe essere ulteriormente agevolata da un *filtro di Kalman* che costituirebbe un efficiente filtro in grado di valutare lo stato di un sistema dinamico a partire da una serie di misure soggette a rumore. Troverebbe utilizzo come osservatore dello stato del target e riuscirebbe a stimare la sua posizione e la sua velocità quando queste informazioni non sono disponibili, ad esempio quando il target si trova fuori dal campo visivo della telecamera dato che i movimenti dei servi di Pan e Tilt sono comunque meccanicamente limitati.
Un ulteriore sviluppo futuro in grado di ampliare il progetto studiato in questa tesi è la possibilità di effettuare un test HIL della legge di controllo proposta. Mediante la connessione tra software di simulazione di volo *X-Plane 9*, Ground Control Station di terra *APM Mission Planner* e *ArduPilot Mega*, si potrebbe pensare di verificare l'efficacia dei controlli progettati prima di effettuare il primo volo reale dell'UAV. Per far ciò, bisognerebbe reperire quei file sorgente con i quali è programmabile la Ground Control Station in modo da riuscire ad interfacciarla con il simulatore di volo, in particolare riuscendo a modificare la visuale di X-Plane in termini di posizione della camera e movimento e dunque verificando a terra il corretto comportamento della legge di controllo.
Resta, inoltre, la possibilità di sfruttare ancora di più il controllore di *ArduPilot Mega* ai fini dello studio di leggi di controllo e funzionalità più avanzate. Considerando

il basso costo del sistema, la sua praticità e programmabilità e il livello di sicurezza garantito dall'impiego di un aeromodello, si tratterebbe di una piattaforma ideale per questo tipo di sperimentazioni, vista la graduale crescita delle applicazioni di UAV e la loro attualità anche in ambito civile.

Appendice

Si riportano di seguito il file MatLab d'inizializzazione *File_ Ini.m* e le modifiche effettuate su varie parti del codice di *ArduPilot Mega*.

Listing 3.1: **File_Ini.m**

```
Tfin = 100;                    % [s]
step_size = 0.1;               % [s]
time = 0:step_size:Tfin;       % [s]

%% Target Initial Conditions
x0 = 0;       % [m]
y0 = 0;       % [m]
z0 = 0;       % [m]
chi_t0 = 0; % [rad]
VT = 0;       % [m/s]

%% UAV Initial Conditions
x_uav0 = 100;          % [m]
y_uav0 = 100;          % [m]
z_uav0 = -100;         % [m]
psi0 = 0;              % [rad]
theta0 = 0;            % [rad]
phi0 = 0;              % [rad]
V_uav = 15;            % [m/s]

%% Vento
Wx = -5;          % [m/s]
Wy = 5;           % [m/s]


```

```
%% Target Tracking Problem Initial Conditions
R0 = sqrt((x0-x_uav0)^2 + (y0-y_uav0)^2);     % [m]
sigma0 = atan2( (y0-y_uav0),(x0-x_uav0));     % [rad]

%% Gain Selection
RD = 57.8112;
C = 3.6057;
K2 = 5;

%% Dynamic Inversion Initial Conditions
psi_c0 =    0 ;              % [rad]
theta_c0 = 0;                % [rad]

R_B2C=
[cos(theta_c0)*cos(psi_c0)
 cos(theta_c0)*sin(psi_c0)
 -sin(theta_c0);
 -sin(psi_c0)
 cos(psi_c0)
 0;
 sin(theta_c0)*cos(psi_c0)
 sin(theta_c0)*sin(psi_c0)
 cos(theta_c0)];

R_I2B=
[cos(theta0)*cos(psi0)
 cos(theta0)*sin(psi0)
 -sin(theta0);
 sin(theta0)*sin(phi0)*cos(psi0)-cos(phi0)*sin(psi0)
 sin(theta0)*sin(phi0)*sin(psi0)+cos(phi0)*cos(psi0)
 sin(phi0)*cos(theta0);
 sin(theta0)*cos(phi0)*cos(psi0)+sin(phi0)*sin(psi0)
 sin(theta0)*cos(phi0)*sin(psi0)-sin(phi0)*cos(psi0)
 cos(phi0)*cos(theta0)];

y=R_I2B*[x0-x_uav0;   y0-y_uav0;  z0-z_uav0];
Z=R_B2C*y;
```

```
66 Pxc0=Z(1);
   Pyc0=Z(2);
68 Pzc0=Z(3);

70 psi_c0    =   atan2(Pyc0,Pxc0);
   theta_c0 =
72 -acos(sqrt(Pxc0^2+Pyc0^2)/(sqrt(Pxc0^2+Pyc0^2+Pzc0^2)));

74 R_B2C=
   [cos(theta_c0)*cos(psi_c0)
76  cos(theta_c0)*sin(psi_c0)
    -sin(theta_c0);
78  -sin(psi_c0)
    cos(psi_c0)
80  0;
    sin(theta_c0)*cos(psi_c0)
82  sin(theta_c0)*sin(psi_c0)
    cos(theta_c0)];
84

   Z=R_B2C*[Pxc0;Pyc0;Pzc0];
86

   Pxc0=Z(1);   % [m]
88 Pyc0=Z(2);   % [m]
   Pzc0=Z(3);   % [m]
90

   %% Pixels
92 Oy=0;              % offset
   Oz=0;              % offset
94 Sy=0.000264583; % [m] fattore di conversione pixel/metro
   Sz=0.000264583; % [m] fattore di conversione pixel/metro
96 f=0.006;          % [m] focale
```

Listing 3.2: **ArduPilotMega.pde**

```
1
    ..........
3   // Telecamera
    boolean flag_camera;
5   float y[5];
    float theta_c;
7   float psi_c;
    float Tilt;
9   float Pan;
    float theta_c_dot;
11  float psi_c_dot;

13  ..........

15  static void fast_loop()
    {
17      .....

19          if (current_loc.alt-home.alt > H_CAMERA)
                    camera_loop();
21          else
            {
23                  APM_RC.OutputCh(TILT_SERVO,1350);
                    APM_RC.OutputCh(PAN_SERVO,1400);
25          }
    }
27

    ..........
29

    static void camera_loop()
31  {
    if(flag_camera)
33          camera_track(y, G_Dt);
    else
35          camera_initialization();
    }
```

Listing 3.3: **APM_Config.h**

```
2  ..........

4  // INIZIALIZZAZIONE TELECAMERA

6  #define H_CAMERA          3000.0

8  #define THETA_C           -90.0
   #define PSI_C             -180.0
10
   #define TILT_SERVO   4
12 #define PAN_SERVO    5

14 #define PAN_RATIO    10.31
   #define TILT_RATIO   10.31
16

   ..........
```

Listing 3.4: **system.pde**

```
1  ..........

3  static void init_ardupilot()
   {
5            .....

7  APM_RC.OutputCh(TILT_SERVO,1350);
   APM_RC.OutputCh(PAN_SERVO,1400);
9  }

11 ..........
```

Listing 3.5: **Cam_Movement.pde**

```
1  void camera_initialization()
   {
3  flag_camera = true;

5  APM_RC.OutputCh(TILT_SERVO,2000);
```

```
7  GPS_mark          = get_wp_with_index(0);

9  cam_pitch         = ((float)(dcm.pitch_sensor))/ 100.0;
   cam_roll          = ((float)(dcm.roll_sensor))/ 100.0;
11 cam_yaw           = ((float)(dcm.yaw_sensor))/ 100.0;

13 if (cam_yaw < 0)
   cam_yaw += 360;
15
   cam_pitch = ToRad(cam_pitch);
17 cam_roll = ToRad(cam_roll);
   cam_yaw = ToRad(cam_yaw);
19
   float P_HOME_GEO[3]={home.lng/1.0e7,home.lat/1.0e7,home.alt
       /1.0e2};
21 float P_UAV_GEO[3]={current_loc.lng/1.0e7,current_loc.lat
       /1.0e7,current_loc.alt/1.0e2};

23 float P_TARGET_GEO[3];

25 for (int i=0;i<3;i++)
           P_TARGET_GEO[i]=P_HOME_GEO[i];
27


29 for (int i=0;i<2;i++)
   {
31         P_HOME_GEO[i]=ToRad(P_HOME_GEO[i]);
           P_TARGET_GEO[i]=ToRad(P_TARGET_GEO[i]);
33         P_UAV_GEO[i]=ToRad(P_UAV_GEO[i]);
   }
35
   float ecc = sqrt(1-sq(radius_of_earth_b)/sq(radius_of_earth)
       );
37
   float NFI_HOME = radius_of_earth/sqrt(1-sq(ecc)*sin(
      P_HOME_GEO[1])*sin(P_HOME_GEO[1]));
39
```

```
float X_HOME_ECEF = (NFI_HOME+P_HOME_GEO[2])*cos(P_HOME_GEO
    [0])*cos(P_HOME_GEO[1]);
float Y_HOME_ECEF = (NFI_HOME+P_HOME_GEO[2])*sin(P_HOME_GEO
    [0])*cos(P_HOME_GEO[1]);
float Z_HOME_ECEF = (NFI_HOME*(1-sq(ecc))+P_HOME_GEO[2])*sin
    (P_HOME_GEO[1]);

float NFI_UAV = radius_of_earth/sqrt(1-sq(ecc)*sin(P_UAV_GEO
    [1])*sin(P_UAV_GEO[1]));

float X_UAV_ECEF = (NFI_UAV+P_UAV_GEO[2])*cos(P_UAV_GEO[0])*
    cos(P_UAV_GEO[1]);
float Y_UAV_ECEF = (NFI_UAV+P_UAV_GEO[2])*sin(P_UAV_GEO[0])*
    cos(P_UAV_GEO[1]);
float Z_UAV_ECEF = (NFI_UAV*(1-sq(ecc))+P_UAV_GEO[2])*sin(
    P_UAV_GEO[1]);

float NFI_TARGET = radius_of_earth/sqrt(1-sq(ecc)*sin(
    P_TARGET_GEO[1])*sin(P_TARGET_GEO[1]));

float X_TARGET_ECEF = (NFI_TARGET+P_TARGET_GEO[2])*cos(
    P_TARGET_GEO[0])*cos(P_TARGET_GEO[1]);
float Y_TARGET_ECEF = (NFI_TARGET+P_TARGET_GEO[2])*sin(
    P_TARGET_GEO[0])*cos(P_TARGET_GEO[1]);
float Z_TARGET_ECEF = (NFI_TARGET*(1-sq(ecc))+P_TARGET_GEO
    [2])*sin(P_TARGET_GEO[1]);

float R_ECEF2NED[3][3] =
{{-sin(P_HOME_GEO[1])*cos(P_HOME_GEO[0]),    -sin(P_HOME_GEO
    [1])*sin(P_HOME_GEO[0]),        cos(P_HOME_GEO[1])},
{-sin(P_HOME_GEO[0]),                        cos(P_HOME_GEO
    [0]),                                        0},
{-cos(P_HOME_GEO[1])*cos(P_HOME_GEO[0]),    -cos(P_HOME_GEO
    [1])*sin(P_HOME_GEO[0]),       -sin(P_HOME_GEO[1])}};

float position_UAV_ECEF[3]={X_UAV_ECEF-X_HOME_ECEF,
    Y_UAV_ECEF-Y_HOME_ECEF,Z_UAV_ECEF-Z_HOME_ECEF};
float position_TARGET_ECEF[3]={X_TARGET_ECEF-X_HOME_ECEF,
    Y_TARGET_ECEF-Y_HOME_ECEF,Z_TARGET_ECEF-Z_HOME_ECEF};
```

```
63
   float P_UAV_NED[3];
65
   for(int k=0;k<3;k++)
67 {
          float a=0.0;
69        for(int i=0;i<3;i++)
                  a+=R_ECEF2NED[k][i]*position_UAV_ECEF[i];
71        P_UAV_NED[k]=a;
   }
73

75 float P_TARGET_NED[3];
   for(int k=0;k<3;k++)
77 {
          float a=0.0;
79        for(int i=0;i<3;i++)
                  a+=R_ECEF2NED[k][i]*position_TARGET_ECEF[i];
81        P_TARGET_NED[k]=a;
   }
83
   float rel_position[3] = {P_TARGET_NED[0]-P_UAV_NED[0],
      P_TARGET_NED[1]-P_UAV_NED[1],P_TARGET_NED[2]-P_UAV_NED
      [2]};
85
   float RI2B[3][3]=
87 {
   {cos(cam_pitch)*cos(cam_yaw),cos(cam_pitch)*sin(cam_yaw),-
      sin(cam_pitch)},
89 {sin(cam_pitch)*sin(cam_roll)*cos(cam_yaw)-cos(cam_roll)*sin
      (cam_yaw), sin(cam_pitch)*sin(cam_roll)*sin(cam_yaw)+cos(
      cam_roll)*cos(cam_yaw), sin(cam_roll)*cos(cam_pitch)},
   {sin(cam_pitch)*cos(cam_roll)*cos(cam_yaw)+sin(cam_roll)*sin
      (cam_yaw),sin(cam_pitch)*cos(cam_roll)*sin(cam_yaw)-sin(
      cam_roll)*cos(cam_yaw),cos(cam_roll)*cos(cam_pitch)}
91 };

93 float theta_c = THETA_C;
   float psi_c   = PSI_C;
```

```
95
   theta_c = ToRad(theta_c);
97 psi_c = ToRad(psi_c);

99 float RB2C[3][3]=
   {
101 {cos(theta_c)*cos(psi_c), cos(theta_c)*sin(psi_c), -sin(
       theta_c)},
   {-sin(psi_c), cos(psi_c), 0},
103 {sin(theta_c)*cos(psi_c),sin(theta_c)*sin(psi_c),cos(theta_c
       )}
   };
105
   float v[3];
107 float p[3];

109 for(int k=0;k<3;k++)
   {
111         float a=0.0;
            for(int i=0;i<3;i++)
113             a+=RI2B[k][i]*rel_position[i];
            v[k]=a;
115 }

117 for(int k=0;k<3;k++)
   {
119         float a=0.0;
            for(int i=0;i<3;i++)
121             a+=RB2C[k][i]*v[i];
            p[k]=a;
123 }

125 float Px = p[0];
   float Py = p[1];
127 float Pz = p[2];

129 theta_c =
   -acos(sqrt(sq(Px)+sq(Py))/sqrt(sq(Px)+sq(Py)+sq(Pz)));
131 psi_c    = atan2(Py,Px);
```

```
133 float RB2Cbis[3][3] =
    {
135 {cos(theta_c)*cos(psi_c), cos(theta_c)*sin(psi_c), -sin(
        theta_c)},
    {-sin(psi_c), cos(psi_c), 0},
137 {sin(theta_c)*cos(psi_c),sin(theta_c)*sin(psi_c),cos(theta_c
        )}
    };
139
    y[3] = theta_c;
141 y[4] = psi_c;

143 for(int k=0;k<3;k++)
    {
145         float a=0.0;
            for(int i=0;i<3;i++)
147                 a+=RB2Cbis[k][i]*p[i];
            y[k]=a;
149 };

151 camera_track(y, 0);
    }
153 }

155 void camera_geometric(float Tilt, float Pan)
    {
157
    // Angoli Eulero
159 cam_pitch       = ((float)(dcm.pitch_sensor))/ 100.0;
    cam_roll        = ((float)(dcm.roll_sensor))/ 100.0;
161 cam_yaw         = ((float)(dcm.yaw_sensor))/ 100.0;

163 if (cam_yaw < 0)
    cam_yaw += 360;
165
    cam_pitch = ToRad(cam_pitch);
167 cam_roll = ToRad(cam_roll);
    cam_yaw = ToRad(cam_yaw);
```

```
169
    float P_HOME_GEO[3]={home.lng/1.0e7,home.lat/1.0e7,home.alt
        /1.0e2};
171 float P_UAV_GEO[3]=
    {current_loc.lng/1.0e7,current_loc.lat/1.0e7,current_loc.alt
        /1.0e2};
173 float P_TARGET_GEO[3];

175 for (int i=0;i<3;i++)
            P_TARGET_GEO[i]=P_HOME_GEO[i];
177
    float ecc =
179 sqrt(1-sq(radius_of_earth_b)/sq(radius_of_earth));

181 float NFI_HOME =
    radius_of_earth/sqrt(1-sq(ecc)*sin(P_HOME_GEO[1])*sin(
        P_HOME_GEO[1]));
183
    float X_HOME_ECEF =
185 (NFI_HOME+P_HOME_GEO[2])*cos(P_HOME_GEO[0])*
    cos(P_HOME_GEO[1]);
187
    float Y_HOME_ECEF =
189 (NFI_HOME+P_HOME_GEO[2])*sin(P_HOME_GEO[0])*
    cos(P_HOME_GEO[1]);
191
    float Z_HOME_ECEF =
193 (NFI_HOME*(1-sq(ecc))+P_HOME_GEO[2])*
    sin(P_HOME_GEO[1]);
195
    float NFI_UAV =
197 radius_of_earth/sqrt(1-sq(ecc)*sin(P_UAV_GEO[1])*sin(
        P_UAV_GEO[1]));

199 float X_UAV_ECEF =
    (NFI_UAV+P_UAV_GEO[2])*cos(P_UAV_GEO[0])*
201 cos(P_UAV_GEO[1]);

203 float Y_UAV_ECEF =
```

```
(NFI_UAV+P_UAV_GEO[2])*sin(P_UAV_GEO[0])*
cos(P_UAV_GEO[1]);

float Z_UAV_ECEF =
(NFI_UAV*(1-sq(ecc))+P_UAV_GEO[2])*sin(P_UAV_GEO[1]);

float NFI_TARGET =
radius_of_earth/sqrt(1-sq(ecc)*sin(P_TARGET_GEO[1])*sin(
   P_TARGET_GEO[1]));

float X_TARGET_ECEF =
(NFI_TARGET+P_TARGET_GEO[2])*cos(P_TARGET_GEO[0])*
cos(P_TARGET_GEO[1]);

float Y_TARGET_ECEF =
(NFI_TARGET+P_TARGET_GEO[2])*sin(P_TARGET_GEO[0])*
cos(P_TARGET_GEO[1]);

float Z_TARGET_ECEF =
(NFI_TARGET*(1-sq(ecc))+P_TARGET_GEO[2])*
sin(P_TARGET_GEO[1]);

float R_ECEF2NED[3][3] =
{{-sin(P_HOME_GEO[1])*cos(P_HOME_GEO[0]),
  -sin(P_HOME_GEO[1])*sin(P_HOME_GEO[0]),
          cos(P_HOME_GEO[1])},
 {-sin(P_HOME_GEO[0]),
   cos(P_HOME_GEO[0]),
 0},
 {-cos(P_HOME_GEO[1])*cos(P_HOME_GEO[0]),
  -cos(P_HOME_GEO[1])*sin(P_HOME_GEO[0]),
  -sin(P_HOME_GEO[1])}};

float position_UAV_ECEF[3]=
{X_UAV_ECEF-X_HOME_ECEF,
Y_UAV_ECEF-Y_HOME_ECEF,
Z_UAV_ECEF-Z_HOME_ECEF};

float position_TARGET_ECEF[3]=
```

```
{X_TARGET_ECEF-X_HOME_ECEF,
Y_TARGET_ECEF-Y_HOME_ECEF,
Z_TARGET_ECEF-Z_HOME_ECEF};

float P_UAV_NED[3];
for(int k=0;k<3;k++)
{
        float a=0.0;
        for(int i=0;i<3;i++)
        a+=R_ECEF2NED[k][i]*position_UAV_ECEF[i];
        P_UAV_NED[k]=a;
}

float P_TARGET_NED[3];
for(int k=0;k<3;k++)
{
        float a=0.0;
        for(int i=0;i<3;i++)
        a+=R_ECEF2NED[k][i]*position_TARGET_ECEF[i];
        P_TARGET_NED[k]=a;
}

float rel_position[3] =
{P_TARGET_NED[0]-P_UAV_NED[0],
P_TARGET_NED[1]-P_UAV_NED[1],
P_TARGET_NED[2]-P_UAV_NED[2]};

float RI2B[3][3] =
{{cos(cam_pitch)*cos(cam_yaw),
cos(cam_pitch)*sin(cam_yaw),
-sin(cam_pitch)},
{sin(cam_pitch)*sin(cam_roll)*cos(cam_yaw)
-cos(cam_roll)*sin(cam_yaw),
sin(cam_pitch)*sin(cam_roll)*sin(cam_yaw)
+cos(cam_roll)*cos(cam_yaw),
sin(cam_roll)*cos(cam_pitch)},
{sin(cam_pitch)*cos(cam_roll)*cos(cam_yaw)
+sin(cam_roll)*sin(cam_yaw),
sin(cam_pitch)*cos(cam_roll)*sin(cam_yaw)
```

```
281 -sin(cam_roll)*cos(cam_yaw),
    cos(cam_roll)*cos(cam_pitch)}};
283
    float theta_c = Tilt;
285 float psi_c   = Pan;

287 float RB2C[3][3] =
    {{cos(theta_c)*cos(psi_c),
289 cos(theta_c)*sin(psi_c),
    -sin(theta_c)},
291 {-sin(psi_c),
    cos(psi_c),
293 0},
    {sin(theta_c)*cos(psi_c),
295 sin(theta_c)*sin(psi_c),
    cos(theta_c)}};
297
    float v[3];
299 float p[3];
    for(int k=0;k<3;k++)
301 {
        float a=0.0;
303     for(int i=0;i<3;i++)
        a+=RI2B[k][i]*rel_position[i];
305     v[k]=a;
    }
307
    for(int k=0;k<3;k++)
309 {
        float a=0.0;
311     for(int i=0;i<3;i++)
        a+=RB2C[k][i]*v[i];
313     p[k]=a;
    }
315
    float Px = p[0];
317 float Py = p[1];
    float Pz = p[2];
319
```

```
y[0] = Px;
y[1] = Py;
y[2] = Pz;
y[3] = theta_c;
y[4] = psi_c;
}

void camera_track(float y[], float timeStep)
{

camera_geometric(y[3],y[4]);

runge4(y, timeStep);

Tilt = y[3];

Pan = y[4];

Tilt = (int)((wrap_180(ToDeg(Tilt)*100))/100);
Pan = (int)((wrap_180(ToDeg(Pan)*100))/100);

if (Pan>90)
{
Pan = Pan-180;
Tilt = 180-Tilt;
}

if (Pan<(-90))
{
Pan = Pan+180;
Tilt = 180-Tilt;
}

APM_RC.OutputCh(TILT_SERVO,2000+Tilt*TILT_RATIO);          //
    send to Servos
APM_RC.OutputCh(PAN_SERVO, 1400+Pan*PAN_RATIO);            //
    send to Servos
}
```

Listing 3.6: **camera_control.pde**

```
float controllo_Tilt(float y[])
{
theta_c=y[3];
psi_c= y[4];

Vector3f gyro = imu.get_gyro();

float p=gyro.x;
float q=gyro.y;
float r=gyro.z;

cam_pitch       = ((float)(dcm.pitch_sensor))/ 100.0;
cam_roll        = ((float)(dcm.roll_sensor))/ 100.0;
cam_yaw         = ((float)(dcm.yaw_sensor))/ 100.0;

if (cam_yaw < 0)
cam_yaw += 360;

cam_pitch = ToRad(cam_pitch);
cam_roll = ToRad(cam_roll);
cam_yaw = ToRad(cam_yaw);

float Vr_I[3];
float V_G = g_gps->ground_speed / 100.0;
float Ground_course = g_gps->ground_course / 100.0;

Vr_I[0]=V_G*cos(Ground_course)*cos(cam_pitch);
Vr_I[1]=V_G*sin(Ground_course)*cos(cam_pitch);
Vr_I[2]=V_G*sin(cam_pitch);

float RI2B[3][3] =
{{cos(cam_pitch)*cos(cam_yaw),
cos(cam_pitch)*sin(cam_yaw),
-sin(cam_pitch)},
{sin(cam_pitch)*sin(cam_roll)*cos(cam_yaw)
-cos(cam_roll)*sin(cam_yaw),
sin(cam_pitch)*sin(cam_roll)*sin(cam_yaw)
```

```
+cos(cam_roll)*cos(cam_yaw),
sin(cam_roll)*cos(cam_pitch)},
{sin(cam_pitch)*cos(cam_roll)*cos(cam_yaw)
+sin(cam_roll)*sin(cam_yaw),
sin(cam_pitch)*cos(cam_roll)*sin(cam_yaw)
-sin(cam_roll)*cos(cam_yaw),
cos(cam_roll)*cos(cam_pitch)}};

float RB2C[3][3] =
{{cos(theta_c)*cos(psi_c),
cos(theta_c)*sin(psi_c),
-sin(theta_c)},
{-sin(psi_c),
cos(psi_c),
0},
{sin(theta_c)*cos(psi_c),
sin(theta_c)*sin(psi_c),
cos(theta_c)}};

float Vr_B[3];
float Vr_C[3];

for(int k=0;k<3;k++)
{
float a=0.0;
for(int h=0;h<3;h++)
a+=RI2B[k][h]*Vr_I[h];
Vr_B[k]=a;
}

for(int k=0;k<3;k++)
{
float a=0.0;
for(int h=0;h<3;h++)
a+=RB2C[k][h]*Vr_B[h];
Vr_C[k]=a;
}

float x_r_C_dot = Vr_C[0];
```

```
77 float y_r_C_dot = Vr_C[1];
   float z_r_C_dot = Vr_C[2];
79
   float p_c =
81 cos(theta_c)*cos(psi_c)*p
   +cos(theta_c)*sin(psi_c)*r
83 -sin(theta_c)*q;
   float q_c =
85 -sin(psi_c)*p
   +cos(psi_c)*q;
87 float r_c =
   sin(theta_c)*cos(psi_c)*p
89 +sin(theta_c)*sin(psi_c)*q
   +cos(theta_c)*r;
91
   float Sy=0.000264583;
93 float Sz=0.000264583;
   float f=0.006;
95
   float K_GAIN[2]={-10*y[1],-10*y[2]};
97
   float F[2]={(f/y[0])*((1/Sy)*(y_r_C_dot-y[0]*r_c+y[2]*p_c)
99 -(y[1]/(Sy*y[0]))*(x_r_C_dot-y[2]*q_c+y[1]*r_c)),(f/y[0])*
   ((1/Sz)*(z_r_C_dot-y[1]*p_c+y[0]*q_c)-(y[2]/(Sz*y[0]))*
101 (x_r_C_dot-y[2]*q_c+y[1]*r_c))};

103 float G_INV[2][2]={{-(y[0]*y[1]*Sy*(y[2]*cos(theta_c)
   -y[0]*sin(theta_c)))/(f*(y[0]*cos(theta_c)
105 +y[2]*sin(theta_c))*(sq(y[0]) + sq(y[1]) +
   sq(y[2]))), (y[0]*Sz*(cos(theta_c)*sq(y[0])
107 + y[2]*sin(theta_c)*y[0] + cos(theta_c)*sq(y[1])))
   /(f*(y[0]*cos(theta_c) + y[2]*sin(theta_c))*(sq(y[0])
109 + sq(y[1]) + sq(y[2])))},
   {-(y[0]*Sy*(sq(y[0])+sq(y[2])))/(f*(y[0]*cos(theta_c)
111 + y[2]*sin(theta_c))*(sq(y[0]) + sq(y[1]) + sq(y[2]))),
   (y[0]*y[1]*y[2]*Sz)/(f*(y[0]*cos(theta_c) + y[2]*
113 sin(theta_c))*(sq(y[0]) + sq(y[1]) + sq(y[2])))}};

115 float M[2][2];
```

```
117 for (int k=0;k<2;k++)
    for(int j=0;j<2;j++)
119 M[k][j]=G_INV[k][j]*(-1);

121 float alpha_x[2];
    float beta_x[2];
123 float out[2];

125 for(int k=0;k<2;k++)
    {
127 float a=0.0;
    for(int i=0;i<2;i++)
129 a+=M[k][i]*F[i];
    alpha_x[k]=a;
131 };

133 for(int k=0;k<2;k++)
    {
135 float a=0.0;
    for(int i=0;i<2;i++)
137 a+=G_INV[k][i]*K_GAIN[i];
    beta_x[k]=a;
139 };

141 for (int k=0;k<2;k++)
    out[k]=alpha_x[k]+beta_x[k];
143

    theta_c_dot=(int)(out[0]*1.0e4);
145 theta_c_dot=(float)(theta_c_dot/1.0e4);
    return theta_c_dot;
147 }

149 float controllo_Pan(float y[])
    {
151 theta_c=y[3];
    psi_c= y[4];
153

    Vector3f gyro = imu.get_gyro();
```

```
155
    float p=gyro.x;
157 float q=gyro.y;
    float r=gyro.z;
159
    cam_pitch       = ((float)(dcm.pitch_sensor))/ 100.0;
161 cam_roll        = ((float)(dcm.roll_sensor))/ 100.0;
    cam_yaw         = ((float)(dcm.yaw_sensor))/ 100.0;
163
    if (cam_yaw < 0)
165 cam_yaw += 360;

167 cam_pitch = ToRad(cam_pitch);
    cam_roll = ToRad(cam_roll);
169 cam_yaw = ToRad(cam_yaw);

171 float Vr_I[3];
    float V_G = g_gps->ground_speed / 100.0;
173 float Ground_course = g_gps->ground_course / 100.0;

175 Vr_I[0]=V_G*cos(Ground_course)*cos(cam_pitch);
    Vr_I[1]=V_G*sin(Ground_course)*cos(cam_pitch);
177 Vr_I[2]=V_G*sin(cam_pitch);

179 float RI2B[3][3] =
    {{cos(cam_pitch)*cos(cam_yaw),
181 cos(cam_pitch)*sin(cam_yaw),
    -sin(cam_pitch)},
183 {sin(cam_pitch)*sin(cam_roll)*cos(cam_yaw)
    -cos(cam_roll)*sin(cam_yaw),
185 sin(cam_pitch)*sin(cam_roll)*sin(cam_yaw)
    +cos(cam_roll)*cos(cam_yaw),
187 sin(cam_roll)*cos(cam_pitch)},
    {sin(cam_pitch)*cos(cam_roll)*cos(cam_yaw)
189 +sin(cam_roll)*sin(cam_yaw),
    sin(cam_pitch)*cos(cam_roll)*sin(cam_yaw)
191 -sin(cam_roll)*cos(cam_yaw),
    cos(cam_roll)*cos(cam_pitch)}};
193
```

```
float RB2C[3][3] =
{{cos(theta_c)*cos(psi_c),
cos(theta_c)*sin(psi_c),
-sin(theta_c)},
{-sin(psi_c),
cos(psi_c),
0},
{sin(theta_c)*cos(psi_c),
sin(theta_c)*sin(psi_c),
cos(theta_c)}};

float Vr_B[3];
float Vr_C[3];

for(int k=0;k<3;k++)
{
float a=0.0;
for(int h=0;h<3;h++)
a+=RI2B[k][h]*Vr_I[h];
Vr_B[k]=a;
}

for(int k=0;k<3;k++)
{
float a=0.0;
for(int h=0;h<3;h++)
a+=RB2C[k][h]*Vr_B[h];
Vr_C[k]=a;
}

float x_r_C_dot = Vr_C[0];
float y_r_C_dot = Vr_C[1];
float z_r_C_dot = Vr_C[2];

float p_c =
cos(theta_c)*cos(psi_c)*p
+cos(theta_c)*sin(psi_c)*r
-sin(theta_c)*q;
float q_c =
```

```
-sin(psi_c)*p
+cos(psi_c)*q;
float r_c =
sin(theta_c)*cos(psi_c)*p
+sin(theta_c)*sin(psi_c)*q
+cos(theta_c)*r;

float Sy=0.000264583;
float Sz=0.000264583;
float f=0.006;

float K_GAIN[2]={-10*y[1],-10*y[2]};

float F[2]={(f/y[0])*((1/Sy)*(y_r_C_dot-y[0]*r_c+y[2]*p_c)
-(y[1]/(Sy*y[0]))*(x_r_C_dot-y[2]*q_c+y[1]*r_c)),(f/y[0])*
((1/Sz)*(z_r_C_dot-y[1]*p_c+y[0]*q_c)-(y[2]/(Sz*y[0]))*
(x_r_C_dot-y[2]*q_c+y[1]*r_c))};


float G_INV[2][2]={{-(y[0]*y[1]*Sy*(y[2]*cos(theta_c)
-y[0]*sin(theta_c)))/(f*(y[0]*cos(theta_c)
+y[2]*sin(theta_c))*(sq(y[0]) + sq(y[1]) +
sq(y[2]))), (y[0]*Sz*(cos(theta_c)*sq(y[0])
+ y[2]*sin(theta_c)*y[0] + cos(theta_c)*sq(y[1])))
/(f*(y[0]*cos(theta_c) + y[2]*sin(theta_c))*(sq(y[0])
+ sq(y[1]) + sq(y[2])))},
{-(y[0]*Sy*(sq(y[0])+sq(y[2])))/(f*(y[0]*cos(theta_c)
+ y[2]*sin(theta_c))*(sq(y[0]) + sq(y[1]) + sq(y[2]))),
(y[0]*y[1]*y[2]*Sz)/(f*(y[0]*cos(theta_c) + y[2]*
sin(theta_c))*(sq(y[0]) + sq(y[1]) + sq(y[2])))}};

float M[2][2];

for (int k=0;k<2;k++)
for(int j=0;j<2;j++)
M[k][j]=G_INV[k][j]*(-1);

float alpha_x[2];
float beta_x[2];
```

```
float out[2];

for(int k=0;k<2;k++)
{
float a=0.0;
for(int i=0;i<2;i++)
a+=M[k][i]*F[i];
alpha_x[k]=a;
};

for(int k=0;k<2;k++)
{
float a=0.0;
for(int i=0;i<2;i++)
a+=G_INV[k][i]*K_GAIN[i];
beta_x[k]=a;
};

for (int k=0;k<2;k++)
out[k]=alpha_x[k]+beta_x[k];

psi_c_dot=(int)(out[1]*1.0e4);
psi_c_dot=(float)(psi_c/1.0e4);
return psi_c_dot;
}
```

Listing 3.7: **defines.h**

```
.........

#define CH_TILT      CH_5
#define CH_PAN       CH_6

.....

// Runge Kutta
#define N      5     /* number of first order equations */
#define dist   0.01     /* stepsize in t*/
#define MAX    0.01
```

Listing 3.8: **runge_kutta.pde**

```
void runge4(float y[], float timeStep)
{
float h=timeStep/2.0,

t1[N], t2[N], t3[N],
k1[N], k2[N], k3[N],k4[N];

int i;

for (i=0;i<N;i++)
t1[i]=y[i]+0.5*(k1[i]=timeStep*f(y, i, timeStep));

for (i=0;i<N;i++)
t2[i]=y[i]+0.5*(k2[i]=timeStep*f(t1, i, timeStep));

for (i=0;i<N;i++)
t3[i]=y[i]+ (k3[i]=timeStep*f(t2, i, timeStep));

for (i=0;i<N;i++)
k4[i]= timeStep*f(t3, i, y_prec, timeStep);

for (i=0;i<N;i++) y[i]+=(k1[i]+2*k2[i]+2*k3[i]+k4[i])/6.0;

}

float f(float y[], int i, float timeStep)
{
switch (i)
{
case 0:
break;
case 1:
break;
case 2:
break;
case 3:
return(controllo_Tilt((float*)y));
```

```
    break;
39  case 4:
    return(controllo_Pan((float*)y));
41  break;
    }
43  }
```

Bibliografia

[1] Matteo Zanzi, Antonio Ghetti, Niki Regina, *Manuale Zanzi Versione 3.1*, Università di Bologna.

[2] Yiying Li, Dylan Davis, Ella M. Atkins, *A Vision-based Tracking System for a Permanent Flight Display with Natural Lighting*, University of Michigan, AIAA Guidance, Navigation, and Control Conference 2012, Minneapolis, USA.

[3] Holly P. Borowsky, Eric W. Frew, *An Evaluation of Path Planners for Guidance with Vision Based Simultaneous Localization and Mapping*, University of Colorado, AIAA Guidance, Navigation, and Control Conference 2012, Minneapolis, USA.

[4] Justin M. Selfridge, *An On-Board Camera Based Localization Solution*, Old Dominon University, AIAA Guidance, Navigation, and Control Conference 2012, Minneapolis, USA.

[5] D. Blake Barber, Joshua D. Redding, Timothy W. McLain, Randal W. Beard, Clark N. Taylor, *Vision-based Target Geo-location using a Fixed-wing Miniature Air Vehicle*, Bringham Young University.

[6] Subodh Bhandari, Amar Raheja, Matthew Rose, Hoving Yaralian, Manuel Segura, Mike DŠZmura, *Tracking of Mobile Targets using Unmanned Aerial Vehicles*, University of California, AIAA Guidance, Navigation, and Control Conference 2012, Minneapolis, USA.

[7] Joseph Nichols, Jeff Ferrin, Mark Owen, Timothy McLain, *Vision-Enhanced Aerial Rendezvous Along Elliptica Paths*, Bringham Young University, AIAA Guidance, Navigation, and Control Conference 2012, Minneapolis, USA.

[8] Niki Regina, Matteo Zanzi, *Camera Pan-Tilt Gimbals Robust Control Law for Target Tracking with Fixed Wing UAV*, Università di Bologna.

[9] Fiorenzo Posterivo, *Sviluppo di un sistema per acquisizione dati ed esecuzione automatizzata di prove di volo su un aeromodello, in supporto ad attività didat-*

tiche sull'Identificazione Parametrica, Politecnico di Milano, Corso di laurea in Ingegneria Aeronautica, Tesi magistrale a.a. 2010/2011.

[10] *Arduino project home page*, Disponibile all'indirizzo:
http://arduino.cc/en/

[11] *ArduPilot Mega Documentation*, Disponibile all'indirizzo:
http://code.google.com/p/ardupilot-mega/

[12] *APM Mission Planner Documentation*, Disponibile all'indirizzo:
http://code.google.com/p/ardupilot-mega/wiki/Mission

[13] *MAVLink Micro Air Vehicle Communication Protocol Documentation*, Disponibile all'indirizzo:
http://www.qgroundcontrol.org/mavlink/start

[14] *DIYDrones forum*, Disponibile all'indirizzo:
http://www.diydrones.com