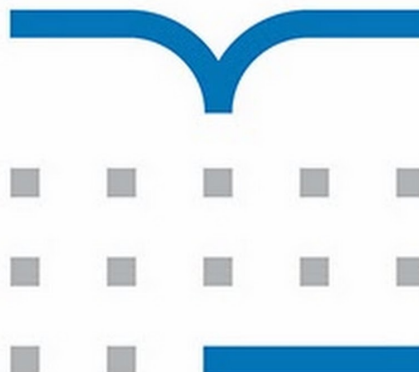




**MASTER DEGREE IN AEROSPACE
ENGINEERING ACCADEMIC YEAR 2024-2025**

**MANNED - UNMANNED TEAMING
controllore di volo per UAV in formazione
col velivolo principale.**

CANDIDATO: Antonio Scazzi
RELATRICE ACCADEMICA: Prof.ssa Elisa Capello, Politecnico di Torino
RELATORE AZIENDALE: Mario Nigra, Sipal s.p.a

Dichiarazioni

Questa tesi magistrale é stata realizzata grazie a una collaborazione tra il Politecnico di Torino e SIPAL S.p.a ed é il proseguimento e l'evoluzione di una tesi precedentemente svolta sullo stesso simulatore. L'Autore dichiara che questa tesi rappresenta un lavoro originale, svolto dopo l'iscrizione al corso di Laurea Magistrale in Ingegneria Aerospaziale presso il Politecnico di Torino. Questo lavoro non é stato precedentemente incluso in una tesi o dissertazione presentata presso questa o qualsiasi altra istituzione per il conseguimento di un titolo di studio, diploma o altra qualifica. Il software e la documentazione legati al codice presente in questo documento sono soggetti alla licenza MIT.

Abstract

Gli UAV (Veicoli Aerei Senza Pilota) assumono un ruolo sempre più centrale in ambito militare e civile grazie alla loro capacità di ridurre i rischi per gli operatori umani in ambienti ostili. Tuttavia, al di fuori di ambienti controllati, i loro impegni operativi sono ancora limitati. La maggior parte degli UAV attuali è controllata in remoto o in modalità semi-autonoma, con scarso livello di adattamento a situazioni impreviste inoltre l'integrazione in spazi aerei condivisi rimane complessa per motivi di sicurezza e regolamentazione. Le nuove tecnologie puntano a migliorare l'autonomia mediante sistemi avanzati di intelligenza artificiale, algoritmi di navigazione evoluti e l'introduzione del concetto di Manned-Unmanned Teaming (MUM-T), ovvero la capacità di velivoli senza pilota di supportare quelli con equipaggio durante una missione.

Questa tesi, frutto della collaborazione tra il Politecnico di Torino e Sipal S.p.a e sviluppata durante uno stage presso l'azienda, si inserisce in questo contesto. Utilizzando l'hardware del laboratorio di Sipal e il software Prepar3d, è stato progettato un autopilota per velivoli ad ala fissa che include le fasi di decollo, immissione in crociera e volo in formazione, con l'obiettivo di simulare uno scenario MUM-T. Poiché sono stati utilizzati due elaboratori connessi sulla stessa rete locale, si è deciso di adottare un'architettura di tipo leader-follower. Sono stati quindi implementati due algoritmi, uno per ciascuna macchina, che gestiscono l'automazione dei task, all'interno dei quali diversi autopiloti, utilizzando controllori PID (Proporzionali integrali derivativi), e LEAD-LAG (filtri d'anticipo o ritardo) portano a termine la missione. Sono state quindi effettuate diverse simulazioni in condizioni standard e perturbate al fine di produrre dei grafici che descrivono l'andamento temporale delle grandezze di interesse. Il sistema può essere adattato al tipo di aeromobile selezionato e alle esigenze della missione tramite la regolazione di specifici parametri di input.

Indice

1	Introduzione	1
2	Ambiente della simulazione	3
2.1	Hardware	3
2.1.1	Postazione Principale	3
2.1.2	Postazione Secondaria	4
2.2	Software	5
2.2.1	Simulatore	5
2.2.2	SDK	6
2.2.3	Wrapper	7
3	Fondamenti matematici	9
3.1	Sistemi di riferimento	9
3.1.1	WGS 84	9
3.1.2	ECEF	11
3.1.3	NED	11
3.1.4	BODY	12
3.2	Modello aeromeccanico	13
3.3	Meccanismi di controllo	15
3.3.1	PID	15
3.3.1.1	Antiwindup	16
3.3.2	Lead-Lag	17
3.4	Approccio Leader-Follower	19
4	Autopilota di formazione	21
4.1	Velivolo	21
4.2	Struttura del codice	23
4.3	Gestione della missione	24
4.4	Trasformazioni di coordinate	25
4.5	Acquisizione dati	27
4.6	Controllori	28
4.6.1	Pitch Hold	29
4.6.2	Bank Hold	31
4.6.3	Autothrottle	33
4.6.4	Altitude Hold	35
4.6.5	Heading Hold	38
4.6.6	Vertical Separation	41
4.6.7	Lateral Separation	44
4.6.8	Forward Separation	47
4.7	Simulazioni	50

4.7.1	Condizioni non perturbate	50
4.7.2	Condizioni perturbate	50
5	Risultati	52
5.0.1	Caso non perturbato	52
5.0.2	Caso perturbato	67
5.1	Conclusioni	78
5.2	Sviluppi futuri	78
A	Documentazione Aggiuntiva	81

Elenco delle figure

1	Formation Flight	1
2	Logo P3D	5
3	Grafico WGS84	9
4	ECEF e NED	11
5	SR in uso.	13
6	Approccio Leader follower - piano XY.	19
7	F35 -Disegno 3 viste.	21
8	Schema a blocchi del codice	23
9	Fasi di missione del leader.	25
10	Fasi di missione del follower.	25
11	Schermata principale addon	27
12	Schema a blocchi del pitch hold	30
13	Schema a blocchi del bank hold	32
14	Schema a blocchi dell'autothrottle	34
15	Schema a blocchi dell'altitude hold	37
16	Schema a blocchi dell'heading hold	40
17	Schema a blocchi Vertical Separation Autopilot	43
18	Schema a blocchi del lateral separation autopilot	46
19	Schema a blocchi del forward separation autopilot	49
20	Pitch Hold.	52
21	Bank Hold.	53
22	Autothrottle.	54
23	Altitude Hold.	55
24	Heading Hold.	56
25	Forward separation autopilot.	57
26	Lateral separation autopilot.	58
27	Vertical separation autopilot.	59
28	Mission Trajectory - 3D	61
29	Mission Trajectory - Top	61
30	Manovra di decollo e avvicinamento.	62
31	NED decollo e avvicinamento.	62
32	Manovra di richiamata.	63
33	NED richiamata.	63
34	Manovra di virata.	64
35	NED virata.	64
36	Manovra di virata ascendente.	65
37	NED virata ascendente.	65
38	Comandi manuali.	66
39	NED comandi manuali.	66

40	Grafico Pitch hold.	67
41	Grafico Bank Hold.	68
42	Grafico autothrottle.	69
43	Grafico altitude hold.	70
44	Grafico heading hold.	71
45	Grafico forward separation autopilot.	72
46	Grafico lateral separation autopilot.	73
47	Grafico vertical separation autopilot.	74
48	Manovra di decollo e avvicinamento perturbati.	75
49	NED decollo e avvicinamento perturbati.	75
50	Manovra di richiamata perturbata.	76
51	NED richiamata perturbata.	76
52	Manovra di Virata perturbata.	77
53	NED virata perturbata.	77

1 Introduzione

La presente Tesi di Laurea Magistrale in Ingegneria Aerospaziale é frutto della collaborazione tra il Politecnico di Torino e Sipal S.p.A.; a partire da febbraio 2025, l'autore ha avuto l'opportunità di svolgere uno stage di sei mesi presso il laboratorio HMI di Sipal a Torino, durante il quale é stata realizzata gran parte del lavoro successivamente descritto. Il progetto si colloca nel contesto del *Manned-Unmanned Teaming* (MUM-T), una modalità avanzata di gestione della missione che prevede la cooperazione tra sistemi con equipaggio umano e sistemi autonomi di diversa natura, aerei, terrestri, marittimi, subacquei e spaziali, con l'obiettivo di migliorare la *situational awareness*, l'efficacia operativa e la sicurezza in scenari ad alto rischio, riducendo al contempo il numero di risorse umane necessarie. In tale quadro, il controllo in formazione di veicoli senza pilota assume un ruolo centrale, poiché il volo coordinato consente non solo di accrescere l'efficacia della missione, ma anche di ottenere benefici in termini di efficienza di personale e di sicurezza di quest'ultimo. I livelli di autonomia degli UAV possono variare dal pieno controllo remoto all'autonomia totale, rendendo cruciale lo studio delle modalità attraverso cui velivoli autonomi, eventualmente accompagnati da aeromobili con equipaggio, possano cooperare in formazione per il successo della missione.

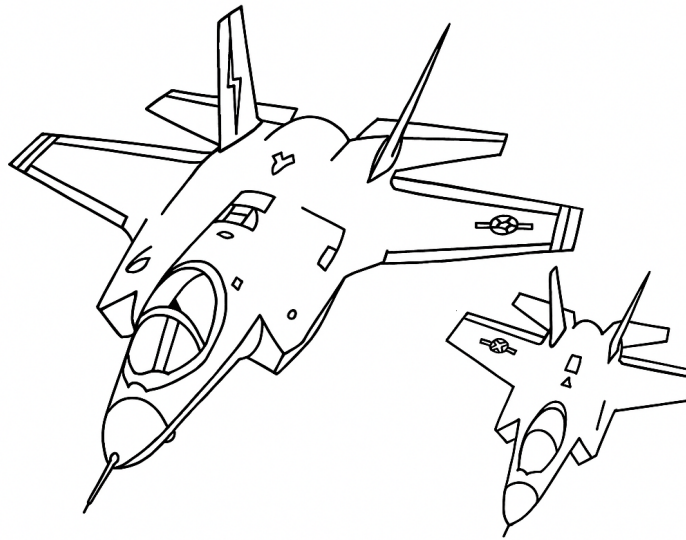


Figura 1: Formation Flight

Storicamente, il volo in formazione nasce nei primi decenni dell'aviazione militare, con i primi esperimenti della Prima Guerra Mondiale volti a migliorare la copertura visiva e la protezione reciproca, mentre durante la Seconda Guerra Mondiale divenne una componente strategica fondamentale, con schemi tattici specifici impiegati per aumentare l'efficacia dei bombardamenti e ridurre la vulnerabilità agli attacchi nemici. Nel periodo della Guerra Fredda, lo sviluppo tecnologico e delle comunicazioni portò a tattiche più sofisticate, mentre in ambito civile il volo in formazione rimase rilevante per manifestazioni aeree, missioni di ricerca e applicazioni sperimentali. Con l'avvento dell'era dell'informazione e delle *Netcentric Operations*,

il concetto si é esteso ai droni, oggi capaci di operare autonomamente in ricognizione, sorveglianza, monitoraggio ambientale e missioni civili quali mappatura, agricoltura di precisione e gestione delle emergenze. In parallelo, la tecnologia MUM-T ha conosciuto un'evoluzione significativa: sebbene le prime sperimentazioni risalgano al periodo tra le due guerre, la definizione ufficiale fu introdotta solo nel 2013 dall'U.S. Army Aviation Center, che la descrisse come l'impiego congiunto e sincronizzato di soldati, sistemi con e senza equipaggio, robotica e sensori per accrescere la letalità, la sopravvivenza e la situational awareness. L'obiettivo principale era quello di ridurre l'esposizione al rischio per gli operatori, delegando ai veicoli senza pilota le missioni più pericolose, come la ricognizione in aree ostili. Un esempio emblematico é rappresentato dall'elicottero AH-64E, capace non solo di ricevere dati dai droni, ma anche di controllarne i sensori e la navigazione, integrando così le informazioni nel processo decisionale. I consistenti investimenti militari hanno favorito lo sviluppo rapido della tecnologia, oggi adottata anche in ambito civile: si pensi alla collaborazione avviata nel 2022 dal Regno Unito tra HM Coastguard e Bristow Helicopters per l'implementazione di servizi MUM-T a supporto delle operazioni di ricerca e soccorso marittimo. Parallelamente, anche i modelli di controllo degli UAV hanno seguito un percorso evolutivo scandito dal progresso tecnologico: negli anni '60,'80 erano predominanti i controllori PID, semplici ma efficaci per la stabilità del volo; negli anni '90, grazie all'aumento della capacità di calcolo, si diffusero approcci ottimi come il LQR e strumenti di stima avanzata come il Filtro di Kalman e la sua versione estesa EKF; nei primi anni 2000, il focus si spostó sulla robustezza, con l'adozione di controlli H_∞ , adattivi e a *sliding mode*. L'avvento dell'intelligenza artificiale negli anni 2010 segnó una svolta, con l'applicazione di reti neurali, logica fuzzy e *reinforcement learning*, capaci di apprendere modelli di controllo complessi direttamente dall'interazione con l'ambiente. Oggi, il panorama é dominato da tecniche predittive come l'MPC (*Model Predictive Control*) e da strategie cooperative multi-UAV, che si ispirano alla *swarm intelligence* per coordinare sciami di droni in maniera distribuita e adattiva. Inoltre, approcci ibridi che combinano metodi classici e intelligenza artificiale stanno aprendo la strada a missioni autonome sempre più complesse, collaborative e critiche.

Gli obiettivi di questa tesi quindi si collocano nel medesimo contesto di Manned-Unmanned Teaming (MUM-T) e mirano a dimostrare la possibilità di integrare in maniera efficace velivoli con e senza pilota in uno scenario cooperativo. In particolare, il lavoro si propone di sviluppare e validare un sistema di controllo per UAV ad ala fissa, capace di interagire con un velivolo leader pilotato in modo stabile e affidabile. La progettazione dell'autopilota si é concentrata sulla definizione di algoritmi di controllo per le principali fasi operative, includendo decollo, salita, volo livellato e mantenimento della formazione, attraverso l'impiego di controllori PID e compensatori di tipo lead-lag, opportunamente calibrati per garantire sia rapidità di risposta che robustezza alle perturbazioni esterne.

Parallelamente la tesi ha perseguito l'obiettivo di garantire la scalabilità della piattaforma, prevedendo la possibilità di estendere il numero di velivoli simulati e di integrare in futuro sensori e moduli software aggiuntivi senza richiedere modifiche invasive all'hardware esistente. Infine, si é posta importanza nella validazione sperimentale del sistema, mediante la raccolta e l'elaborazione dei dati di simulazione, con lo scopo di quantificare in maniera oggettiva le prestazioni dei controllori in termini di tempi di risposta, errori a regime e capacità di mantenere la separazione nella formazione. L'elaborato intende quindi non soltanto proporre una soluzione tecnica innovativa per il controllo cooperativo di UAV, ma anche fornire un framework sperimentale solido e replicabile, utile come base di partenza per studi futuri e per l'evoluzione verso applicazioni operative reali. [1, 2]

2 Ambiente della simulazione

Un ambiente di simulazione può essere definito come una rappresentazione virtuale del mondo reale o di un suo sottoinsieme, concepita per riprodurre fenomeni fisici, dinamiche operative o condizioni ambientali in maniera controllata e ripetibile. Esso costituisce uno strumento fondamentale per l'analisi, l'addestramento e la sperimentazione, in quanto consente di osservare e valutare il comportamento di sistemi complessi senza la necessità di operare direttamente nel contesto reale, riducendo costi, rischi e vincoli logistici. Un ambiente simulato integra tipicamente più componenti tra loro interconnessi: la descrizione dello scenario, che comprende la modellazione del terreno, delle infrastrutture e degli elementi contestuali; la riproduzione delle condizioni meteorologiche e ambientali, in grado di variare su diverse scale temporali e spaziali; e la presenza di entità interattive, come veicoli, macchine o attori virtuali, dotati di caratteristiche dinamiche e comportamentali specifiche. Tali ambienti possono essere impiegati per finalità differenti, dal supporto all'addestramento del personale alla validazione di algoritmi di controllo o strategie decisionali, fino alla sperimentazione di scenari operativi complessi che coinvolgono più domini e attori. Un aspetto cruciale è la possibilità di riprodurre condizioni iniziali definite con precisione, garantendo la ripetibilità degli esperimenti e quindi la validità del confronto tra differenti soluzioni o approcci. Inoltre, la natura modulare e scalabile di tali sistemi consente l'estensione progressiva delle funzionalità, ad esempio introducendo nuovi modelli, regole comportamentali o vincoli ambientali, in modo da aumentare il grado di realismo o adattare il simulatore a specifiche esigenze applicative. Nelle sezioni successive verranno descritti i componenti sia fisici che software utilizzati per questo progetto.

2.1 Hardware

2.1.1 Postazione Principale

La postazione principale del sistema di simulazione è stata progettata e realizzata con particolare attenzione alla scelta e all'integrazione dei componenti hardware, in modo da garantire un ambiente di lavoro stabile, scalabile e il più realistico possibile. Questa servirà per simulare il cockpit del velivolo leader pilotato.

Al centro dell'allestimento vi è un computer aziendale già disponibile, equipaggiato con un processore Intel Core i7 di settima generazione e due schede grafiche Nvidia T1000. La decisione di utilizzare due GPU è stata determinata principalmente da due esigenze: garantire un adeguato livello di prestazioni, distribuendo il carico di calcolo grafico, e offrire la scalabilità necessaria per integrare in futuro ulteriori schermi senza dover intervenire con aggiornamenti hardware invasivi.

La prima GPU è dedicata al collegamento con il proiettore, che rappresenta l'elemento visivo centrale della simulazione, mentre la seconda gestisce tre monitor, utilizzati per fornire angoli di visione aggiuntivi e per ospitare interfacce di controllo. Questa configurazione consente di mantenere un ambiente immersivo e, allo stesso tempo, funzionale per la gestione del sistema. Nonostante la complessità della configurazione, la postazione garantisce comunque una resa fluida con una media di 24fps, valore sufficiente per assicurare un'esperienza di simulazione stabile e realistica.

Per quanto riguarda la proiezione, è stato selezionato un videoproiettore Epson EH-LS650B a tiro ultracorto scelto a causa di vincoli logistici e da requisiti di qualità: nel laboratorio non era possibile installare un proiettore a soffitto e il posizionamento frontale permetteva di

evitare la formazione di ombre sulla zona di seduta. Grazie alla tecnologia a tiro ultracorto, é stato possibile ottenere uno schermo di grandi dimensioni in uno spazio ridotto, ottimizzando cosí l'utilizzo dell'ambiente. A ciò si aggiungono l'elevata luminosità e la fedeltà cromatica del dispositivo, aspetti fondamentali per riprodurre scenari realistici e coinvolgenti.

Accanto al proiettore, la postazione integra diversi schermi dedicati a funzioni specifiche. Un monitor touchscreen da 24 pollici (DELL P2424HT) é utilizzato per riprodurre il display multifunzionale del velivolo, fornendo dati essenziali sul volo e sullo stato dei sistemi. A questo si affianca un monitor touchscreen da 14 pollici (Verbatim PMT-14), che visualizza il GPS, migliorando la navigazione e la consapevolezza situazionale durante la simulazione.

Per completare la postazione sono stati acquistati controlli dedicati e una seduta specifica. I comandi, dotati di numerosi pulsanti configurabili, consentono un'interazione diretta e dettagliata con l'ambiente virtuale, simulando in maniera fedele i comandi presenti in un cockpit reale. La seduta scelta é il modello Next Level Racing Flight Simulator - Boeing Military Edition, completamente regolabile e predisposta per l'aggiunta di accessori. La sedia permette di adattare la configurazione a diversi scenari.

I controlli principali sono costituiti dal sistema HOTAS Warthog e dai pedali TPR Rudder, entrambi prodotti da Thrustmaster. Il joystick e la doppia manetta del Warthog utilizzano sensori magnetici a effetto Hall, in grado di garantire precisione elevatissima e assenza di usura meccanica. Il joystick interamente in metallo offre un feedback realistico con una resistenza calibrata dei pulsanti, mentre la throttle, dotata di sistema a doppia leva, include meccanismi di idle e afterburner con blocchi meccanici a scatto che riproducono fedelmente quelli presenti in cabina. L'insieme fornisce oltre trenta comandi programmabili tra pulsanti, interruttori e hat switch, permettendo di gestire in maniera completa le principali funzioni di pilotaggio e di missione.

A complemento, i pedali TPR Rudder sfruttano il sistema a pendolo T.Pendular che consente un'escursione fluida e progressiva, con angoli regolabili fra 35° e 75° . Ciascun pedale integra freni differenziali indipendenti, utili per manovre di rullaggio e controllo fine in fase di atterraggio. Il dispositivo é completamente configurabile tramite software oltre a poter essere integrato con gli altri prodotti Thrustmaster grazie al pacchetto T.A.R.G.E.T., cosí da gestire l'intero sistema come un unico dispositivo.

L'adozione congiunta di HOTAS Warthog e TPR Rudder, montati sulla struttura Boeing Military Edition, permette quindi di ricreare una postazione immersiva e al tempo stesso stabile, in grado di restituire sensazioni estremamente vicine al volo reale, sia per quanto riguarda il comfort, sia per la fedeltà nell'interazione con i comandi. [3]

2.1.2 Postazione Secondaria

La postazione secondaria del sistema di simulazione é stata concepita con un approccio piú leggero e compatto, pensato per garantire comunque la possibilità di svolgere attività e test in parallelo a quella principale. Essa é basata su un laptop Alienware ad alte prestazioni, collegato a un monitor esterno di grandi dimensioni che funge da display principale, cosí da assicurare un'ampia superficie visiva per la simulazione. Accanto al notebook é presente un secondo schermo portatile touch, che può essere utilizzato per visualizzare pannelli di controllo aggiuntivi o interfacce software di supporto. Questa configurazione, pur meno immersiva rispetto alla postazione leader, consente comunque una gestione ordinata e funzionale delle informazioni necessarie al volo virtuale.

Dal punto di vista dei controlli, la postazione é equipaggiata con una periferica Thrustmaster

T.16000M FCS, un dispositivo dotato di numerosi pulsanti programmabili e di un hat switch multidirezionale che permette di riprodurre i principali comandi di volo in maniera intuitiva e immediata. questo controllo però non verrà utilizzato in quanto la postazione secondaria é relegata alla simulazione del drone follower che non necessita di controlli.

Complessivamente, la postazione secondaria si presenta come una soluzione scalabile e facilmente adattabile, utile sia come sistema di supporto al cockpit principale, sia come piattaforma autonoma per esercitazioni di natura specifica. Pur rinunciando a seduta dedicata e pedali professionali, mantiene un livello di fedeltá adeguato e rappresenta un complemento strategico all'ecosistema di simulazione, consentendo di simulare piú elementi all'interno dello stesso scenario di addestramento senza richiedere le stesse risorse logistiche della postazione leader.

2.2 Software

2.2.1 Simulatore

Prepar3D, sviluppato da Lockheed Martin, é una piattaforma di simulazione professionale progettata per l'addestramento e la formazione in ambito aeronautico, terrestre e marittimo. Il suo scopo é fornire un ambiente immersivo per preparare operatori, piloti e personale specializzato a gestire scenari complessi, sia in condizioni normali che di emergenza. Nonostante la somiglianza con i simulatori di volo di tipo consumer, Prepar3D non é un videogioco, ma uno strumento certificato e sviluppato per finalitá professionali, accademiche e militari. Dal



Figura 2: Logo P3D

punto di vista tecnico, Prepar3D eredita le basi da Microsoft ESP, ma é stato completamente evoluto negli anni per rispondere a esigenze moderne di prestazioni e fedeltá visiva. A partire dalla versione 4, il passaggio al modello a 64 bit ha permesso di superare i limiti di memoria dei sistemi precedenti, consentendo scenari piú estesi e modelli di maggiore complessitá. Con la versione 5, il motore grafico é stato aggiornato a DirectX 12, introducendo atmosfere volumetriche, riflessioni fisicamente corrette e un modello idrico piú realistico. L'attuale versione 6 ha ulteriormente perfezionato il rendering, migliorando l'illuminazione globale, le condizioni atmosferiche e introducendo tecniche moderne come DLSS/DLAA, FSR2 e Screen Space Reflections, con un significativo guadagno in termini di qualitá e performance.

Dal punto di vista funzionale, Prepar3D mette a disposizione strumenti avanzati per la creazione e la gestione degli scenari. SimDirector é l'editor integrato che consente di progettare missioni e addestramenti interattivi, definendo condizioni iniziali, logiche di evento e comportamenti dinamici. Per l'ambito piú strettamente professionale (quella utilizzata in questo

progetto), chiamata Professional Plus, é disponibile anche SimOperator, un Instructor Operator Station che permette di controllare in tempo reale l'andamento di una simulazione, intervenendo su parametri ambientali o introducendo eventi critici per testare la capacità di reazione dell'operatore.

La piattaforma é aperta all'integrazione grazie a un ricco SDK. Le API SimConnect consentono a sviluppatori esterni di accedere a dati e funzioni del simulatore, mentre il PDK (Prepar3D Development Kit) permette un'integrazione a basso livello per applicazioni che richiedono massima efficienza. In questo modo, Prepar3D non é solo un simulatore pronto all'uso, ma anche un framework espandibile che può adattarsi a contesti addestrativi molto diversi

Dal punto di vista dei contenuti, Prepar3D include una rappresentazione completa del globo terrestre basata sul sistema WGS-84, con oltre 24.000 aeroporti, traffico terrestre, marittimo e aereo generato dall'intelligenza artificiale, e scenari arricchiti da vegetazione e infrastrutture 3D. Le capacità immersive sono estese anche al supporto per la realtà virtuale e mista, con compatibilità verso i principali visori professionali. Nelle versioni utilizzate, il software consente inoltre la simulazione di sensori specifici come NVG (Night Vision Goggles), camere a infrarossi e radar, aprendo l'utilizzo anche a contesti militari e di difesa.

Un aspetto cruciale riguarda il modello di licenza. Prepar3D é distribuito in versioni differenziate: Personal, rivolta a un uso domestico e scolastico di base; Professional, destinata ad aziende e centri di formazione; e Professional Plus, che aggiunge le capacità di simulazione avanzata, inclusi armamenti, interoperabilità con protocolli di simulazione distribuita come DIS e CIGI, e il già citato SimOperator. [4]

2.2.2 SDK

L'SDK di Prepar3D é il tassello che trasforma la piattaforma di Lockheed Martin da semplice simulatore a ecosistema estendibile: consente di modellare veicoli e sensori, generare scenari e traffico, integrare pannelli e strumenti, automatizzare logiche via script e collegare applicazioni esterne o plug-in nativi al motore di simulazione. La documentazione ufficiale organizza l'SDK in kit funzionali e ne indica il percorso di installazione predefinito su Windows, all'interno di Program Files. In questa architettura l'assunto chiave é che Prepar3D sia data-driven: nuove risorse, modelli e scenari possono essere aggiunti senza un limite imposto dal motore, purché conformi ai formati supportati.

La parte Modeling é, per molti sviluppatori, il cuore operativo: ogni aereo, mezzo terrestre o marittimo, edificio o infrastruttura nasce da un modello 3D in un formato specifico del simulatore, tipicamente creato con 3ds Max e corredato da animazioni, materiali PBR, effetti e illuminazione. Per gli effetti particellari l'SDK fornisce un tool dedicato e un formato testuale.

I Simulation Objects definiscono struttura e comportamento dei veicoli e degli altri oggetti simulati nel mondo. La configurazione é file-based e separa in modo ordinato gli aspetti estetici quali modelli, texture e suoni, da quelli funzionali: sistemi: motori, carburante, avionica, carrelli, aerodinamica, camere e effetti. Per gli aeromobili, i file AIR e le tabelle di coefficienti determinano la dinamica di volo, mentre le cfg orchestrano varianti e dotazioni; la stessa logica, semplificata, si applica ad altri oggetti simulabili. Questo approccio rende riproducibile il tuning dei sistemi e favorisce la manutenzione nel ciclo di vita del contenuto.

Sul fronte World, l'SDK espone gli strumenti per plasmare l'ambiente: lo stack BGL per scenari e aeroporti, il Terrain SDK composto da Resample per l'ortofoto/elevazioni, TmfViewer

per l'ispezione dei livelli di dettaglio e Shp2Vec per convertire forme vettoriali in dati nativi, l'Autogen per generare edificato e vegetazione e i tool per il traffico AI aereo e marittimo. É un set coerente che copre il ciclo completo: dalla sorgente alla priorità e fusione in runtime mediante i file di configurazione.

La strumentazione di bordo e l'interfaccia utente sono affrontate nella sezione Panels and UI: pannelli in finestra o in cabina possono essere definiti via XML o Scaleform, orchestrati dal panel.cfg e duplicati con sincronizzazione di stato. A completare il comparto "percepito" c'è l'estensione Cameras, che ammette configurazioni articolate e post-process custom (HLSL) applicabili alla render chain con accesso a colore e profondità: meccanismo pensato tanto per sensori simulati (NVG/IR, pod di puntamento) quanto per la resa grafica.

La produzione di scenari didattici é delegata a SimDirector, l'editor integrato che copre l'intero flusso: dall'editing della scena alle modalità "Virtual Instructor" per registrare interazioni in cockpit, fino a Flight Instructor per catturare manovre e valutarle con criteri oggettivi. La modalità Preview offre strumenti di debug e profiling in tempo reale, mentre la possibilità di creare delle configurazioni dello scenario nelle parti di briefing, multiplayer e realismo consente di confezionare esperienze strutturate, singole o multi-ruolo. Per chi sviluppa contenuti formativi, SimDirector riduce drasticamente i tempi tra ideazione, verifica e distribuzione.

Per l'interazione dall'esterno il riferimento é l'API SimConnect. Il modello é client-server: il simulatore pubblica servizi e uno o più client (anche remoti) si collegano per leggere SimVars, ricevere eventi o trasmetterne di nuovi, manipolare AI e strutture, fino a controllare camere e meteo. La documentazione v6 raccomanda di privilegiare progetti out-of-process (eseguibili) per robustezza e facilità di debug, lasciando ai wrapper .NET la strada breve per GUI e strumenti. é la via maestra per telemetria, pannellistica esterna, IOS personalizzate e automazioni di scenario.

2.2.3 Wrapper

Nel panorama delle piattaforme di simulazione di volo, una delle componenti chiave per garantire estensibilità e integrazione con sistemi esterni é SimConnect. Si tratta dell'API ufficiale sviluppata inizialmente da Microsoft e oggi mantenuta sia nell'ambito di Microsoft Flight Simulator che in quello di Prepar3D di Lockheed Martin. Il suo scopo é fornire un'interfaccia standard attraverso la quale applicazioni esterne, o moduli caricati all'interno del simulatore, possono interagire con il motore di simulazione, accedendo a dati in tempo reale o inviando comandi ed eventi. In altre parole, SimConnect rappresenta il "ponte" che consente di trasformare un simulatore da applicazione isolata a piattaforma aperta e integrabile.

Dal punto di vista architetturale, SimConnect adotta un modello di tipo client-server. Il simulatore stesso si comporta come server, mentre uno o più programmi esterni assumono il ruolo di client. Questa separazione consente non solo di eseguire applicazioni sulla stessa macchina del simulatore, ma anche di distribuire il carico su più computer collegati in rete. La connessione viene stabilita attraverso un'API dedicata, con la possibilità di utilizzare canali locali oppure protocolli TCP/IP su IPv4 o IPv6. La configurazione avviene mediante due file: SimConnect.xml lato server, che definisce gli endpoint disponibili, e SimConnect.cfg lato client, dove vengono specificati parametri come indirizzo, porta e protocollo. Questa flessibilità permette di realizzare scenari che spaziano dall'applicazione singola, come un pannello strumenti virtuale, fino a laboratori complessi con più postazioni e istruttori collegati.

La comunicazione é interamente event-driven. Una volta stabilita la connessione, l'applicazione client può sottoscrivere variabili di simulazione oppure mappare eventi e trasmetterli al

simulatore. Le variabili disponibili coprono l'intero spettro del modello simulativo: dall'altitudine e velocità del velivolo, allo stato dei sistemi di bordo, fino a parametri ambientali come vento, temperatura o visibilità. É possibile decidere con quale frequenza ricevere i dati: ogni fotogramma di simulazione, ogni secondo, oppure soltanto quando il valore cambia, opzione molto utile per ridurre il traffico e ottimizzare le prestazioni. Allo stesso modo, gli eventi permettono di pilotare funzioni interne al simulatore, come l'attivazione del pilota automatico, il settaggio delle luci di bordo o l'inserimento di anomalie.

Oltre alla gestione di variabili e comandi, SimConnect offre meccanismi per interagire con il mondo simulato. Attraverso specifiche chiamate é possibile creare e controllare oggetti AI, come aeromobili o veicoli, oppure interrogare i database delle strutture aeroportuali e radionavigazionali. Sono disponibili anche aree di memoria condivisa chiamate Client Data, utili per lo scambio di informazioni tra più applicazioni collegate allo stesso simulatore, senza dover implementare ulteriori canali di comunicazione. Tutti questi strumenti concorrono a trasformare il simulatore in un hub interoperabile, adatto sia ad applicazioni civili che militari.

3 Fondamenti matematici

3.1 Sistemi di riferimento

3.1.1 WGS 84

Il *World Geodetic System 1984* (WGS 84) rappresenta oggi il sistema di riferimento geodetico di uso globale ed è quello adottato dal sistema GPS. Si tratta di un modello matematico e fisico che descrive la forma della Terra, la sua posizione nello spazio e il modo in cui è possibile esprimere le coordinate di qualsiasi punto della superficie terrestre in maniera coerente.

Il sistema è costruito attorno a un ellissoide di riferimento, cioè una superficie matematica che approssima la forma reale della Terra, la quale non è perfettamente sferica ma leggermente schiacciata ai poli. L'ellissoide WGS 84 è definito da un semi-asse maggiore di 6.378.137 metri e da un appiattimento pari a $1/298,257223563$. Questi valori, frutto di osservazioni geodetiche a scala globale, permettono di ottenere una rappresentazione uniforme del pianeta, che costituisce la base per calcolare latitudine, longitudine e quota ellissoidica di un punto.

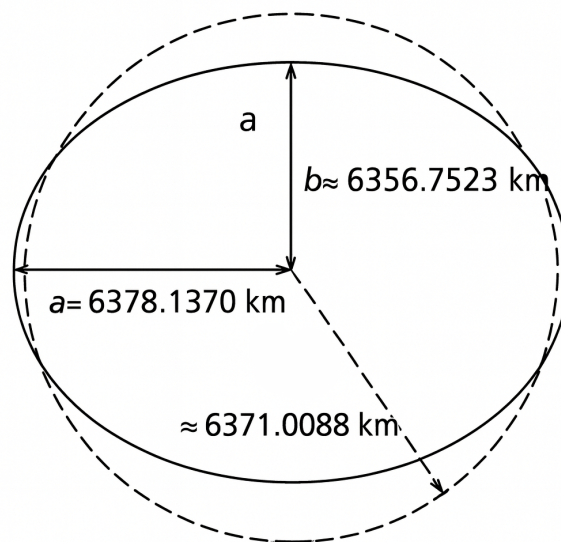


Figura 3: Grafico WGS84

Un aspetto fondamentale del WGS 84 è il suo datum geodetico, ovvero il modo in cui l'ellissoide è posizionato e orientato rispetto alla Terra reale. A differenza di molti datum locali, questo è di tipo geocentrico: l'origine del sistema coincide con il centro di massa terrestre, includendo oceani e atmosfera. L'asse Z è orientato lungo l'asse medio di rotazione terrestre, l'asse X passa per il meridiano di Greenwich all'equatore e l'asse Y completa la terna destrorsa. In questo modo, il sistema è direttamente collegato alla dinamica fisica del pianeta ed è coerente con i sistemi di riferimento internazionali utilizzati in geodesia.

Un altro elemento che rende il WGS 84 uno standard di riferimento è la sua realizzazione pratica. Il sistema non è statico, perché la Terra è soggetta a movimenti tettonici e variazioni dinamiche: per questo motivo, nel tempo sono state prodotte diverse versioni aggiornate, dette "realizzazioni", che mantengono l'allineamento con i frame di riferimento internazionali (ITRF). Dal punto di vista operativo, ciò significa che le coordinate fornite dai ricevitori

ri GPS sono già espresse in WGS 84, con una precisione che, a seconda delle tecniche di posizionamento, può andare dal livello del metro fino al centimetro. [5]

3.1.2 ECEF

Il sistema di coordinate ECEF (Earth-Centered, Earth-Fixed) é strettamente legato al WGS 84 e rappresenta il modo piú naturale per descrivere posizioni sulla Terra in tre dimensioni. Si tratta di un sistema cartesiano tridimensionale con origine nel centro di massa terrestre, lo stesso utilizzato dal WGS 84 per definire il proprio datum.

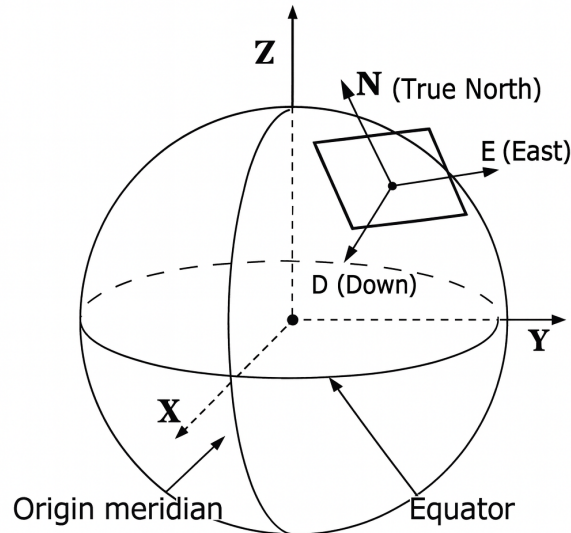


Figura 4: ECEF e NED

L'asse Z é orientato verso il polo nord geografico, coincidente con l'asse medio di rotazione terrestre. L'asse X giace sul piano equatoriale e passa per l'intersezione tra l'equatore e il meridiano di Greenwich. L'asse Y, infine, completa il sistema seguendo la regola della mano destra, e anch'esso si trova sul piano equatoriale, ortogonale rispetto all'asse X.

La caratteristica principale del sistema ECEF é che esso viene definito come *Earth-Fixed*, cioè ruota solidalmente con la Terra. Questo significa che le coordinate (X, Y, Z) di un punto fissato sulla superficie terrestre rimangono costanti nel tempo, almeno in assenza di movimenti tettonici o deformazioni locali. Ciò lo rende particolarmente utile per descrivere qualsiasi punto che debba mantenere stabilità rispetto alla crosta terrestre.

Le coordinate in ECEF sono espresse in metri e permettono un'immediata conversione con le coordinate geodetiche (φ, λ, h) , ossia latitudine, longitudine e quota ellissoidica. Questa trasformazione é alla base del funzionamento dei ricevitori GPS: i satelliti trasmettono dati che vengono elaborati direttamente in ECEF e solo in seguito convertiti in latitudine e longitudine per essere comprensibili agli utenti. [5]

3.1.3 NED

Il sistema di riferimento NED é un sistema cartesiano locale comunemente utilizzato in aeronautica e navigazione per descrivere i movimenti e le posizioni di un veicolo rispetto alla superficie terrestre. A differenza del sistema ECEF, che é globale e centrato nel centro della Terra, NED é legato al punto in cui si trova il veicolo e fornisce una rappresentazione intuitiva dei movimenti in prossimitá della superficie.

Il suo origine é tipicamente posto nel punto in cui si trova il velivolo o il veicolo di riferimento. L'asse X é orientato verso il Nord geografico, l'asse Y é orientato verso Est, mentre l'asse Z punta verso il basso, ossia nella direzione del centro della Terra. Questa scelta di convenzione, con l'asse Z orientato verso il basso, é molto diffusa in aviazione perché semplifica il calcolo delle quote e la rappresentazione della gravità, che risulta avere valori positivi lungo tale direzione.

Il sistema NED é particolarmente utile nelle applicazioni operative poiché fornisce una rappresentazione direttamente interpretabile dai piloti e dagli strumenti di bordo: spostarsi lungo l'asse X significa andare verso nord, lungo l'asse Y verso est e lungo l'asse Z significa scendere in altitudine. In questo modo, i sensori di navigazione come accelerometri, magnetometri e GPS possono tradurre le misure in coordinate facilmente gestibili per il controllo e la navigazione. Dal punto di vista matematico, il sistema NED si ottiene tramite una trasformazione dal sistema ECEF. Dati latitudine e longitudine del punto di riferimento, é possibile costruire la matrice di rototraslazione che porta dal frame globale (ECEF) al frame locale (NED). Questa operazione permette di collegare dati geodetici assoluti con informazioni locali utilizzabili per la navigazione e il controllo di un veicolo.

In sintesi, NED é un sistema locale, intuitivo e funzionale, che permette di descrivere i movimenti in prossimità della superficie terrestre con un'immediatezza che i sistemi globali come ECEF non offrono. [5]

3.1.4 BODY

Il sistema di riferimento BODY é un sistema cartesiano solidale con il veicolo stesso, spesso utilizzato in aeronautica e robotica per descrivere le grandezze cinematiche e dinamiche direttamente dal punto di vista del mezzo. A differenza del sistema NED, che rimane fisso rispetto alla Terra, il sistema BODY si muove e ruota insieme al velivolo.

L'origine del sistema é generalmente posto nel centro di gravità del veicolo. L'asse X é orientato lungo la direzione longitudinale del velivolo, cioè verso il muso; l'asse Y é orientato lateralmente, verso l'ala destra; infine, l'asse Z punta verso il basso, in direzione del ventre del velivolo. Questa convenzione é standard in ambito aeronautico e garantisce coerenza nelle formulazioni delle equazioni del moto.

L'utilizzo del sistema BODY é fondamentale per la dinamica del volo e per il controllo. Le forze aerodinamiche (portanza, resistenza, forza laterale) e i momenti (rollio, beccheggio, imbardata) vengono infatti descritti e calcolati in questo sistema, poiché agiscono direttamente sul corpo dell'aeromobile. Allo stesso modo, i sensori montati a bordo - come giroscopi e accelerometri - misurano grandezze fisiche espresse nel sistema BODY, che devono poi essere convertite in sistemi di riferimento più adatti alla navigazione (come NED o ECEF).

Dal punto di vista matematico, la trasformazione tra BODY e NED si ottiene tramite le angoli di Eulero (rollio, beccheggio, imbardata) o, in alternativa, tramite rappresentazioni più robuste come i quaternioni. Questi strumenti consentono di passare da un sistema solidale con il veicolo a uno solidale con la Terra, integrando così le misure di bordo con le esigenze della navigazione globale.

In definitiva, il sistema BODY rappresenta la prospettiva "interna" del veicolo, essenziale per la descrizione delle dinamiche di volo e per l'elaborazione dei segnali provenienti dai sensori. é in stretta relazione con NED ed ECEF, poiché consente di tradurre i dati locali acquisiti dal veicolo in coordinate interpretabili su scala geografica e operativa. [5]

3.2 Modello aeromeccanico

Prepar3D v6 é un simulatore parametrico che adotta un modello a sei gradi di libertà per un corpo rigido, in cui le forze e i momenti aerodinamici e propulsivi vengono determinati sulla base di coefficienti tabellari e scalari. Tali coefficienti sono definiti principalmente in due archivi: il file `.AIR`, che raccoglie le tabelle e le derivate aerodinamiche, e il file `aircraft.cfg`, che contiene informazioni su geometria, masse e inerzie, parametri di propulsione e scalari di taratura entrambi disponibili nell'SDK. L'insieme di questi dati consente al motore fisico di integrare le equazioni di Eulero-Newton in forma classica, attraverso interpolazioni lineari e correzioni in funzione del numero di Mach. [4]

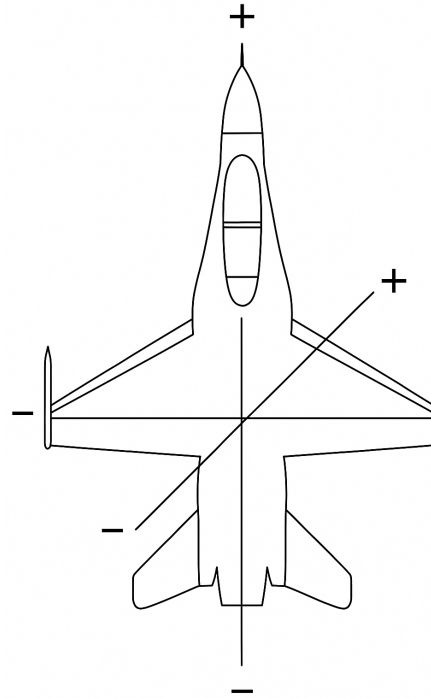


Figura 5: SR in uso.

Il sistema di riferimento utilizzato é quello riferito agli assi ortogonali mancini: l'asse longitudinale $+X$ punta in avanti, l'asse laterale $+Y$ verso destra e l'asse verticale $+Z$ verso l'alto. Le rotazioni seguono la convenzione: beccheggio positivo con il muso verso il basso, rollio positivo con l'ala sinistra abbassata e imbardata positiva verso destra. Questi segni valgono per tutte le derivate e i momenti aerodinamici.

Il modello aerodinamico si basa su due insiemi di dati.

Da un lato, il file `.AIR` contiene i coefficienti aerodinamici di base, organizzati in tabelle per portanza, resistenza, momento di beccheggio, forze laterali, momenti di rollio e imbardata. Sono inoltre presenti curve di portanza e di momento in funzione dell'angolo d'attacco, così come tabelle correttive dipendenti dal numero di Mach.

Dall'altro lato, il file `aircraft.cfg` definisce geometria, masse, inerzie e parametri di propulsione, oltre a scalari che permettono di modificare i coefficienti tabellari, facilitando il processo di taratura.

Tutte le tabelle sono interpolate linearmente e, se l'input eccede l'intervallo, i valori vengono mantenuti costanti agli estremi (*clamping*).

Le Mach tables utilizzano una discretizzazione fissa da Mach 0 a Mach 3.2 con passo 0.2, ed ogni voce viene normalizzata.

Il nucleo del modello é costituito dalle equazioni di Newton Eulero per un corpo rigido a sei gradi di libert , espresse nel sistema corpo:

$$m \dot{\mathbf{V}}_b + \boldsymbol{\omega}_b \times (m \mathbf{V}_b) = \mathbf{F}_b, \quad (1)$$

$$I \dot{\boldsymbol{\omega}}_b + \boldsymbol{\omega}_b \times (I \boldsymbol{\omega}_b) = \mathbf{M}_b, \quad (2)$$

dove m é la massa, $\mathbf{V}_b$ la velocit  traslazionale nel sistema corpo, $\boldsymbol{\omega}_b = [p \ q \ r]^T$ la velocit  angolare e I il tensore d'inerzia. Le forze totali $\mathbf{F}_b$ comprendono contributi aerodinamici, propulsivi, gravitazionali e da contatto, mentre i momenti $\mathbf{M}_b$ includono sia quelli aerodinamici sia quelli derivanti dai bracci di spinta dei motori.

I coefficienti aerodinamici sono ricavati dal file `.AIR` e successivamente corretti tramite scalari del

`aircraft.cfg`. In forma generale:

$$C_X(\alpha, \beta, \delta, M) = C_{X, \text{base}}(\cdot) + \Delta C_X^{\text{Mach}}(M), \quad (3)$$

dove $C_{X, \text{base}}$ proviene dalle tabelle AIR, $\Delta C_X^{\text{Mach}}(M)$ dalle Mach tables e infine il risultato é moltiplicato dagli scalari presenti in `[flight_tuning]`.

Le forze aerodinamiche principali sono:

$$L = qSC_L, \quad D = qSC_D, \quad Y = qSC_Y, \quad (4)$$

mentre i momenti, calcolati rispetto agli assi corpo, sono:

$$M_x = qSbC_l, \quad M_y = qS\bar{c}C_m, \quad M_z = qSbC_n. \quad (5)$$

Qui $q = \frac{1}{2}\rho V^2$ é la pressione dinamica, S la superficie alare, b l'apertura alare e $\bar{c}$ la corda media aerodinamica. Le derivate dinamiche (C_{m_q} , C_{l_p} , C_{n_r}) e quelle di controllo ($C_{m_{\delta_e}}$, $C_{l_{\delta_a}}$, $C_{n_{\delta_r}}$) sono scalabili tramite parametri di `[flight_tuning]`.

Gli assi di riferimento per centro di gravit  e momenti d'inerzia sono definiti nella sezione `[weight_and_balance]`. In mancanza di valori sperimentali, il simulatore propone una stima degli MOI basata su geometria e peso a vuoto:

$$MOI_{\text{pitch}} \approx W_{\text{vuoto}} \frac{L^2}{810}, \quad MOI_{\text{roll}} \approx W_{\text{vuoto}} \frac{b^2}{1870}, \quad MOI_{\text{yaw}} \approx W_{\text{vuoto}} \frac{\left(\frac{1}{2}(L+b)\right)^2}{770}, \quad (6)$$

dove L é la lunghezza, b l'apertura e W_{vuoto} il peso a vuoto del velivolo.

Per i motori a turbina, i parametri sono definiti nelle sezioni:

`[generalenginedata]`, `[turbineenginedata]`.

Oltre alla spinta statica e alla geometria di installazione, viene introdotto il consumo specifico di carburante (TSFC).

Il modello assume:

$$\dot{m}_{\text{fuel}} = TSFC \cdot T, \quad (7)$$

dove $\dot{m}_{\text{fuel}}$ é il consumo in lb/h e T la spinta in lb. é possibile specificare valori separati di TSFC per l'afterburner, cos  come abilitare reverse thrust o angoli di montaggio della spinta che introducono momenti aggiuntivi.

3.3 Meccanismi di controllo

3.3.1 PID

Tra i diversi algoritmi di controllo sviluppati e applicati in ambito industriale, il controllore PID con la sua azione Proporzionale, Integrata e Derivativa rappresenta senza dubbio il piú diffuso e utilizzato. Il suo largo impiego deriva dal perfetto compromesso tra semplicitá di implementazione, robustezza e capacitá di adattarsi a una grande varietá di processi, dai sistemi meccanici a quelli termici, fino ai processi chimici e biologici. [6, 7]

Il principio di funzionamento del PID é piuttosto intuitivo. L'algoritmo riceve in ingresso un segnale di errore, cioé la differenza tra il valore desiderato di una variabile ovvero il riferimento e il valore effettivamente misurato dal sistema. Questo errore, indicato convenzionalmente con $e(t)$, viene elaborato dal controllore secondo tre modalitá di azione: proporzionale, integrale e derivativa. Il risultato di questa elaborazione é il segnale di controllo $u(t)$, che viene inviato ad un potenziale attuatore per guidare il sistema verso il comportamento desiderato. [8]

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt} \quad (8)$$

L'azione proporzionale é la piú immediata: il controllore produce un'uscita proporzionale all'errore istantaneo. Se l'errore é grande, anche la correzione sará grande; se l'errore é piccolo, la correzione sará piú contenuta. L'effetto pratico é quello di ridurre rapidamente lo scostamento dal riferimento. Tuttavia, un guadagno proporzionale eccessivo puó portare ad oscillazioni e persino instabilitá.

L'azione integrale guarda non solo all'errore istantaneo, ma anche alla sua storia. Integra infatti l'errore nel tempo, accumulandolo: in questo modo é in grado di correggere gli errori a regime, cioé gli scostamenti persistenti che il solo termine proporzionale non riesce ad annullare. Grazie a questo contributo, il sistema puó raggiungere con precisione il valore di riferimento. D'altra parte, un'azione integrale troppo aggressiva puó rendere la risposta lenta e oscillante.

L'azione derivativa, invece, ha il compito di anticipare il comportamento del sistema. Essa agisce in base alla velocitá di variazione dell'errore: se l'errore sta crescendo rapidamente, la derivata produce una correzione preventiva, come una sorta di "freno predittivo". Questo riduce oscillazioni e sovraelongazioni, migliorando la stabilitá. Tuttavia, il termine derivativo é molto sensibile al rumore di misura e deve essere utilizzato con cautela.

La combinazione di questi tre effetti - proporzionale, integrale e derivativo - dá origine al PID completo, un controllore versatile che riesce a garantire rapiditá, precisione e stabilitá allo stesso tempo.

Il controllore PID nasce in forma continua, ma per poter essere implementato in un calcolatore deve essere riscritto in maniera *discreta*, cioé definita a intervalli di tempo scanditi dal periodo di campionamento T_s . In questo modo, l'algoritmo puó essere eseguito passo dopo passo, elaborando i segnali misurati e producendo in uscita il comando da applicare al sistema.

L'idea di fondo resta la stessa: il controllore elabora l'*errore* tra il riferimento e l'uscita del sistema, indicato con $e[k]$ al passo k , e lo trasforma in un segnale di controllo $u[k]$ attraverso le tre azioni: proporzionale, integrale e derivativa.

La componente proporzionale calcola un contributo direttamente proporzionale all'errore istantaneo:

$$u_P[k] = K_P e[k] \quad (9)$$

La parte integrale tiene conto della *storia dell'errore*. In forma digitale, l'integrale viene sostituito da una somma che accumula gli errori campione dopo campione. Questo si traduce nella ricorrenza:

$$u_I[k] = u_I[k-1] + K_I T_s e[k] \quad (10)$$

Se l'errore persiste nel tempo, l'azione integrale cresce progressivamente, garantendo l'eliminazione dell'errore a regime.

La parte derivativa agisce prevedendo l'andamento dell'errore, reagendo alla sua variazione nel tempo. In forma discreta, questa operazione corrisponde a una *differenza finita*:

$$u_D[k] = K_D \frac{e[k] - e[k-1]}{T_s} \quad (11)$$

In questo modo il controllore riesce ad anticipare la dinamica del sistema, riducendo oscillazioni e migliorando la stabilità. Nelle applicazioni pratiche, questa formula viene spesso arricchita con un piccolo filtro, in modo da attenuare l'effetto del rumore di misura.

Mettendo insieme i tre contributi, si ottiene l'espressione diretta del controllore in forma discreta:

$$u[k] = K_P e[k] + K_I T_s \sum_{i=0}^k e[i] + K_D \frac{e[k] - e[k-1]}{T_s} \quad (12)$$

Questa é detta *forma posizionale*, perché fornisce direttamente il valore assoluto dell'uscita del controllore al passo k .

In alternativa, si può scrivere il PID in forma *incrementale*, concentrandosi non sul valore assoluto di $u[k]$, ma sulla variazione rispetto al passo precedente. In questo caso si calcola:

$$\Delta u[k] = K_P (e[k] - e[k-1]) + K_I T_s e[k] + \frac{K_D}{T_s} (e[k] - 2e[k-1] + e[k-2]) \quad (13)$$

e quindi:

$$u[k] = u[k-1] + \Delta u[k] \quad (14)$$

Questa formulazione ha il vantaggio di essere più robusta dal punto di vista numerico e di ridurre il rischio di accumulo di errori di calcolo. [5]

3.3.1.1 Antiwindup Uno dei problemi più comuni nell'uso del controllore PID riguarda il cosiddetto *integral windup*. Questo fenomeno si verifica quando l'attuatore del sistema raggiunge i propri limiti di saturazione: in tali condizioni, l'uscita del controllore non può più crescere o decrescere oltre un certo valore, mentre il termine integrale continua ad accumulare errore. Il risultato é che l'integratore mantiene un valore elevato anche quando l'errore tende a ridursi o cambia segno. La conseguenza pratica é una risposta lenta, caratterizzata da sovraelongazioni marcate e instabilità temporanee.

Per ovviare a questo problema, sono state sviluppate diverse strategie di *anti-windup*, che mirano a limitare o correggere il comportamento dell'integratore in presenza di saturazioni.

La tecnica piú semplice consiste nel **clamping** che implica il bloccaggio dell'integratore. Quando il segnale di controllo si trova in saturazione se l'errore mantiene lo stesso segno che quindi lo farebbe crescere ulteriormente, l'azione integrale cessa. In pratica, l'azione viene sospesa in condizioni critiche e riattivata non appena l'uscita torna entro i limiti consentiti. Questa strategia é di immediata implementazione, ma può introdurre discontinuitá evidenti nel passaggio tra le due modalitá operative. [9]

Una delle tecniche piú diffuse é la **back-calculation** che consiste in una retroazione di anti-windup. In questo caso si introduce un termine correttivo proporzionale alla differenza tra l'uscita calcolata dal controllore u e quella effettivamente applicata dopo la saturazione u_{sat} . Il modello matematico dell'integratore viene quindi modificato come segue:

$$\dot{x}_I = K_I e(t) + K_{aw} (u_{sat} - u) \quad (15)$$

dove K_{aw} é un guadagno di retroazione specifico per l'anti-windup. Questa soluzione consente di "scaricare" gradualmente l'integratore, riducendo gli effetti del windup in modo continuo e controllato. L'unico svantaggio é la necessitá di introdurre un parametro aggiuntivo (K_{aw}), che deve essere scelto e tarato con attenzione.

Un'altra strategia consiste nel **forcing** ovvero forzare lo stato dell'integratore affinché segua il valore dell'uscita effettivamente disponibile dopo la saturazione. In questo caso, il termine integrale non viene lasciato libero di crescere, ma viene riposizionato in modo da rispettare il vincolo imposto:

$$x_I(t) = u_{sat} - (u_P + u_D) \quad (16)$$

Questa tecnica é particolarmente efficace, poiché impedisce che l'integratore accumuli valori irrealistici, mantenendolo coerente con le condizioni operative del sistema.

Una soluzione ancora piú immediata é quella di **imporre limiti minimi e massimi** ai valori che l'integratore può assumere. In questo modo, si impedisce a priori che il termine integrale possa crescere oltre soglie prestabilite. Tuttavia, se i limiti sono troppo stringenti, esiste il rischio che l'integrale perda parte della sua efficacia nell'eliminare l'errore a regime.

Una variante del clamping prevede l'introduzione di una **zona morta**, all'interno della quale l'integratore é attivo, mentre al di fuori di essa viene bloccato. In questo modo si evita che errori troppo grandi (che porterebbero rapidamente in saturazione) alimentino l'integrale, pur mantenendo la capacitá di correzione quando il sistema si trova in prossimitá del regime desiderato.

3.3.2 Lead-Lag

Nel campo dell'ingegneria dei controlli, i filtri *lead-lag* rivestono un ruolo fondamentale per la modellazione e la regolazione dei sistemi dinamici. Essi rappresentano una famiglia di reti lineari tempo-invarianti che, in funzione della configurazione dei parametri, possono modificare la risposta in frequenza di un segnale amplificando o attenuando determinate componenti. [10] Questi filtri sono strettamente collegati ai piú noti filtri *passa-basso* e *passa-alto*. La loro combinazione permette di modellare comportamenti intermedi e di ottenere la cosiddetta funzione di trasferimento lead-lag, che unisce entrambe le azioni.

Un filtro **lag** introduce un ritardo di fase: attenua le componenti ad alta frequenza, lasciando pressoché inalterate quelle a bassa frequenza. é quindi simile a un filtro passa-basso del primo ordine.

Un filtro **lead**, al contrario, anticipa la risposta, introducendo un guadagno di fase positivo alle frequenze medio-alte. Ciò equivale, dal punto di vista funzionale, a un filtro passa-alto del primo ordine, con una pendenza limitata.

Il filtro **lead-lag** combina entrambi i comportamenti: consente di spostare la fase e modellare il guadagno di un sistema in funzione della frequenza, rendendolo uno strumento essenziale sia per la compensazione di dinamiche indesiderate sia per il miglioramento della stabilità e delle prestazioni dei controllori.

Nel dominio di Laplace, la funzione di trasferimento di un filtro lead-lag si può scrivere come:

$$H(s) = K \cdot \frac{T_{\text{lead}}s + 1}{T_{\text{lag}}s + 1} \quad (17)$$

dove:

- K é il guadagno statico,
- T_{lead} é la costante di tempo associata alla parte anticipatrice (lead),
- T_{lag} é la costante di tempo associata alla parte ritardatrice (lag).

Se $T_{\text{lead}} = 0$, si ottiene:

$$H_{\text{lag}}(s) = \frac{1}{T_{\text{lag}}s + 1} \quad (18)$$

che corrisponde a un filtro passa-basso del primo ordine.

Se invece $T_{\text{lag}} \rightarrow 0$, si ottiene:

$$H_{\text{lead}}(s) = T_{\text{lead}}s + 1 \quad (19)$$

che rappresenta un filtro con caratteristiche di passa-alto.

Quando il filtro deve essere implementato in un calcolatore, occorre passare dalla forma continua a quella discreta. La discretizzazione può avvenire tramite diversi metodi: tra i più comuni troviamo l'approssimazione di **Eulero all'indietro** (Backward Euler) o la trasformata bilineare (Tustin).

Dato il sistema continuo:

$$H(s) = \frac{1}{T_f s + 1} \quad (20)$$

si ottiene la ricorrenza discreta:

$$y[k] = (1 - a)y[k - 1] + a u[k] \quad (21)$$

con

$$a = \frac{T_s}{T_f + T_s} \quad (22)$$

dove T_s é il tempo di campionamento.

Invece applicando la sostituzione di Tustin al filtro lead lag nella sua interezza

$$s \approx \frac{2}{T_s} \frac{z - 1}{z + 1} \quad (23)$$

si ottiene:

$$H(z) = K \cdot \frac{\left(\frac{2T_{\text{lead}}}{T_s} + 1\right)z + \left(1 - \frac{2T_{\text{lead}}}{T_s}\right)}{\left(\frac{2T_{\text{lag}}}{T_s} + 1\right)z + \left(1 - \frac{2T_{\text{lag}}}{T_s}\right)} \quad (24)$$

Questa forma consente di implementare direttamente il filtro in digitale, utilizzando ricorrenze del tipo:

$$y[k] = b_0 u[k] + b_1 u[k - 1] - a_1 y[k - 1] \quad (25)$$

con coefficienti a_1, b_0, b_1 derivati dai parametri del filtro. [5]

3.4 Approccio Leader-Follower

Uno dei modelli piú classici e intuitivi per il volo in formazione é quello detto **Leader-Follower**. In questo approccio, il comportamento dell'intera formazione é determinato principalmente dalla traiettoria di uno o piú **leader**, mentre i droni rimanenti, detti **follower**, regolano la propria dinamica in funzione della posizione e del moto del leader.

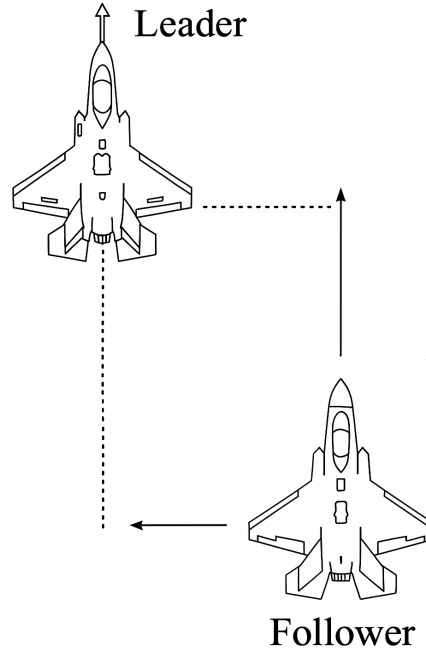


Figura 6: Approccio Leader follower - piano XY.

Matematicamente, se indichiamo con $x_L(t) \in \mathbb{R}^3$ la posizione del leader in funzione del tempo, la traiettoria desiderata di un follower i é data da:

$$x_i^{des}(t) = x_L(t) + \Delta_i \quad (26)$$

dove $\Delta_i \in \mathbb{R}^3$ é il vettore che definisce la posizione relativa desiderata rispetto al leader. In altre parole, il follower non segue una traiettoria assoluta, ma mantiene un “offset” fisso dal leader, che può rappresentare una distanza di sicurezza, una disposizione a griglia o una formazione geometrica specifica.

Per ottenere questa traiettoria, il controllore del follower calcola l’errore di inseguimento:

$$e_i(t) = x_i^{des}(t) - x_i(t) \quad (27)$$

dove $x_i(t)$ é la posizione reale del follower. L’obiettivo del controllore é minimizzare l’errore $e_i(t)$ mediante leggi di controllo che agiscono sulle velocità o accelerazioni.

Il modello Leader-Follower trova applicazione diretta nei tre piani fondamentali dello spazio tridimensionale:

Piano Orizzontale (XY): In questo piano il leader definisce la rotta laterale. Supponiamo che il leader percorra una traiettoria parametrica

$$x_L(t) = \begin{bmatrix} x_L(t) \\ y_L(t) \end{bmatrix}. \quad (28)$$

Allora, per un follower disposto lateralmente, la posizione desiderata é:

$$\begin{bmatrix} x_i^{des}(t) \\ y_i^{des}(t) \end{bmatrix} = \begin{bmatrix} x_L(t) \\ y_L(t) \end{bmatrix} + \begin{bmatrix} \Delta_{i,x} \\ \Delta_{i,y} \end{bmatrix}. \quad (29)$$

Questo consente di creare formazioni a linea, a griglia o a matrice, mantenendo la stessa distanza orizzontale tra droni anche durante curve o cambi di direzione.

Piano Verticale (XZ): Qui si regola l’altitudine. Se il leader varia la propria quota secondo $z_L(t)$, ogni follower mantiene un dislivello costante:

$$z_i^{des}(t) = z_L(t) + \Delta_{i,z}. \quad (30)$$

Questo permette di organizzare formazioni multilivello, ad esempio con UAV a diverse altezze per effettuare misure stratificate o ottimizzare la copertura sensoriale.

Piano Longitudinale (YZ): In direzione del moto, i follower devono mantenere una distanza longitudinale costante per evitare collisioni. Se il leader accelera o decelera, la legge di controllo del follower include una compensazione sulla velocità.

Il modello Leader-Follower ha il vantaggio di essere concettualmente semplice e facilmente implementabile. Ogni follower deve solo stimare la posizione e velocità del leader o, in versioni più complesse, del drone che lo precede nella catena. Questo riduce notevolmente la complessità computazionale e di comunicazione rispetto a metodi più sofisticati come la formazione virtuale.

Tuttavia, l’approccio soffre di due limiti fondamentali. Se il leader fallisce o perde il controllo, l’intera formazione viene compromessa. Nei sistemi a catena lunga, piccoli errori di inseguimento possono amplificarsi man mano che ci si allontana dal leader, causando instabilità nella parte finale della formazione.

Per mitigare questi problemi, in letteratura sono stati sviluppati modelli con più leader o con architetture ibride come Leader-Follower combinato con controllo distribuito che migliorano robustezza e resilienza.

4 Autopilota di formazione

4.1 Velivolo

Per lo sviluppo di questo progetto é stato utilizzato sia come velivolo mothership che come drone gregario l'F-35 Lightning II. Si tratta di un caccia multiruolo di quinta generazione progettato per integrare in un unico sistema caratteristiche di furtività, superiorità aerea, capacità d'attacco e funzioni di supporto elettronico. A differenza dei velivoli delle generazioni precedenti, che erano concepiti essenzialmente come mezzi di combattimento con funzioni ben definite, l'F-35 é stato ideato come un vero e proprio nodo informativo capace di raccogliere, elaborare e redistribuire dati in tempo reale, rendendolo un attore centrale nelle moderne operazioni multi-dominio.

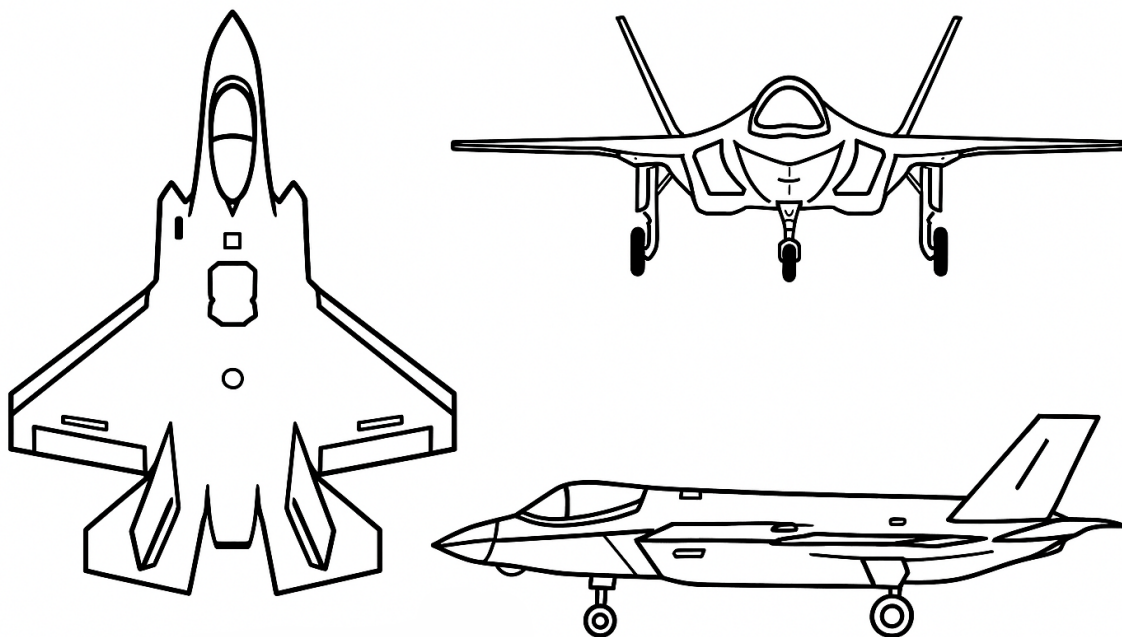


Figura 7: F35 -Disegno 3 viste.

Dal punto di vista tecnologico, l'F-35 incorpora numerose innovazioni che lo rendono unico. La sua architettura stealth, ottenuta grazie a materiali radar-assorbenti e a una progettazione delle superfici aerodinamiche volta a ridurre la sezione radar equivalente, gli consente di operare in spazi aerei altamente difesi. Ma ciò che davvero lo distingue é l'insieme dei suoi sistemi di sensori e la capacità di fusione dei dati. Il radar AESA a scansione elettronica, i sensori a infrarossi distribuiti (DAS), il sistema elettro-ottico EOTS e i sistemi di guerra elettronica producono enormi quantità di informazioni che vengono elaborate da algoritmi avanzati per fornire al pilota una rappresentazione sintetica, unificata e coerente dello scenario operativo. Questo processo, noto come *sensor fusion*, riduce notevolmente il carico cognitivo del pilota e aumenta la situational awareness.

Oltre a ciò, l'F-35 dispone di avanzati datalink come il MADL (Multifunction Advanced Datalink), che permettono di condividere in tempo reale le informazioni raccolte con altre piattaforme aeree, navali e terrestri, anche in condizioni ostili e con bassa probabilità di intercettazione. Questa capacità lo trasforma non soltanto in un aereo da combattimento, ma in un vero e proprio coordinatore di missione, capace di guidare altri assetti meno sofisticati

e di integrarsi in reti operative distribuite. Un elemento fondamentale è anche l'architettura software del velivolo, pensata per essere modulare e aggiornabile. A differenza dei velivoli tradizionali, che richiedevano modifiche hardware per introdurre nuove funzioni, l'F-35 può ricevere aggiornamenti digitali periodici che gli permettono di incorporare rapidamente algoritmi sempre più sofisticati. Questo lo rende una piattaforma adatta a sperimentare ed evolvere progressivamente verso capacità autonome.

La possibilità di sviluppare autonomia a bordo di un sistema come l'F-35 trova giustificazione in diverse caratteristiche intrinseche. In primo luogo, la fusione sensoriale costituisce già la base per qualsiasi forma di intelligenza artificiale: un sistema autonomo deve infatti poter contare su una percezione affidabile e completa dell'ambiente circostante. L'F-35, grazie all'integrazione di sensori eterogenei e all'elaborazione in tempo reale, è già in grado di fornire una visione globale e coerente del campo di battaglia, riducendo una delle principali difficoltà nello sviluppo dell'autonomia.

In secondo luogo, la sua potenza di calcolo e l'architettura avionica avanzata permettono di sostenere carichi computazionali elevati. Algoritmi di apprendimento automatico, modelli predittivi per il comportamento dei nemici e logiche decisionali distribuite possono essere integrati senza necessità di riprogettazioni radicali. Ciò significa che l'autonomia può essere sviluppata in modo incrementale, partendo da funzioni di supporto decisionale fino ad arrivare a capacità di missione completamente automatizzate.

Un altro elemento chiave è la riduzione del carico cognitivo del pilota. Già oggi, l'F-35 si pone come interfaccia avanzata uomo-macchina, alleggerendo il pilota da compiti di basso livello e permettendogli di concentrarsi su decisioni tattiche di alto livello. L'autonomia potrebbe spingersi oltre, arrivando a sostituire il pilota in missioni particolarmente rischiose o a supportarlo con suggerimenti decisionali calcolati in millisecondi, tempi impossibili da raggiungere per la mente umana in scenari di combattimento complessi.

Le capacità di rete dell'F-35, inoltre, rendono naturale la sua evoluzione verso il ruolo di coordinatore di sistemi autonomi. Esso potrebbe agire come piattaforma madre (*mothership*) in grado di controllare sciame di droni collaborativi, delegando a essi compiti specifici come la ricognizione, la guerra elettronica o l'attacco a bersagli multipli, mantenendo per sé la supervisione strategica. In un contesto multi-dominio, l'autonomia permetterebbe all'F-35 non solo di condurre missioni aeree, ma di integrarsi direttamente con assetti navali e terrestri, diventando un nodo intelligente di un'unica rete bellica distribuita.

Dal punto di vista della sicurezza e dell'affidabilità, il velivolo possiede già sistemi ridondanti e protocolli di gestione dei guasti che possono fungere da garanzia per l'introduzione dell'autonomia. Modalità di fallback e supervisione remota potrebbero essere implementate senza dover modificare radicalmente la struttura del sistema. Questo riduce uno degli ostacoli principali all'adozione dell'autonomia in aviazione militare: la necessità di assicurare affidabilità e prevedibilità in ogni scenario operativo.

In termini operativi, rendere l'F-35 autonomo significherebbe ottenere diversi vantaggi. Missioni ad altissimo rischio, come la penetrazione in spazi aerei pesantemente difesi, potrebbero essere svolte senza rischiare la vita dei piloti. Le operazioni potrebbero durare più a lungo, senza i limiti fisiologici dell'uomo. Sciame di droni potrebbero essere gestiti direttamente dal velivolo, moltiplicandone l'efficacia e la capacità di proiezione. La superiorità informativa, infine, sarebbe amplificata, poiché un sistema autonomo può analizzare e reagire più velocemente di un operatore umano a scenari mutevoli.

4.2 Struttura del codice

Il progetto consiste in un add-on sviluppato per il simulatore Prepar3D, che utilizza l'interfaccia *SimConnect* per pilotare automaticamente un velivolo designato come *C2 Plane* e un gregario a controllo automatico chiamato *Drone*. L'obiettivo principale del codice é la gestione autonoma delle fasi di missione, dall'inizializzazione dei velivoli a terra fino alla crociera, tramite una serie di autopiloti implementati manualmente. Parallelamente, l'add-on fornisce funzionalità di logging e test automatici dei regolatori, con lo scopo di valutare le prestazioni dei controlli e permettere successiva analisi.

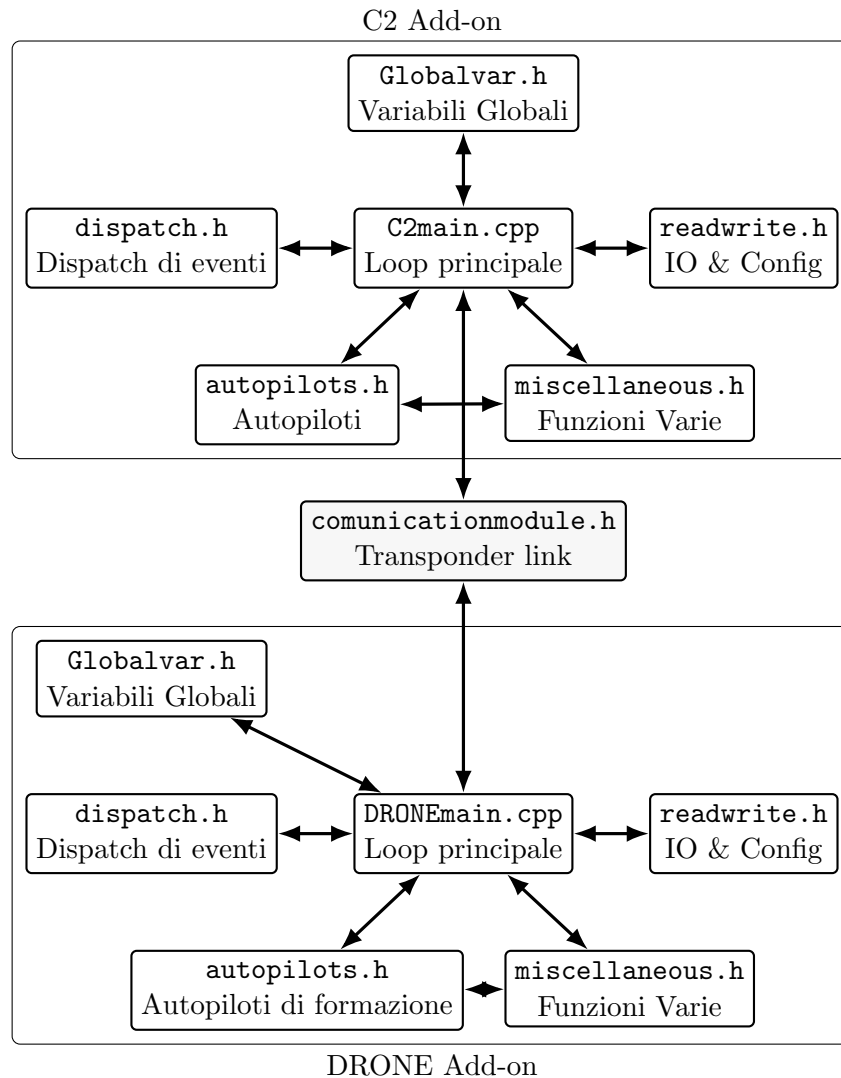


Figura 8: Schema a blocchi del codice

Il codice é stato progettato in maniera modulare, organizzato in una serie di file `.h` inclusi da un unico `main.cpp` per velivolo, che funge da cuore esecutivo del sistema. Tale struttura rende l'architettura facilmente estendibile, ma introduce anche un livello di complessità dovuto all'utilizzo di numerose variabili globali.

La struttura del progetto é composta da diversi moduli, ciascuno con un ruolo specifico. Le *variabili globali*, le strutture dati e i flag di stato sono racchiusi nel file `globalvar.h`. In esso

troviamo, ad esempio, la definizione di *ObjectDataStruct*, che contiene lo stato dinamico e aerodinamico del velivolo, insieme a due istanze specifiche denominate *UserPlane* e *OtherPlane*. Questi oggetti fungono da interfaccia tra simulatore e add-on, veicolando posizione, assetto, velocità e deflessioni dei comandi di ciascun velivolo.

Il file *dispatchfun.h* è invece deputato alla gestione degli eventi di SimConnect. All'interno di questo modulo è implementata la funzione di callback *MyDispatchProc*, che riceve ed elabora i pacchetti informativi provenienti dal simulatore, aggiornando lo stato interno dell'add-on. Contestualmente, la stessa funzione si occupa di definire i canali di input-output: associa i tasti della tastiera agli eventi SimConnect, mappa gli eventi di controllo e organizza il flusso informativo dal simulatore al programma.

Un ruolo cruciale è ricoperto dal file *autopilots.h*, in cui risiedono gli omonimi autopiloti. Questi sono implementati come sistemi Lead-Lag, PID e ibridi, dotati di meccanismi di anti-windup, con saturazioni esplicite per evitare comandi irrealizzabili. I regolatori sono organizzati gerarchicamente: quelli esterni, come il controllo di quota e di rotta, forniscono set-point ai regolatori interni, responsabili invece di variabili dinamicamente più rapide, quali pitch e bank. La separazione di banda garantisce stabilità e semplicità di taratura.

Per quanto riguarda le funzioni di servizio, il file *miscellaneous.h* raccoglie utility di vario tipo: il riconoscimento dello stato *on-ground*, la gestione automatica del carrello e soprattutto le funzioni di trasformazione tra sistemi di coordinate. In particolare, il passaggio da coordinate geodetiche a frame ECEF (Earth-Centered, Earth-Fixed) e poi a frame locali NED (North-East-Down) e body è essenziale per calcolare separazioni relative tra velivoli.

La gestione dell'I/O, distinta dal resto del codice, è centralizzata nel file *readwrite.h*. Questo modulo si occupa di leggere i file di configurazione (*inputs/configs.txt*), di inizializzare i file di output per il logging e di chiedere all'utente, tramite prompt a console, se abilitare funzionalità come test o registrazione dati. I file di output generati riportano, per ciascun passo temporale, l'intero stato del velivolo.

Infine, *communicationmodule.h* fornisce un canale di comunicazione rudimentale con il drone gregario attraverso l'uso del transponder code. Sono definiti tre comandi simbolici: *TakeOff*, *Reach C2* e *Formation*, che corrispondono rispettivamente ai valori 1, 2 e 3 del transponder.

4.3 Gestione della missione

Il cuore logico della missione è una macchina a stati governata dal flag *flag_decollo*. Inizialmente, quando il velivolo è a terra, viene memorizzato l'heading iniziale. Se la velocità supera un certo valore soglia, l'add-on assume che il velivolo sia già in fase di rullaggio e salta alcune fasi iniziali.

La sequenza di missione procede in maniera deterministica. Nella prima fase, il velivolo viene spinto al 90% di potenza, i freni vengono rilasciati e viene inviato un comando di decollo al drone che a questo punto effettua la stessa manovra. Successivamente, in fase di rullaggio, il timone viene comandato per allineare l'heading, mentre gli elevatori mantengono l'aereo a terra fino a che la velocità di stacco non viene raggiunta. Nella fase di salita, il controllo passa a un regolatore di pitch che forza l'assetto a valori negativi per ottenere una salita stabilizzata, mentre gli alettoni mantengono l'allineamento di rotta. Una volta raggiunta circa un terzo della quota di crociera, l'aereo entra nella fase di immissione. Qui i regolatori di quota e rotta stabilizzano il velivolo ai valori desiderati. Se la quota resta stabile entro un margine di ± 10 piedi per almeno cinque secondi, il sistema dichiara il raggiungimento del C2 in crociera e invia un nuovo comando di stato al drone per il raggiungimento. Infine, in crociera, l'autothrottle

mantiene una velocità costante di 850 ft/s, mentre gli altri regolatori tengono invariati assetto e direzione.

Parallelamente, il carrello viene retratto automaticamente al superamento di una velocità pre-stabilita, senza richiedere input manuali. Questa sequenza logica, seppur semplice, consente al velivolo di compiere un profilo di volo standardizzato e ripetibile.

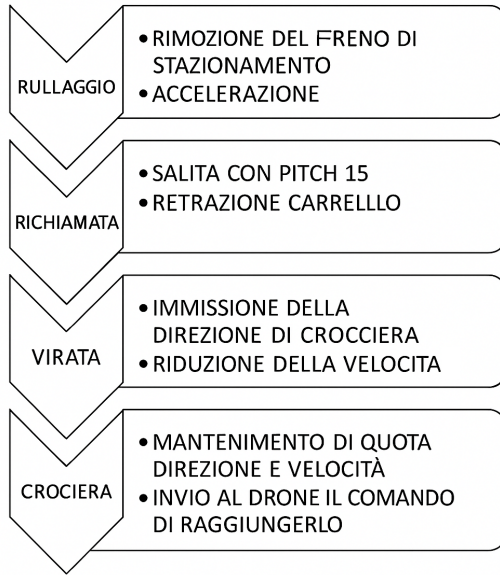


Figura 9: Fasi di missione del leader.

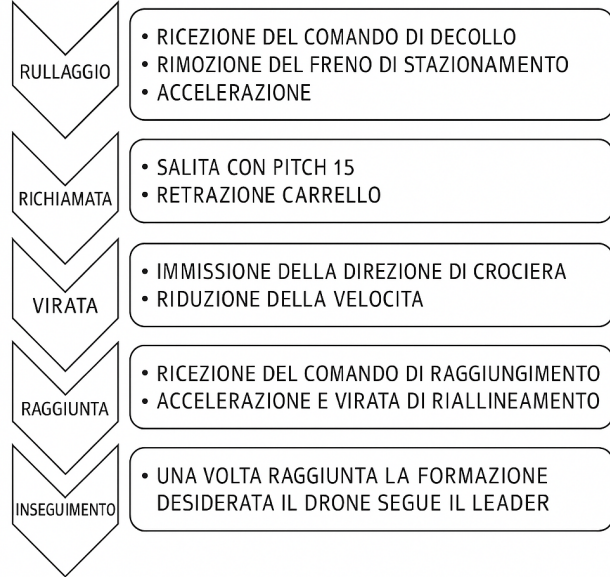


Figura 10: Fasi di missione del follower.

4.4 Trasformazioni di coordinate

Il calcolo delle separazioni tra velivoli é basato su una pipeline di trasformazioni coordinate che garantisce consistenza geometrica rispetto alla superficie ellissoidale terrestre. A partire dalle posizioni geodetiche (latitudine ϕ , longitudine λ , quota h), si effettua la conversione in coordinate ECEF mediante i parametri WGS-84. L'eccentricità dell'ellissoide e il raggio di curvatura nel primo verticale $N(\phi)$ sono definiti come:

$$e^2 = 1 - \frac{b^2}{a^2}, \quad N(\phi) = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}, \quad (31)$$

$$\mathbf{r}_{\text{ecef}}(\phi, \lambda, h) = \begin{bmatrix} (N(\phi) + h) \cos \phi \cos \lambda \\ (N(\phi) + h) \cos \phi \sin \lambda \\ (N(\phi)(1 - e^2) + h) \sin \phi \end{bmatrix}, \quad (32)$$

dove a e b sono rispettivamente semiasse maggiore e minore. La differenza tra i vettori ECEF del drone e del velivolo di riferimento fornisce il vettore relativo $\Delta \mathbf{r}_{\text{ecef}}$:

$$\Delta \mathbf{r}_{\text{ecef}} = \mathbf{r}_{\text{ecef}}(\phi, \lambda, h) - \mathbf{r}_{\text{ecef}}(\phi_0, \lambda_0, h_0). \quad (33)$$

Tale differenza viene riportata nel frame locale NED del velivolo di riferimento attraverso la matrice di rototrasformazione:

$$C_e^m(\phi_0, \lambda_0) = \begin{bmatrix} -\sin \phi_0 \cos \lambda_0 & -\sin \phi_0 \sin \lambda_0 & \cos \phi_0 \\ -\sin \lambda_0 & \cos \lambda_0 & 0 \\ -\cos \phi_0 \cos \lambda_0 & -\cos \phi_0 \sin \lambda_0 & -\sin \phi_0 \end{bmatrix}, \quad (34)$$

$$\mathbf{r}_n = C_e^m(\phi_0, \lambda_0) \Delta \mathbf{r}_{\text{ecef}}. \quad (35)$$

Per rappresentare le separazioni nel frame di corpo del leader, utile al calcolo di distanze frontali, laterali e verticali, il vettore NED viene ulteriormente trasformato mediante le rotazioni di assetto (heading ψ , pitch θ , roll φ). Le matrici elementari sono:

$$R_z(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}, \quad R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix}. \quad (36)$$

Nel codice implementato la trasformazione segue l'ordine *heading*, *pitch*, *roll* con segno negativo sull'heading, risultando:

$$\mathbf{r}_b = R_x(\varphi) R_y(\theta) R_z(-\psi) \mathbf{r}_n, \quad (37)$$

mentre in casi particolari é disponibile anche la sola rotazione di heading:

$$\mathbf{r}_{n,\psi} = R_z(-\psi) \mathbf{r}_n. \quad (38)$$

Questa catena di trasformazioni, coerente con le funzioni `EcefToBody` ed `EcefToNEDPhi` implementate nel codice, consente di esprimere le separazioni nel sistema di corpo del leader, mantenendo la correttezza geometrica e tenendo conto della curvatura terrestre. Tale soluzione, pur con approssimazioni intrinseche, risulta adeguata agli scenari simulativi previsti.

4.5 Acquisizione dati

Oltre al profilo missione standard, il codice prevede una modalità di test dei regolatori. L'utente può scegliere di eseguire prove specifiche, come il controllo del pitch o della velocità. In tali casi, vengono generati set-point tempo-varianti, che costringono i regolatori a reagire a variazioni brusche. Questo permette di analizzare la dinamica di risposta e valutare l'efficacia dei guadagni.

```
Connected to Prepar3D
Config file correctly loaded.
Vuoi eseguire un test dei controlli?? (Y/N): y
Su cosa vuoi eseguire il test?
[1] PITCH
[2] BANK
[3] ALTITUDE
[4] HEADING
[5] VELOCITY
[6] NORTH
[7] EAST
[8] DOWN
8
Avviamento test autopilota di mantenimento della posizione relativa DOWN
Vuoi salvare i dati della simulazione? (Y/N): y
Creazione del file output: outputs/C2_2025-04-07_12-39-30.txt
Creazione del file output: outputs/C2_2025-04-07_12-39-30.txt
Inizializzazione
Initialization completata
Simulatore in Pausa
```

Figura 11: Schermata principale addon

La funzionalità di logging affianca l'esecuzione dei test e delle missioni. Vengono creati file di testo distinti per il C2 e per il drone, con intestazioni descrittive e campi che riportano lo stato completo a ogni campione temporale. I file vengono nominati in maniera univoca attraverso un timestamp, così da evitare sovrascritture e facilitare l'archiviazione.

Un aspetto centrale del progetto è la modalità con cui l'addon interagisce con il simulatore Prepar3D per acquisire i dati necessari al funzionamento degli autopiloti e della logica di missione. L'interfaccia scelta è SimConnect, che rappresenta il canale ufficiale di comunicazione tra applicazioni esterne e l'ambiente simulativo. Attraverso SimConnect, l'addon dichiara esplicitamente quali grandezze desidera ricevere, stabilendo così una sorta di contratto con il simulatore. Questo processo inizia nella fase di inizializzazione, quando la funzione di callback `MyDispatchProc`, contenuta nel file `dispatchfun.h`, definisce le variabili da monitorare. In questa fase vengono registrate le cosiddette *data definitions*, cioè gli insiemi di parametri che il simulatore dovrà fornire ad ogni aggiornamento.

Il ciclo operativo dell'acquisizione prevede tre momenti fondamentali. In primo luogo, il programma richiede i dati sul velivolo controllato dall'utente, il cosiddetto *C2 Plane*, attraverso la funzione `SimConnect_RequestDataOnSimObject`. In parallelo, la stessa funzione viene invocata per ottenere le informazioni relative ad altri aeromobili presenti nello scenario, come ad esempio un drone o il traffico circostante. Successivamente, i dati viaggiano dal simulatore al programma in maniera asincrona e giungono alla funzione di callback precedentemente

registrata. Qui vengono interpretati e ricondotti alla struttura `ObjectDataStruct`, che costituisce il contenitore principale dei dati aeronautici all'interno dell'addon. Le due istanze `UserPlane` e `OtherPlane` vengono così aggiornate a ogni ciclo con i valori più recenti. Infine, il flusso acquisitivo si completa con la frequenza di campionamento stabilita: il programma inserisce una pausa di 50 millisecondi all'interno del ciclo principale, ottenendo un ritmo di acquisizione e aggiornamento di circa 20 Hertz. Questo valore rappresenta un compromesso efficace tra prontezza di risposta e carico computazionale.

L'insieme delle variabili acquisite è molto ampio e permette di ricostruire uno stato quasi completo del velivolo. I dati più immediatamente intuitivi sono quelli relativi alla cinematica e alla posizione: latitudine, longitudine e altitudine forniscono il riferimento geografico, mentre gli angoli di assetto, espressi come pitch, bank e heading, descrivono l'orientamento. A ciò si aggiungono le velocità, sia come modulo rispetto al suolo, sia nelle componenti lungo gli assi del velivolo. Non mancano le accelerazioni lineari e i ratei angolari, elementi indispensabili per una descrizione dinamica più raffinata.

Un altro gruppo di variabili riguarda i comandi e le superfici mobili. L'addon riceve dal simulatore la posizione della manetta, la deflessione di timone, alettoni ed elevatori, oltre allo stato di flap, carrello e freno di parcheggio. Queste informazioni permettono non solo di monitorare lo stato effettivo dei comandi, ma anche di chiudere i loop di controllo quando i regolatori inviano i loro segnali correttivi.

Sono inoltre acquisiti dati aerodinamici e ambientali, come i coefficienti di portanza, resistenza e momento, le forze aerodinamiche nel sistema di riferimento del velivolo e le condizioni del vento (velocità e direzione). Tali variabili forniscono un quadro completo delle interazioni tra velivolo e ambiente, utile sia per la valutazione delle prestazioni sia per la robustezza dei controllori. Infine, il simulatore comunica anche variabili di stato globale, tra cui un flag che indica se il velivolo si trova a terra, il codice transponder utilizzato come canale di comunicazione e il tempo di simulazione, che rappresenta il riferimento temporale per il calcolo delle derivate e per l'esecuzione dei test.

4.6 Controllori

Nel progetto sono implementate una serie di autopiloti, ciascuno progettato per controllare una specifica variabile del volo. Tutti i regolatori utilizzano strutture Lead-Lag, PID o ibride, arricchite da meccanismi di anti-windup e saturazione per tenere conto dei limiti fisici dei comandi. Nei paragrafi seguenti vengono descritti i diversi autopiloti con le relative leggi matematiche.

4.6.1 Pitch Hold

Il controllore di *Pitch Hold* presente in Figure 12 ha l'obiettivo di stabilizzare l'assetto longitudinale del velivolo agendo sull'equilibratore. L'implementazione segue uno schema a due stadi di tipo *lead-lag*, composta da un blocco *lead* che processando l'errore fornisce un pitch rate di riferimento e da un blocco *lag* che filtra il segnale, evitando oscillazioni troppo rapide. Il primo stadio (*lead*) trasforma l'errore di pitch in un rateo di beccheggio desiderato, rendendo la risposta pronta ai cambiamenti rapidi; il secondo stadio (*lag*) filtra il segnale di controllo per smussare le componenti ad alta frequenza ed evitare comandi troppo bruschi. L'uscita viene infine saturata entro i limiti fisici accettati dal simulatore.

Il punto di partenza é il calcolo dell'errore di pitch come differenza tra riferimento e misura:

$$e_\theta(t) = \theta_{\text{cmd}}(t) - \theta(t), \quad (39)$$

dove $\theta_{\text{cmd}}(t)$ é il pitch desiderato e $\theta(t)$ il pitch misurato.

Lo stadio *lead* costruisce un *rateo di pitch* comandato combinando l'errore e la sua derivata numerica:

$$\dot{\theta}_{\text{cmd}}(t) = K_\ell \left(e_\theta(t) + \tau_d \frac{e_\theta(t) - e_\theta(t - \Delta t)}{\Delta t} \right), \quad (40)$$

con $K_\ell = 9000$ e $\tau_d = 0.3$. Il termine derivativo anticipa la dinamica e incrementa la prontezza del controllore, mentre Δt é il passo di campionamento stimato dalla simulazione.

Il rateo di pitch desiderato viene confrontato con il rateo effettivamente misurato (variabile $q(t)$). Nel codice questa misura é scalata da un guadagno di retroazione K_q :

$$e_q(t) = \dot{\theta}_{\text{cmd}}(t) - K_q q(t), \quad K_q = 4500. \quad (41)$$

Il segnale $e_q(t)$ attraversa quindi un filtro *lag* del primo ordine, implementato come media esponenziale dipendente da Δt :

$$u(t) = \alpha e_q(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0.8}, \quad (42)$$

dove $u(t)$ rappresenta il comando grezzo agli elevatori. La scelta di α rende il filtro adattivo rispetto al passo temporale effettivo del simulatore, favorendo una risposta regolare anche in presenza di jitter temporale.

Infine, il comando viene limitato ai valori massimi consentiti dall'interfaccia di SimConnect.

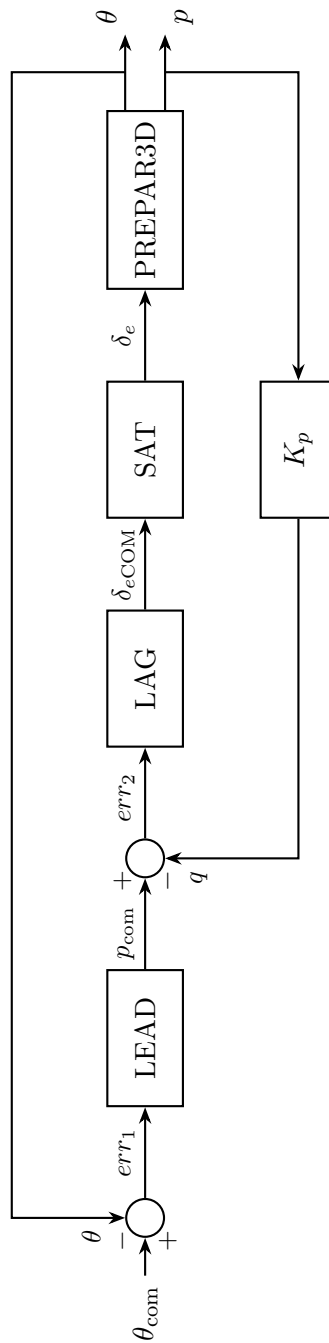


Figura 12: Schema a blocchi del pitch hold

4.6.2 Bank Hold

Il *Bank Hold* presente in Figure 13ha il compito di stabilizzare l'angolo di rollio del velivolo, generando un comando sugli alettoni che permetta di seguire l'angolo desiderato. L'implementazione utilizza una struttura a due stadi, composta da un blocco *lead* che processando l'errore fornisce un roll rate di riferimento e da un blocco *lag* che filtra il segnale, evitando oscillazioni troppo rapide. L'uscita finale viene infine saturata ai limiti fisici consentiti dal simulatore.

Il punto di partenza é il calcolo dell'errore:

$$e_\phi(t) = \phi_{\text{cmd}}(t) - \phi(t) \quad (43)$$

dove $\phi_{\text{cmd}}(t)$ rappresenta l'angolo desiderato e $\phi(t)$ quello misurato.

Questo errore viene utilizzato per calcolare un *roll rate* comandato attraverso un controllore di tipo lead, che combina l'errore con la sua derivata:

$$\dot{\phi}_{\text{cmd}}(t) = K_\ell \left(e_\phi(t) + \tau_d \frac{e_\phi(t) - e_\phi(t - \Delta t)}{\Delta t} \right) \quad (44)$$

dove $K_\ell = 400$ é il guadagno principale del blocco, $\tau_d = 0.64$ é la costante di anticipo e Δt rappresenta l'intervallo temporale tra due campioni consecutivi. Questa forma di compensazione anticipatrice rende il controllore piú reattivo ai cambiamenti rapidi dell'angolo di rollio.

Il valore ottenuto viene confrontato con il roll rate effettivamente misurato, ponderato da un fattore di scala:

$$e_{\dot{\phi}}(t) = \dot{\phi}_{\text{cmd}}(t) - K_r \dot{\phi}(t) \quad (45)$$

dove $\dot{\phi}(t)$ é il roll rate reale e $K_r = 200$ é un guadagno di retroazione.

Il segnale $e_{\dot{\phi}}(t)$ passa quindi attraverso un filtro di tipo lag, che smussa le variazioni veloci e riduce il rumore:

$$u(t) = \alpha e_{\dot{\phi}}(t) + (1 - \alpha) u(t - \Delta t) \quad (46)$$

con

$$\alpha = \frac{\Delta t}{\Delta t + 0.7} \quad (47)$$

dove $u(t)$ é il comando agli alettoni e $u(t - \Delta t)$ é il valore del comando al passo temporale precedente. L'uso di α rende il filtro dipendente dalla frequenza di campionamento e quindi adattivo rispetto al passo temporale effettivo della simulazione.

Infine, il comando agli alettoni é limitato attraverso una saturazione, cosí da rientrare nell'intervallo gestibile dal simulatore.

Dove θ é l'angolo di pitch, θ_{ref} il riferimento, q_{cmd} il rateo di pitch desiderato, $f(\cdot)$ un filtro passa-basso, u_θ il comando agli elevatori.

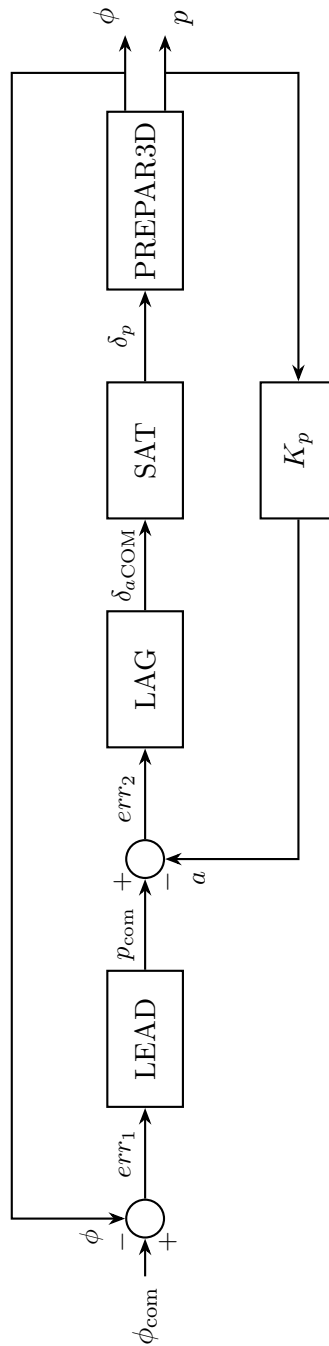


Figura 13: Schema a blocchi del bank hold

4.6.3 Autothrottle

L'*Autothrottle* presente in Figure 14 ha lo scopo di regolare la posizione della manetta per inseguire una velocità desiderata. L'implementazione segue lo schema a due stadi *lead-lag* già impiegato per i canali di assetto, con una particolarità: il blocco *lag* risulta, nei parametri correnti, degenerare in un semplice passaggio diretto (di fatto senza filtraggio), come si vedrà più avanti.

Il segnale di ingresso è l'errore di velocità

$$e_V(t) = V_{\text{cmd}}(t) - V(t), \quad (48)$$

dove $V_{\text{cmd}}(t)$ è la velocità desiderata e $V(t)$ la velocità misurata. Questo errore viene trasformato, nello stadio *lead*, in un comando di manetta che combina una parte proporzionale e una parte derivativa numerica dell'errore:

$$u_{\text{cmd}}(t) = K_\ell \left(e_V(t) + \tau_d \frac{e_V(t) - e_V(t - \Delta t)}{\Delta t} \right), \quad K_\ell = 3000, \quad \tau_d = 0.27, \quad (49)$$

con Δt il passo temporale calcolato dalla simulazione. Il termine proporzionale assicura che la manetta cresca all'aumentare dell'errore di velocità, mentre il termine derivativo anticipa i cambi rapidi dell'errore, rendendo il controllore più pronto.

Per attenuare eventuali sovraelongazioni dovute alla dinamica propulsiva e all'inerzia del velivolo, il comando $u_{\text{cmd}}(t)$ viene corretto tramite un confronto con l'accelerazione longitudinale $a(t)$ misurata dal simulatore:

$$e_u(t) = u_{\text{cmd}}(t) - K_a a(t), \quad K_a = 1500. \quad (50)$$

In questo modo, se l'aereo sta già accelerando (accelerazione positiva), il termine $K_a a(t)$ riduce il comando di manetta, e viceversa se sta decelerando: si tratta di una retroazione che smorza l'azione del controllore tenendo conto dell'inerzia e della spinta già in sviluppo.

Il segnale $e_u(t)$ è quindi inviato a uno stadio *lag* del primo ordine, implementato come media esponenziale:

$$u(t) = \alpha e_u(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0}. \quad (51)$$

Con i parametri attuali si ha $\alpha = 1$ (per ogni $\Delta t > 0$), quindi

$$u(t) \equiv e_u(t), \quad (52)$$

ossia il filtro *lag* è degenerare e non applica alcuna smussatura temporale. In termini pratici, l'uscita del blocco *lag* coincide con l'ingresso, rendendo il controllore più reattivo ma anche più esposto a eventuale rumore del segnale.

Infine, il comando di manetta è vincolato entro i limiti operativi desiderati

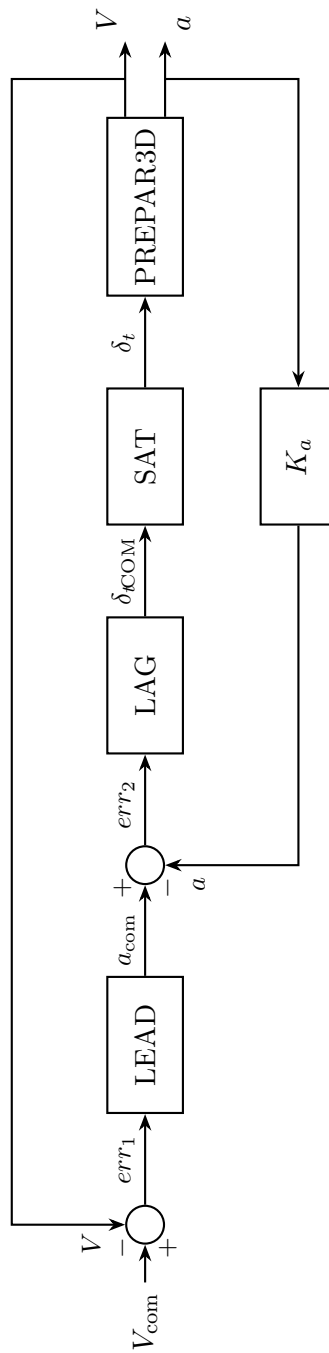


Figura 14: Schema a blocchi dell'autothrottle

4.6.4 Altitude Hold

L'*Altitude Hold* presente in Figure 15 é organizzato gerarchicamente in due anelli: un *anello esterno* che, a partire dall'errore di quota, calcola un *pitch di riferimento* e un *anello interno* che realizza tale riferimento comandando il timone. Questa architettura separa le dinamiche lente (quota) da quelle rapide (pitch), migliorando la robustezza complessiva.

Si definisce anzitutto l'errore di quota come

$$e_h(t) = h(t) - h_{\text{cmd}}(t), \quad (53)$$

dove $h(t)$ é la quota misurata e $h_{\text{cmd}}(t)$ la quota desiderata. L'implementazione adotta una protezione all'inizializzazione: se l'intervallo di campionamento Δt risulta troppo grande (ad es. al primo passo), i termini derivativo e integrale vengono azzerati per evitare transitori numerici.

La derivata dell'errore é calcolata come differenza in avanti

$$\dot{e}_h(t) = \frac{e_h(t) - e_h(t - \Delta t)}{\Delta t}, \quad (54)$$

mentre l'integrale sfrutta la regola del trapezio e include un termine di *anti-windup* basato sulla saturazione del pitch:

$$I_h(t) = I_h(t - \Delta t) + \frac{e_h(t) + e_h(t - \Delta t)}{2} \Delta t - K_{aw} (\theta_{\text{cmd}}(t - \Delta t) - \theta_{\text{sat}}(t - \Delta t)), \quad (55)$$

dove $K_{aw} > 0$ é il guadagno di back-calculation (nel codice $K_{aw} = 1$) e il termine tra parentesi rappresenta l'errore di saturazione del pitch (vedi oltre). In pratica, quando il pitch comandato viene limitato, si scarica l'integratore per prevenire accumulo eccessivo.

Il pitch desiderato é quindi fornito da un PID su e_h :

$$\theta_{\text{cmd}}(t) = K_p^h e_h(t) + K_i^h I_h(t) + K_d^h \dot{e}_h(t), \quad K_p^h = 0.08, K_i^h = 0.01, K_d^h = 0.09. \quad (56)$$

Per rispettare i limiti operativi si applica una saturazione simmetrica.

L'anello interno fa inseguire al velivolo il riferimento $\theta_{\text{sat}}(t)$, agendo sull'equilibratore. Si definisce

$$e_\theta(t) = \theta_{\text{sat}}(t) - \theta(t), \quad (57)$$

dove $\theta(t)$ é il pitch misurato.

Un blocco *lead* genera un rateo di pitch comandato combinando errore e sua derivata numerica:

$$\dot{\theta}_{\text{cmd}}(t) = K_\ell \left(e_\theta(t) + \tau_d \frac{e_\theta(t) - e_\theta(t - \Delta t)}{\Delta t} \right), \quad K_\ell = 9000, \tau_d = 0.3. \quad (58)$$

Questo rateo desiderato é poi confrontato con il rateo reale $q(t)$ (pitch rate), pesato da un guadagno K_q :

$$e_q(t) = \dot{\theta}_{\text{cmd}}(t) - K_q q(t), \quad K_q = 4500. \quad (59)$$

Il segnale $e_q(t)$ attraversa un filtro *lag* del primo ordine con coefficiente dipendente dal passo temporale:

$$u(t) = \alpha e_q(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0.8}, \quad (60)$$

dove $u(t)$ é il comando agli elevatori prima della limitazione. Infine si impone la saturazione compatibile con SimConnect.

La catena complessiva realizza:

$$h_{\text{cmd}} \xrightarrow{e_h, \text{PID+AW}} \theta_{\text{sat}} \xrightarrow{\text{Lead}} \dot{\theta}_{\text{cmd}} \xrightarrow{-K_q q} e_q \xrightarrow{\text{Lag}} u \xrightarrow{\text{sat}} \text{elevator}. \quad (61)$$

L'anello esterno garantisce il tracciamento della quota limitando il pitch entro valori ammissibili; l'anello interno assicura inseguimento rapido e ben smorzato del pitch di riferimento. L'anti-windup evita che l'integratore di quota accumuli errore quando il pitch é saturato, scongiurando sovraelongazioni al rientro.

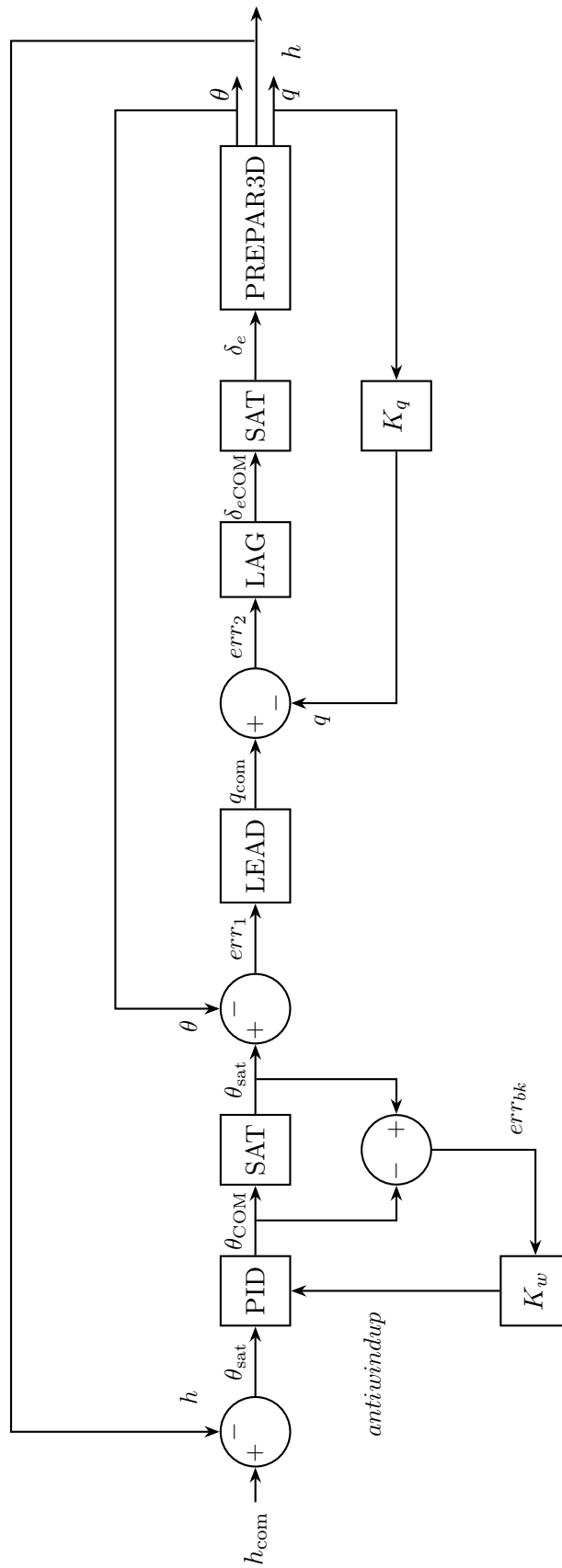


Figura 15: Schema a blocchi dell'altitude hold

4.6.5 Heading Hold

L'*Heading Hold* presente in Figure 16 mantiene la prua del velivolo al valore desiderato generando, in un anello esterno, un angolo di rollio di riferimento e, in un anello interno, il comando agli alettoni che realizza tale banca. L'architettura é dunque gerarchica: heading (lento) $\rightarrow$ bank (piú rapido).

Poiché l'heading é un angolo definito modulo 360° , l'errore va calcolato come l'angolo *minimo* che porta dalla prua attuale a quella desiderata. Indichiamo con $\psi_{\text{cmd}}(t)$ la prua richiesta e con $\psi(t)$ la prua misurata. L'errore avvolto é

$$e_\psi(t) = \text{wrap}_{(-180^\circ, 180^\circ]}(\psi_{\text{cmd}}(t) - \psi(t)), \quad (62)$$

cioé $e_\psi \in (-180^\circ, 180^\circ]$. Nel codice questo avviene con diramazioni che sommano o sottraggono 360° quando la differenza supera 180° .

Si applica una protezione all'inizializzazione: se il passo temporale Δt risulta grande (ad es. al primo ciclo), i termini derivativo e integrale sono azzerati per evitare problemi numerici. In regime normale si calcolano:

$$\dot{e}_\psi(t) = \frac{e_\psi(t) - e_\psi(t - \Delta t)}{\Delta t}, \quad (63)$$

$$I_\psi(t) = I_\psi(t - \Delta t) + \frac{e_\psi(t) + e_\psi(t - \Delta t)}{2} \Delta t - K_{aw} (\phi_{\text{cmd}}(t - \Delta t) - \phi_{\text{sat}}(t - \Delta t)), \quad (64)$$

dove il termine di *anti-windup* (back-calculation) scarica l'integratore quando l'angolo comandato viene saturato; nel codice $K_{aw} = 0.001$.

L'angolo di rollio di riferimento é dato da una legge PID, qui ridotta a puro P :

$$\phi_{\text{cmd}}(t) = K_p^\psi e_\psi(t) + K_i^\psi I_\psi(t) + K_d^\psi \dot{e}_\psi(t), \quad K_p^\psi = 10, \quad K_i^\psi = 0, \quad K_d^\psi = 0. \quad (65)$$

Si impone poi una saturazione simmetrica in funzione del limite operativo ϕ_{max}

L'anello interno fa seguire al velivolo l'angolo di rollio di riferimento $\phi_{\text{sat}}(t)$ agendo sugli alettoni. Si definisce

$$e_\phi(t) = \phi_{\text{sat}}(t) - \phi(t), \quad (66)$$

dove $\phi(t)$ é l'angolo. Un blocco *lead* genera un roll rate desiderato combinando errore e sua derivata numerica:

$$\dot{\phi}_{\text{cmd}}(t) = K_\ell \left(e_\phi(t) + \tau_d \frac{e_\phi(t) - e_\phi(t - \Delta t)}{\Delta t} \right), \quad K_\ell = 400, \quad \tau_d = 0.6. \quad (67)$$

Il roll rate desiderato é confrontato con il roll rate reale $p(t)$, pesato da K_r :

$$e_p(t) = \dot{\phi}_{\text{cmd}}(t) - K_r p(t), \quad K_r = 200. \quad (68)$$

Il segnale $e_p(t)$ attraversa quindi un filtro *lag* del primo ordine (media esponenziale) con coefficiente dipendente da Δt :

$$u(t) = \alpha e_p(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0.7}, \quad (69)$$

dove $u(t)$ é il comando agli alettoni prima della limitazione. Infine si applica la saturazione ammessa da SimConnect:

$$u(t) \in [-16383, 16383]. \quad (70)$$

$$\psi_{\text{cmd}} \xrightarrow{e_\psi} \phi_{\text{cmd}} \xrightarrow{\text{sat}} \phi_{\text{sat}} \xrightarrow{\text{Lead}} \dot{\phi}_{\text{cmd}} \xrightarrow{-K_r p} e_p \xrightarrow{\text{Lag}} u \xrightarrow{\text{sat}} \text{aileron}. \quad (71)$$

L'uso dell'errore bidirezionale assicura la via piú corta e correzioni su di essa; l'anello esterno trasforma l'errore di prua in un riferimento di angolo di rollio limitato limitato; l'anello interno *lead-lag* garantisce inseguimento rapido e smorzato del riferimento, mentre la saturazione finale rende il comando compatibile con i limiti fisici.

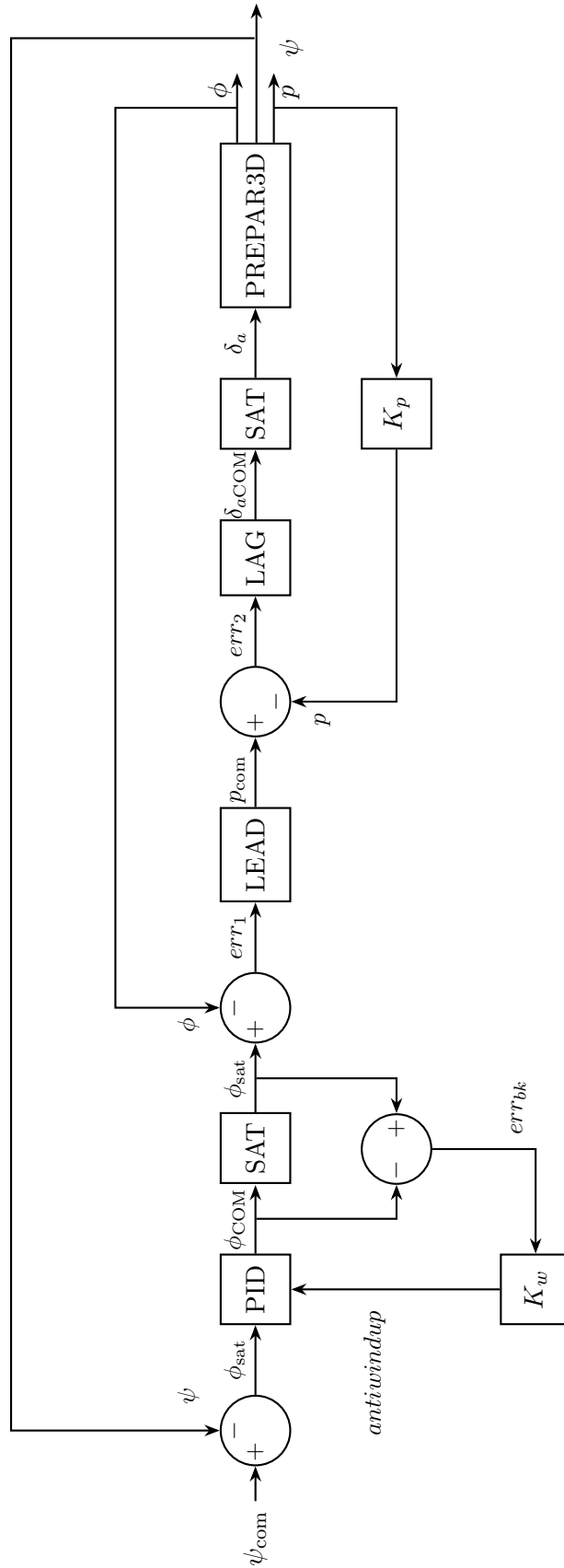


Figura 16: Schema a blocchi dell'heading hold

4.6.6 Vertical Separation

L'autopilota di *Vertical Separation* presente in Figure 17 mantiene una distanza verticale desiderata rispetto a un riferimento (ad esempio il velivolo leader), utilizzando come ingresso la separazione misurata nel sistema di riferimento locale NED (North-East-Down). In NED la coordinata D (indice 2 del vettore) é positiva verso il basso ciò implica, per convenzione, che una separazione verticale positiva corrisponde ad un drone più in basso. L'architettura di controllo é gerarchica: un *anello esterno* (PID su separazione verticale) fornisce un *pitch di riferimento* saturato, seguito da un *anello interno* (pitch hold) che realizza tale riferimento comandando gli equilibratori tramite una catena *lead-lag*.

Sia $NED = [N, E, D]^T$ il vettore di separazione nel frame NED e sia s_{des} la separazione verticale desiderata. L'errore in *piedi* é costruito come

$$e_h(t) = -(D(t) - s_{des}) \cdot \kappa, \quad \kappa = 3.28084 \text{ (ft/m)}. \quad (72)$$

Il segno “ $-$ ” tiene conto dell'asse D positivo verso il basso: se il riferimento é più in basso (D maggiore), il controllo comanda un pitch coerente con la risalita/discesa necessaria secondo la convenzione di segno adottata.

Per robustezza numerica si introduce una protezione: se il passo Δt risulta grande (ad es. all'inizializzazione), i termini derivativo e integrativo vengono azzerati. In regime normale si hanno:

$$\dot{e}_h(t) = \frac{e_h(t) - e_h(t - \Delta t)}{\Delta t}, \quad (73)$$

$$I_h(t) = I_h(t - \Delta t) + \frac{e_h(t) + e_h(t - \Delta t)}{2} \Delta t - K_{aw} (\theta_{cmd}(t - \Delta t) - \theta_{sat}(t - \Delta t)), \quad (74)$$

dove il termine di *anti-windup* (back-calculation) sottrae dall'integratore l'errore di saturazione del pitch, evitando accumulo quando il comando é limitato. Nel codice $K_{aw} = 1$.

Il pitch desiderato é quindi dato da un PID su e_h :

$$\theta_{cmd}(t) = K_p^h e_h(t) + K_i^h I_h(t) + K_d^h \dot{e}_h(t), \quad K_p^h = 0.08, \quad K_i^h = 0.01, \quad K_d^h = 0.09. \quad (75)$$

Si applica quindi una saturazione simmetrica per rispettare il limite operativo θ_{max} passato alla funzione:

$$\theta_{sat}(t) = \begin{cases} \theta_{max}, & \text{se } \theta_{cmd}(t) > \theta_{max} \text{ e } \theta_{cmd}(t) > 0, \\ -\theta_{max}, & \text{se } \theta_{cmd}(t) < -\theta_{max} \text{ e } \theta_{cmd}(t) < 0, \\ \theta_{cmd}(t), & \text{altrimenti.} \end{cases} \quad (76)$$

L'anello interno fa inseguire al velivolo il riferimento $\theta_{sat}(t)$ agendo sugli elevatori.

Definiamo l'errore di pitch

$$e_\theta(t) = \theta_{sat}(t) - \theta(t), \quad (77)$$

dove $\theta(t)$ é il pitch misurato. Un blocco *lead* genera un rateo di pitch desiderato combinando errore e derivata numerica:

$$\dot{\theta}_{cmd}(t) = K_\ell \left(e_\theta(t) + \tau_d \frac{e_\theta(t) - e_\theta(t - \Delta t)}{\Delta t} \right), \quad K_\ell = 9000, \quad \tau_d = 0.2. \quad (78)$$

Questo rateo desiderato é confrontato con il rateo reale $q(t)$ (pitch rate), pesato da K_q :

$$e_q(t) = \dot{\theta}_{\text{cmd}}(t) - K_q q(t), \quad K_q = 4500. \quad (79)$$

Il segnale $e_q(t)$ é poi filtrato con un *lag* (media esponenziale) a coefficiente dipendente da Δt :

$$u(t) = \alpha e_q(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0.8}, \quad (80)$$

dove $u(t)$ rappresenta il comando grezzo agli elevatori. L'uscita é infine limitata secondo i vincoli di SimConnect:

$$u(t) \in [-16383, 16383]. \quad (81)$$

$$(N, E, D), s_{\text{des}} \longrightarrow e_h \xrightarrow{\text{PID+AW}} \theta_{\text{sat}} \xrightarrow{\text{Lead}} \dot{\theta}_{\text{cmd}} \xrightarrow{-K_q q} e_q \xrightarrow{\text{Lag}} u \xrightarrow{\text{sat}} \text{elevator}. \quad (82)$$

La conversione metrica κ assicura coerenza delle unitá, l'anti-windup impedisce l'accumulo dell'integratore quando il pitch é saturato, e la catena *lead-lag* garantisce un inseguimento del riferimento rapido ma smorzato.

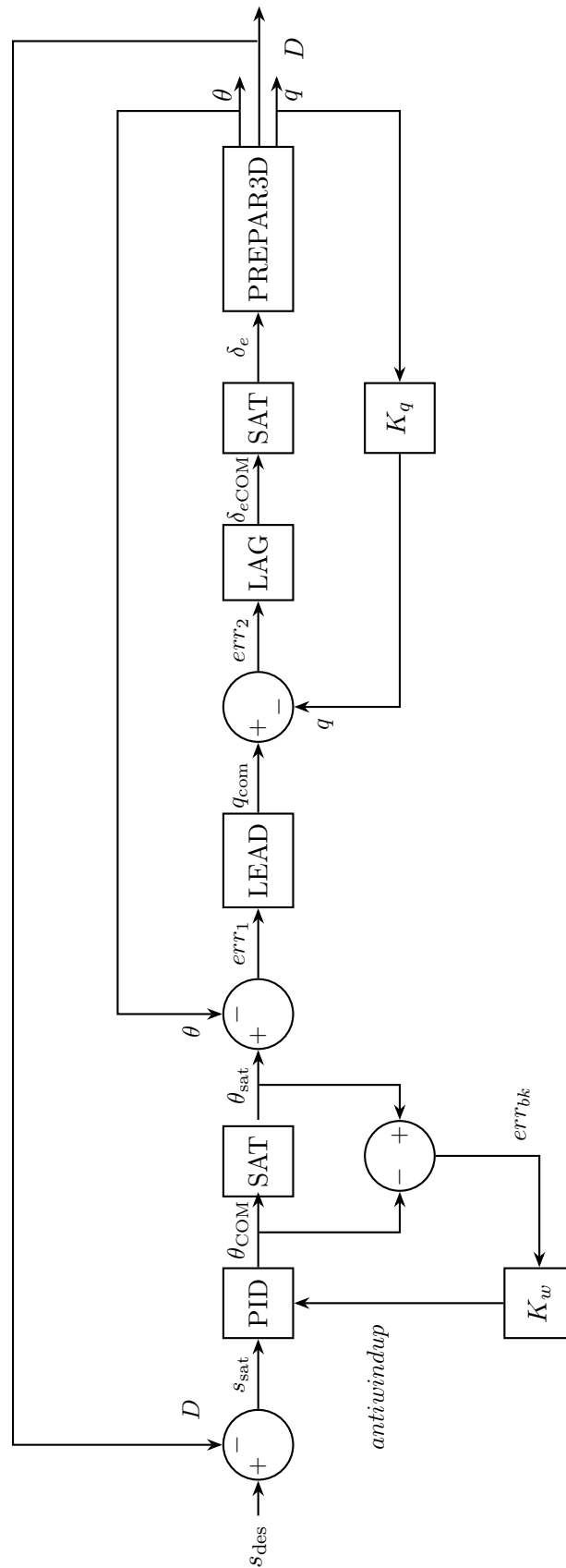


Figura 17: Schema a blocchi Vertical Separation Autopilot

4.6.7 Lateral Separation

L'autopilota di *Lateral Separation* presente in Figure 18 mantiene una distanza prefissata lungo l'asse Est del frame NED (North-East-Down). Indichiamo con

$$\text{NED}(t) = \begin{bmatrix} N(t) \\ E(t) \\ D(t) \end{bmatrix} \quad (83)$$

il vettore di separazione relativo (leader $\rightarrow$ ownship) e con s_{lat} la separazione laterale desiderata lungo l'asse Est. L'architettura del controllore é gerarchica: un *anello esterno* trasforma l'errore laterale in una *banca di riferimento*, poi un *anello interno* di *bank hold* realizza tale riferimento comandando gli alettoni attraverso una catena *lead-lag*.

L'errore laterale é definito (coerentemente al codice) come

$$e_y(t) = -(s_{\text{lat}} - E(t)) = E(t) - s_{\text{lat}}. \quad (84)$$

Questa convenzione di segno é scelta per allinearsi alle direzioni positive interne; il *segno* del guadagno proporzionale (e degli altri termini) determina la direzione della correzione.

Per robustezza numerica si applica una protezione all'inizializzazione: se il passo di campionamento Δt risulta grande (ad es. al primo ciclo), i termini derivativo e integrale sono azzerati. In regime normale, si calcolano:

$$\dot{e}_y(t) = \frac{e_y(t) - e_y(t - \Delta t)}{\Delta t}, \quad (85)$$

$$I_y(t) = I_y(t - \Delta t) + \frac{e_y(t) + e_y(t - \Delta t)}{2} \Delta t - K_{aw} (\phi_{\text{cmd}}(t - \Delta t) - \phi_{\text{sat}}(t - \Delta t)), \quad (86)$$

dove il termine di *anti-windup* realizza una back-calculation sull'integratore quando la banca comandata viene saturata; nel codice $K_{aw} = 0.001$.

La banca di riferimento é quindi una legge PID (qui con guadagni esplicitati):

$$\phi_{\text{cmd}}(t) = K_p^y e_y(t) + K_i^y I_y(t) + K_d^y \dot{e}_y(t), \quad K_p^y = 1, \quad K_i^y = 0.001, \quad K_d^y = 3.6. \quad (87)$$

Per rispettare i limiti operativi si impone la saturazione:

$$\phi_{\text{sat}}(t) = \begin{cases} \phi_{\text{max}}, & \text{se } \phi_{\text{cmd}}(t) > \phi_{\text{max}} \text{ e } \phi_{\text{cmd}}(t) > 0, \\ -\phi_{\text{max}}, & \text{se } \phi_{\text{cmd}}(t) < -\phi_{\text{max}} \text{ e } \phi_{\text{cmd}}(t) < 0, \\ \phi_{\text{cmd}}(t), & \text{altrimenti.} \end{cases} \quad (88)$$

L'anello interno fa seguire al velivolo la banca di riferimento $\phi_{\text{sat}}(t)$ agendo sugli alettoni. Si definisce l'errore di banca

$$e_\phi(t) = \phi_{\text{sat}}(t) - \phi(t), \quad (89)$$

dove $\phi(t)$ é la banca misurata. Un blocco *lead* genera un roll rate desiderato combinando errore e derivata numerica:

$$\dot{\phi}_{\text{cmd}}(t) = K_\ell \left(e_\phi(t) + \tau_d \frac{e_\phi(t) - e_\phi(t - \Delta t)}{\Delta t} \right), \quad K_\ell = 400, \quad \tau_d = 0.6. \quad (90)$$

Il roll rate desiderato é confrontato con il roll rate reale $p(t)$, pesato da K_r :

$$e_p(t) = \dot{\phi}_{\text{cmd}}(t) - K_r p(t), \quad K_r = 200. \quad (91)$$

Il segnale $e_p(t)$ attraversa quindi un filtro *lag* del primo ordine (media esponenziale) con coefficiente dipendente da Δt :

$$u(t) = \alpha e_p(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0.7}, \quad (92)$$

dove $u(t)$ é il comando agli alettoni prima della limitazione. Infine si impone la saturazione compatibile con SimConnect:

$$u(t) \in [-16383, 16383]. \quad (93)$$

$$E, s_{\text{lat}} \longrightarrow e_y \xrightarrow{\text{PID+AW}} \phi_{\text{sat}} \xrightarrow{\text{Lead}} \dot{\phi}_{\text{cmd}} \xrightarrow{-K_{rp}} e_p \xrightarrow{\text{Lag}} u \xrightarrow{\text{sat}} \text{aileron}. \quad (94)$$

L'anello esterno mappa l'errore laterale in un riferimento di banca limitato; l'anello interno *lead-lag* assicura inseguimento rapido e smorzato del riferimento. L'anti-windup previene l'accumulo dell'integratore quando la banca é satura.

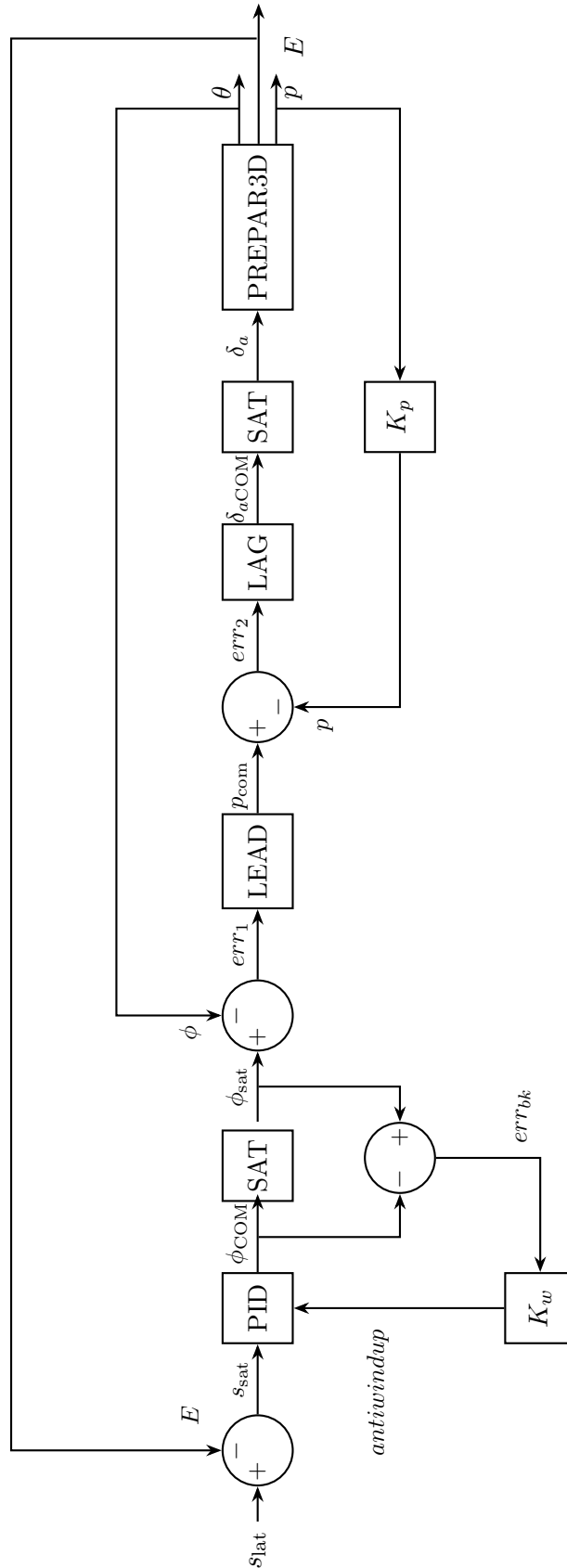


Figura 18: Schema a blocchi del lateral separation autopilot

4.6.8 Forward Separation

Il controllore di *Forward Separation* presente in Figure 17 mantiene una distanza prefissata lungo l'asse Nord del frame NED (North–East–Down), cioè lungo la direzione "avanti/indietro" rispetto al riferimento. Indichiamo con

$$\text{NED}(t) = \begin{bmatrix} N(t) \\ E(t) \\ D(t) \end{bmatrix} \quad (95)$$

il vettore di separazione relativo (leader $\rightarrow$ ownship) e con s_{fwd} la separazione longitudinale desiderata sull'asse N . L'architettura é gerarchica: un *anello esterno* converte l'errore di separazione in una *velocità di riferimento* V_{cmd} , mentre un *anello interno* di *autothrottle* realizza tale velocità agendo sulla manetta.

Coerentemente al codice, l'errore longitudinale é

$$e_x(t) = s_{\text{fwd}} - N(t). \quad (96)$$

Per robustezza numerica si applica una protezione all'inizializzazione: se il passo temporale Δt é grande (ad es. al primo ciclo), i termini derivativo e integrale sono azzerati. In regime normale si calcolano:

$$\dot{e}_x(t) = \frac{e_x(t) - e_x(t - \Delta t)}{\Delta t}, \quad (97)$$

$$I_x(t) = I_x(t - \Delta t) + \frac{e_x(t) + e_x(t - \Delta t)}{2} \Delta t - K_{aw} (V_{\text{cmd}}(t - \Delta t) - V_{\text{sat}}(t - \Delta t)), \quad (98)$$

dove il termine di *anti-windup* (back-calculation) scarica l'integratore quando la velocità comandata viene saturata; nel codice $K_{aw} = 0.5$.

La velocità di riferimento é quindi una legge PID su e_x :

$$V_{\text{cmd}}(t) = K_p^x e_x(t) + K_i^x I_x(t) + K_d^x \dot{e}_x(t), \quad K_p^x = 1, \quad K_i^x = 0.03, \quad K_d^x = 2. \quad (99)$$

Per rispettare i vincoli operativi si impone la saturazione:

$$V_{\text{sat}}(t) = \begin{cases} 1060, & \text{se } V_{\text{cmd}}(t) > 1060, \\ 600, & \text{se } V_{\text{cmd}}(t) < 600, \\ V_{\text{cmd}}(t), & \text{altrimenti.} \end{cases} \quad (100)$$

Il valore V_{sat} diventa il riferimento per l'anello interno.

Definiamo l'errore di velocità come

$$e_V(t) = V_{\text{sat}}(t) - V(t), \quad (101)$$

dove $V(t)$ é la velocità misurata. Lo stadio *lead* dell'autothrottle costruisce un comando "grezzo" di manetta combinando e_V e la sua derivata numerica:

$$u_{\text{cmd}}(t) = K_\ell \left(e_V(t) + \tau_d \frac{e_V(t) - e_V(t - \Delta t)}{\Delta t} \right), \quad K_\ell = 3000, \quad \tau_d = 0.27. \quad (102)$$

Per smorzare l'azione in funzione della dinamica propulsiva, u_{cmd} é corretto con una retroazione sull'accelerazione longitudinale $a(t)$:

$$e_u(t) = u_{\text{cmd}}(t) - K_a a(t), \quad K_a = 1500. \quad (103)$$

Lo stadio *lag* é implementato come media esponenziale

$$u(t) = \alpha e_u(t) + (1 - \alpha) u(t - \Delta t), \quad \alpha = \frac{\Delta t}{\Delta t + 0}, \quad (104)$$

che, con i parametri correnti, dá $\alpha = 1$ (per ogni $\Delta t > 0$), quindi $u(t) \equiv e_u(t)$: il lag é degenero e non introduce filtraggio. Infine si impone la saturazione della manetta:

$$u(t) \in [0.3 \cdot 16383, 0.9 \cdot 16383]. \quad (105)$$

$$N, s_{\text{fwd}} \longrightarrow e_x \xrightarrow{\text{PID+AW}} V_{\text{sat}} \xrightarrow{\text{Lead}} u_{\text{cmd}} \xrightarrow{-K_a a} e_u \xrightarrow{\text{Lag } (\alpha=1)} u \xrightarrow{\text{sat}} \text{throttle}. \quad (106)$$

L'anello esterno regola la distanza lungo l'asse Nord traducendola in una velocità ammissibile; l'anello interno *lead-lag* regola la manetta per inseguire la velocità, smorzando l'azione in base all'accelerazione già presente. La scelta $\alpha = 1$ rende l'autothrottle molto reattivo; qualora si desiderasse maggiore smussatura, basterebbe adottare una costante positiva al denominatore (es. $\alpha = \frac{\Delta t}{\Delta t + \tau_{\text{lag}}}$ con $\tau_{\text{lag}} > 0$).

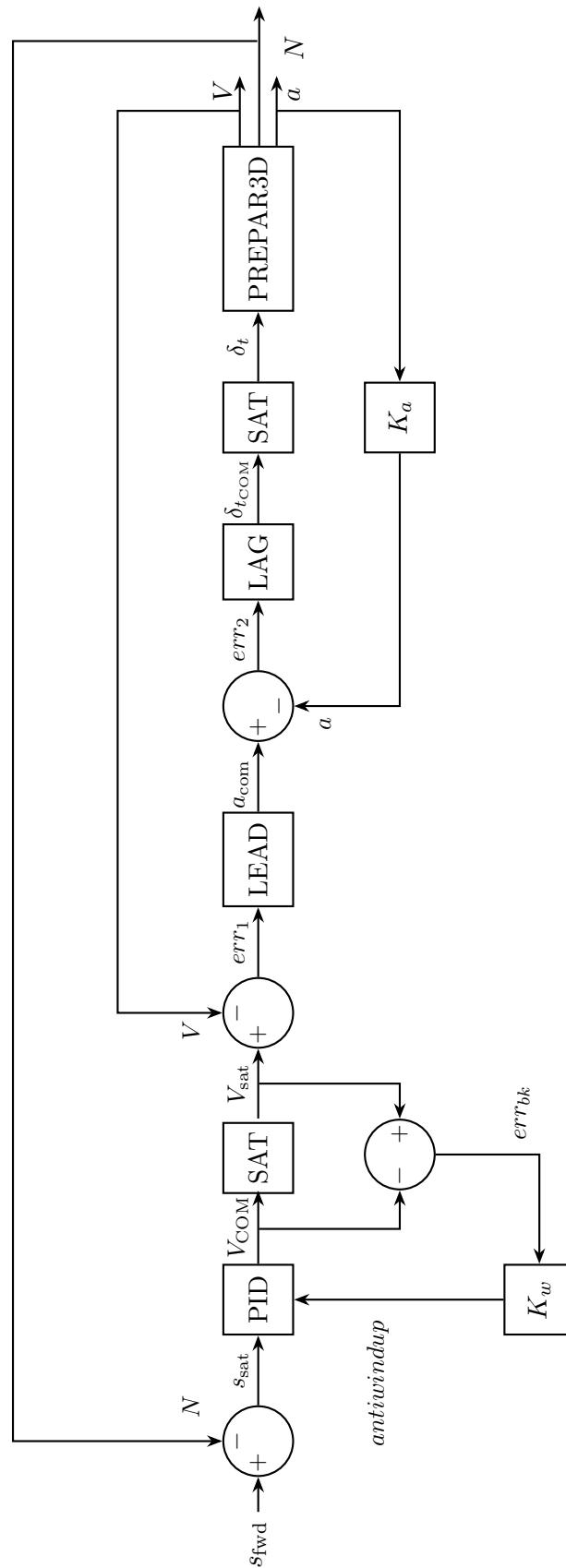


Figura 19: Schema a blocchi del forward separation autopilot

4.7 Simulazioni

Le simulazioni sono state condotte utilizzando l'ambiente *Prepar3D v6* precedentemente descritto e impiegando come piattaforma di volo il velivolo *Lockheed Martin F-35A Lightning II*. L'obiettivo delle simulazioni è quello di analizzare e confrontare il comportamento dinamico del velivolo in due condizioni distinte: una condizione non perturbata, priva cioè di vento e turbolenze, e una condizione perturbata, caratterizzata dalla presenza di vento trasversale, raffiche, turbolenza e wind shear a bassa quota in modo da testare la robustezza dei controlli progettati.

In entrambi i casi si è scelto di mantenere costanti alcuni parametri generali al fine di garantire coerenza e comparabilità tra gli scenari. Le operazioni si sono svolte presso l'aeroporto Ronald Reagan Washington National (KDCA), utilizzando la pista 1 come punto di partenza. L'orario simulato è stato fissato alle ore 12:00 locali in un contesto stagionale estivo. La modalità di simulazione selezionata è stata quella multiplayer, con due istanze connesse in rete locale. Il velivolo host, denominato *C2 Plane*, ha avuto il compito di gestire la sessione, mentre il velivolo client, denominato *DRONE*, ha partecipato in parallelo riproducendo le medesime condizioni.

Entrambi i velivoli hanno utilizzato la stessa configurazione iniziale e lo stesso modello aerodinamico di F-35A. Il collegamento multiplayer è stato configurato con un'elevata frequenza di aggiornamento pari a 0.05 secondi, riducendo così eventuali disallineamenti temporali o spaziali.

4.7.1 Condizioni non perturbate

Nel primo scenario si è optato per una configurazione priva di disturbi meteorologici. Il cielo è stato impostato sereno, con vento calmo e totale assenza di turbolenze e shear. Questa condizione ha permesso di osservare il comportamento del velivolo in una situazione di riferimento nominale.

Durante il rullaggio e il decollo l'F-35A ha mostrato un'ottima stabilità dinamica. L'allineamento con l'asse pista è stato mantenuto senza richiedere correzioni significative e la fase di virata si è svolta in maniera fluida e lineare. La salita iniziale è stata caratterizzata da un assetto costante e da risposte ai comandi dell'autopilota precise e prevedibili. L'assenza di disturbi ha consentito di effettuare un volo pulito e regolare, che può essere considerato come la baseline per il confronto con le condizioni perturbate.

Dal punto di vista della simulazione in rete, entrambi i velivoli, host e client, hanno mostrato una perfetta sincronizzazione. Le posizioni, i movimenti e l'evoluzione temporale delle manovre coincidevano, confermando la validità del collegamento multiplayer e la robustezza della piattaforma utilizzata.

In conclusione, lo scenario non perturbato ha permesso di verificare la stabilità intrinseca del modello aerodinamico del simulatore e degli autopiloti e di disporre di un caso di riferimento utile per analizzare successivamente le differenze introdotte dalle perturbazioni atmosferiche.

4.7.2 Condizioni perturbate

Nel secondo scenario sono state introdotte condizioni atmosferiche avverse con l'obiettivo di valutare la risposta del velivolo e la robustezza degli autopiloti. Il vento prevalente proveniva da ovest, con direzione 270 gradi, ed era caratterizzato da raffiche variabili tra zero e trenta

nodi. é stata inoltre attivata una turbolenza di intensit  moderata e un wind shear a bassa quota, localizzato a circa cinquecento piedi dal suolo.

Fin dalla fase di rullaggio si   osservata una chiara influenza del vento trasversale. L'F-35A tendeva a deviare lateralmente rispetto all'asse pista, richiedendo continue correzioni al timone per mantenere l'allineamento. Durante la viarta e la salita iniziale le raffiche improvvise hanno generato oscillazioni sia in rollio sia in beccheggio, complicando la stabilit  dell'assetto. La turbolenza ha reso pi  difficoltoso il mantenimento di una traiettoria regolare, imponendo agli autopiloti di effettuare micro-correzioni costanti per contrastare le perturbazioni.

L'ingresso nella zona di wind shear ha rappresentato la fase pi  critica della simulazione. La velocit  indicata ha subito variazioni repentine, riproducendo il rischio di una perdita momentanea di portanza. Questa condizione   notoriamente pericolosa nelle operazioni reali, soprattutto nelle fasi di decollo e atterraggio, e la simulazione ne ha mostrato con efficacia la complessit  gestionale.

Sul piano del multiplayer, anche in presenza di perturbazioni, entrambi i velivoli hanno condiviso le stesse condizioni atmosferiche. Le variazioni dovute a vento, turbolenza e shear sono state percepite in maniera coerente da host e client, dimostrando la consistenza del modello meteorologico distribuito.

In sintesi, lo scenario perturbato ha evidenziato la maggiore complessit  operativa del volo in condizioni atmosferiche avverse.

5 Risultati

In questa sezione vengono discussi i risultati ottenuti dai diversi controllori implementati per il velivolo principale e gregario nella configurazione *leader follower*. Ogni autopilota é stato valutato mediante simulazioni dedicate e i risultati sono riportati sotto forma di grafici temporali e spaziali. La descrizione include un'analisi qualitativa del comportamento dinamico e una quantificazione numerica delle principali metriche prestazionali: tempo di salita (t_r), tempo di assestamento (t_s), sovraelongazione percentuale (%OS), errore a regime (e_∞) e valori massimi dei comandi/attuatori.

5.0.1 Caso non perturbato

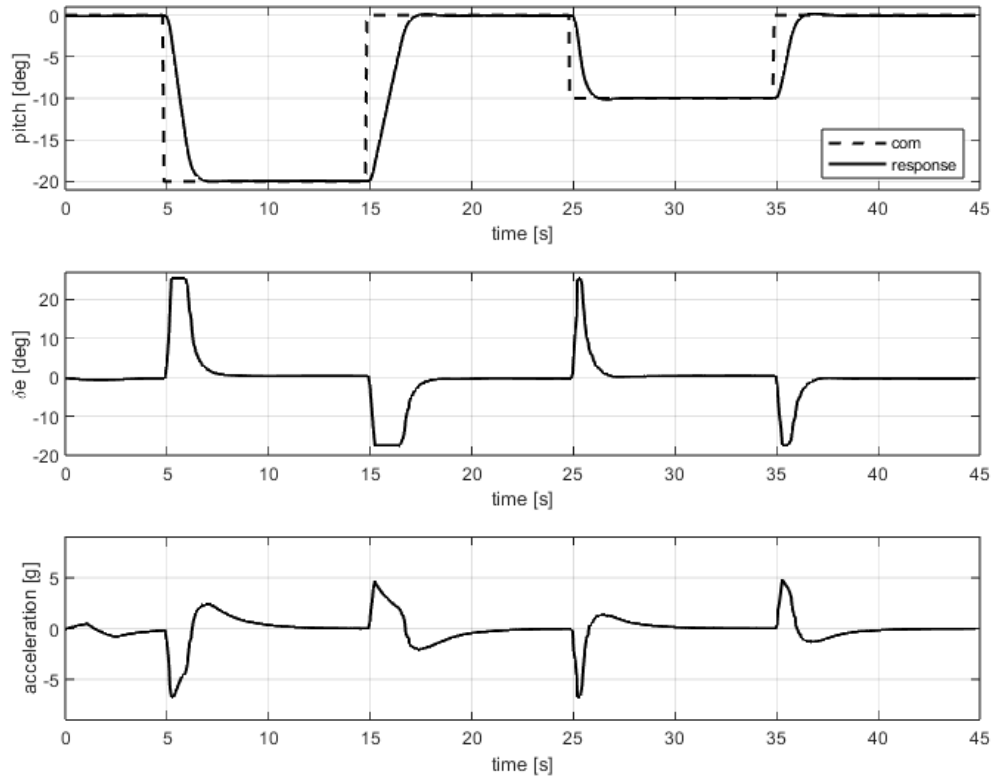


Figura 20: Pitch Hold.

Il controllore *Pitch Hold* (Figure 20) ha il compito di stabilizzare l'assetto longitudinale, agendo sull'equilibratore. La struttura *lead-lag* consente di convertire l'errore di pitch in un rateo desiderato, anticipando i cambiamenti rapidi (componente lead) e filtrando le alte frequenze (componente lag). Il grafico mostra un inseguimento rapido e stabile, con transitori puliti e assenza di oscillazioni persistenti.

Risultati numerici: $t_r \approx 2.5\text{--}3.0$ s, $t_s \approx 4\text{--}5$ s, %OS $\approx 5\text{--}8\%$, $e_\infty < 0.5^\circ$, deflessione massima $\delta_e \approx 20^\circ$, accelerazione massima $a_x \approx 5\text{--}6$ g.

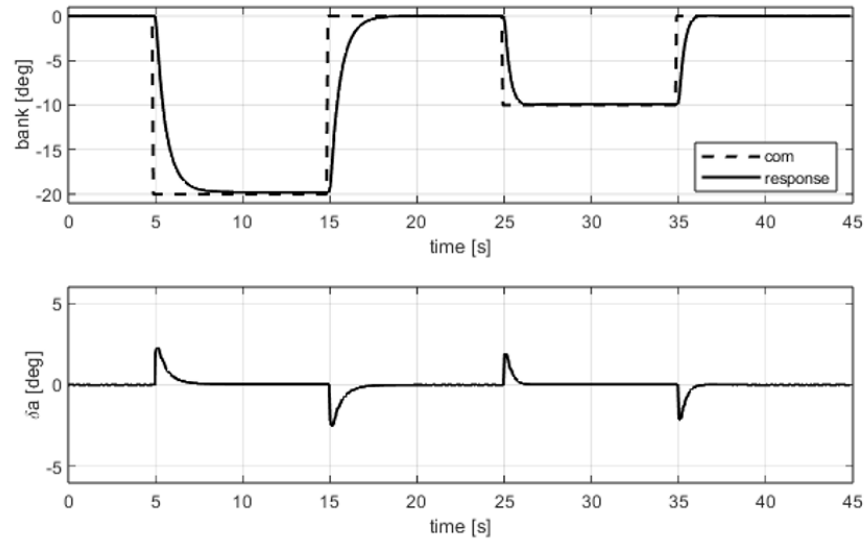


Figura 21: Bank Hold.

Il controllore *Bank Hold* (Figure 21) regola l'angolo di rollio tramite gli alettoni, anch'esso con struttura *lead-lag*. La risposta mostra un inseguimento regolare, con piccoli picchi nel comando di rollio dovuti all'azione derivativa, ma senza instabilità.

Risultati numerici: $t_r \approx 2$ s, $t_s \approx 3-4$ s, $\%OS \approx 3-5\%$, $e_\infty < 0.3^\circ$, deflessione massima $\delta_a \approx 2-3^\circ$.

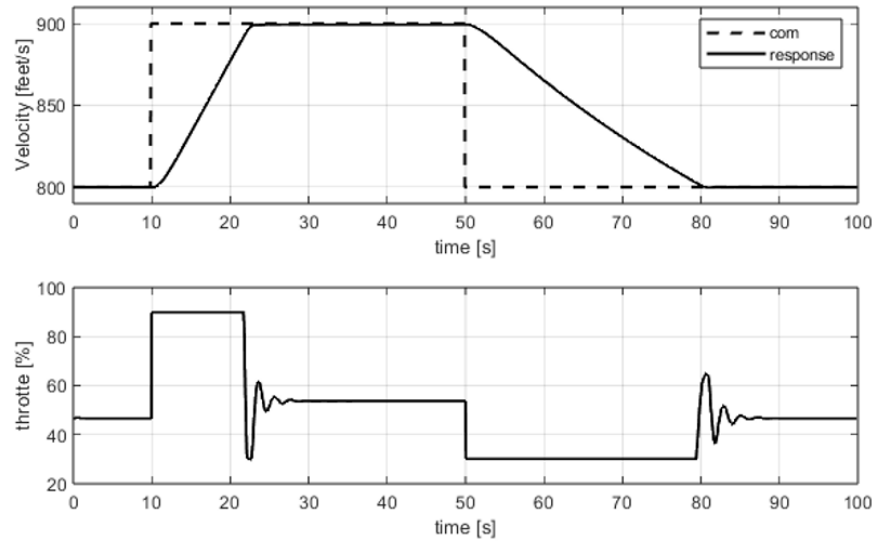


Figura 22: Autothrottle.

Il controllore *Autothrottle* (Figure 22) regola la velocità longitudinale agendo sulla manetta. La dinamica evidenzia tempi più lunghi rispetto ai canali d'assetto, riflettendo l'inerzia propulsiva. Il feedback sull'accelerazione longitudinale riduce i comandi quando il velivolo è già in accelerazione o decelerazione, migliorando la stabilità complessiva.

Risultati numerici: per uno step 800→900 ft/s, $t_r \approx 8\text{--}10$ s, $t_s \approx 12\text{--}15$ s, $\%OS \approx 0\text{--}2\%$, $e_\infty < 2$ ft/s; variazione throttle 40–90%, stabilizzazione al 55%. In decelerazione, $t_s \approx 25\text{--}30$ s.

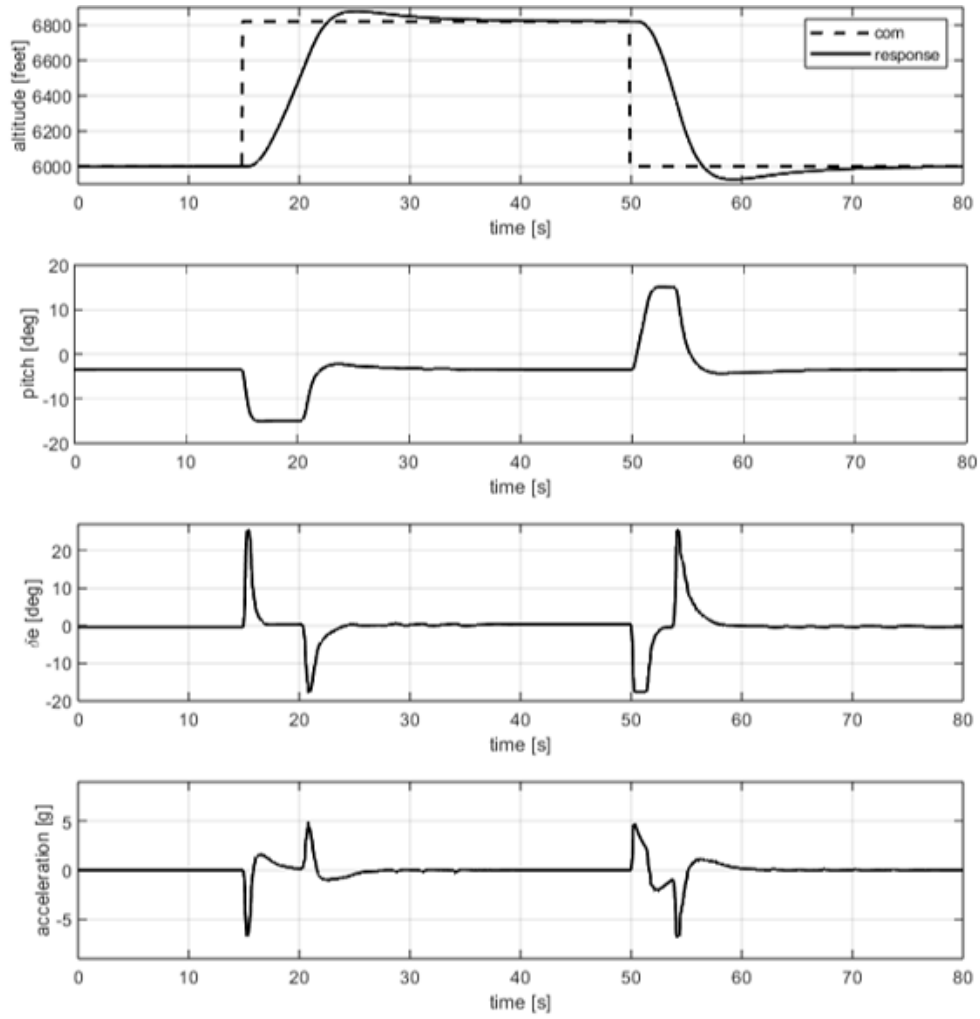


Figura 23: Altitude Hold.

L'autopilota *Altitude Hold* (Figure 23) é organizzato in due anelli: PID esterno per la quota e controllore interno di pitch. L'errore iniziale di quota viene compensato rapidamente, con picchi contenuti sugli elevatori. La struttura a cascata assicura robustezza anche in presenza di perturbazioni.

Risultati numerici: per uno step 6000→6800 ft, $t_r \approx 8\text{--}10$ s, $t_s \approx 12\text{--}14$ s, $\%OS \approx 3\text{--}5\%$, $e_\infty < 20$ ft, pitch massimo $|\theta| \approx 15^\circ$, $\delta_e \approx 20^\circ$, $a_x \approx 4\text{--}5$ g.

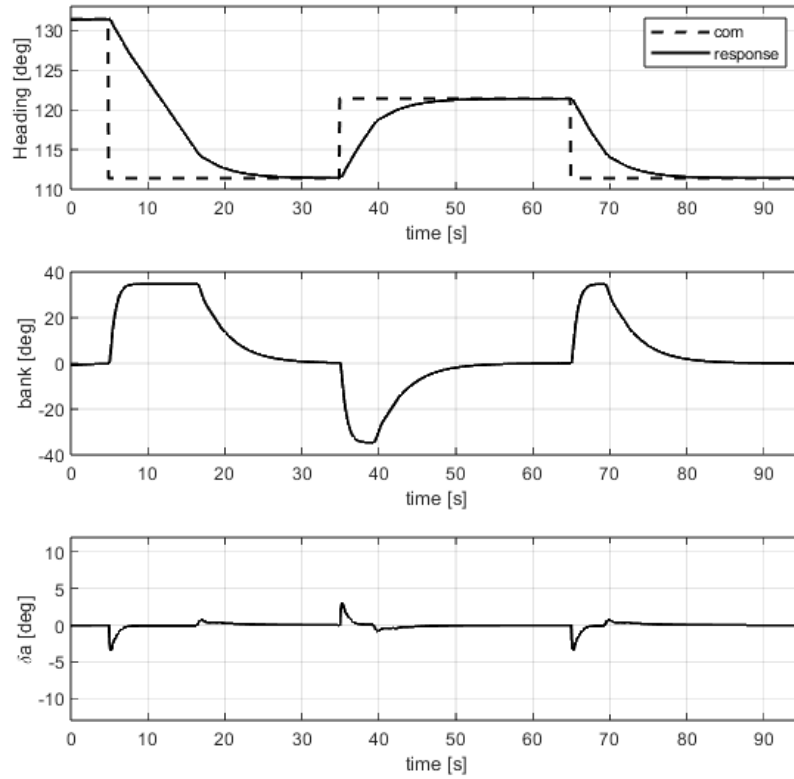


Figura 24: Heading Hold.

Il controllore *Heading Hold* (Figure 24) adotta una struttura gerarchica: l'errore di prua viene convertito in un bank di riferimento dall'anello esterno, poi realizzato dall'anello interno di rollio. I grafici mostrano un inseguimento stabile della rotta con overshoot moderati e saturazioni ben gestite.

Risultati numerici: $t_s \approx 6-8$ s, $\%OS \approx 5-7\%$, $e_\infty < 0.5^\circ$, bank massimo $|\phi| \approx 35^\circ$, $\delta_a \approx 6-7^\circ$.

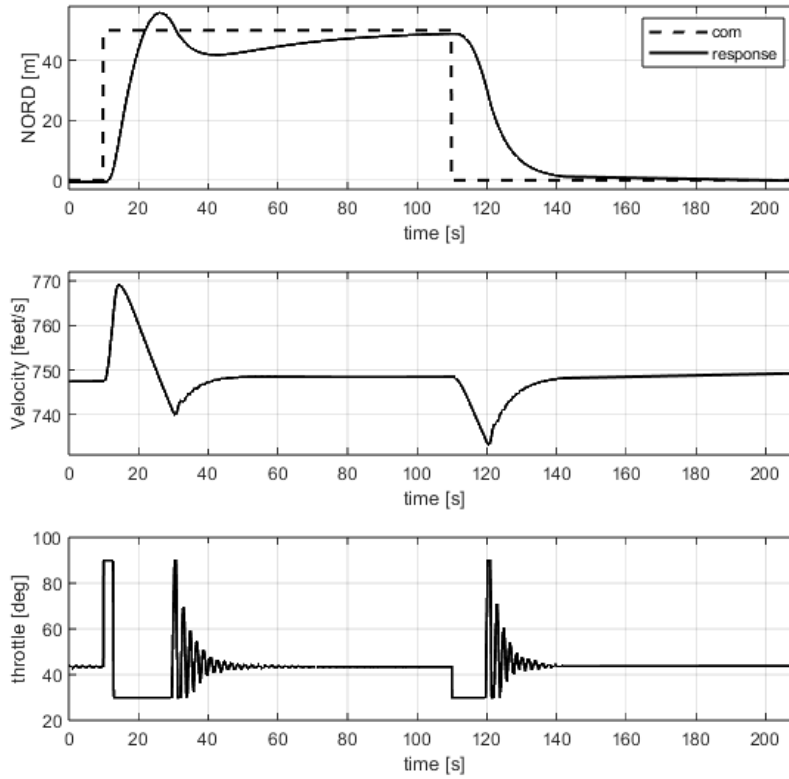


Figura 25: Forward separation autopilot.

Il controllore di *Forward Separation* (Figure 25) mantiene la distanza longitudinale dal leader. Le traiettorie bidimensionali e tridimensionali mostrano che il follower segue con precisione anche durante manovre complesse. Le discrepanze massime si verificano nelle variazioni rapide di rotta, ma vengono recuperate in pochi secondi.

Risultati numerici: errore RMS lungo rotta $\approx 20\text{--}40$ m, errore massimo in manovra $\approx 80\text{--}120$ m, errore altimetrico $\approx 30\text{--}50$ m, tempo di riallineamento dopo manovra leader $\approx 6\text{--}9$ s.

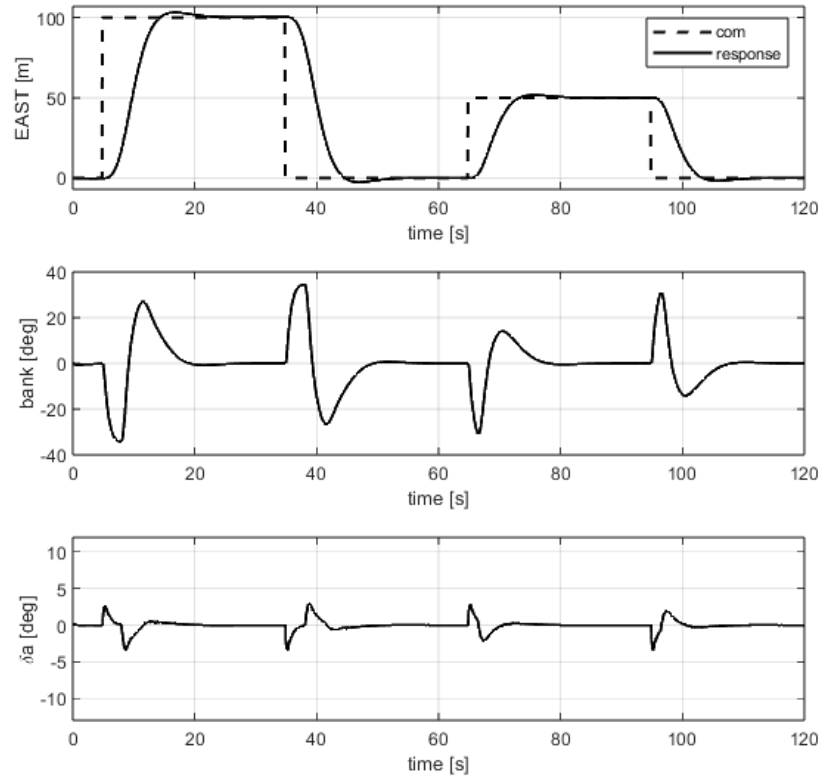


Figura 26: Lateral separation autopilot.

Il controllore di *Lateral Separation* (Figure 26) mantiene la distanza desiderata lungo l'asse Est rispetto al leader. La distanza viene seguita con precisione, con oscillazioni moderate nei transitori e angoli di rollio consistenti. L'anti-windup previene accumuli durante le saturazioni.

Risultati numerici: $t_s \approx 8-10$ s, $\%OS \approx 8-12\%$, $e_\infty < 2$ m, bank massimo $|\phi| \approx 30-35^\circ$, $\delta_a \approx 2-3^\circ$.

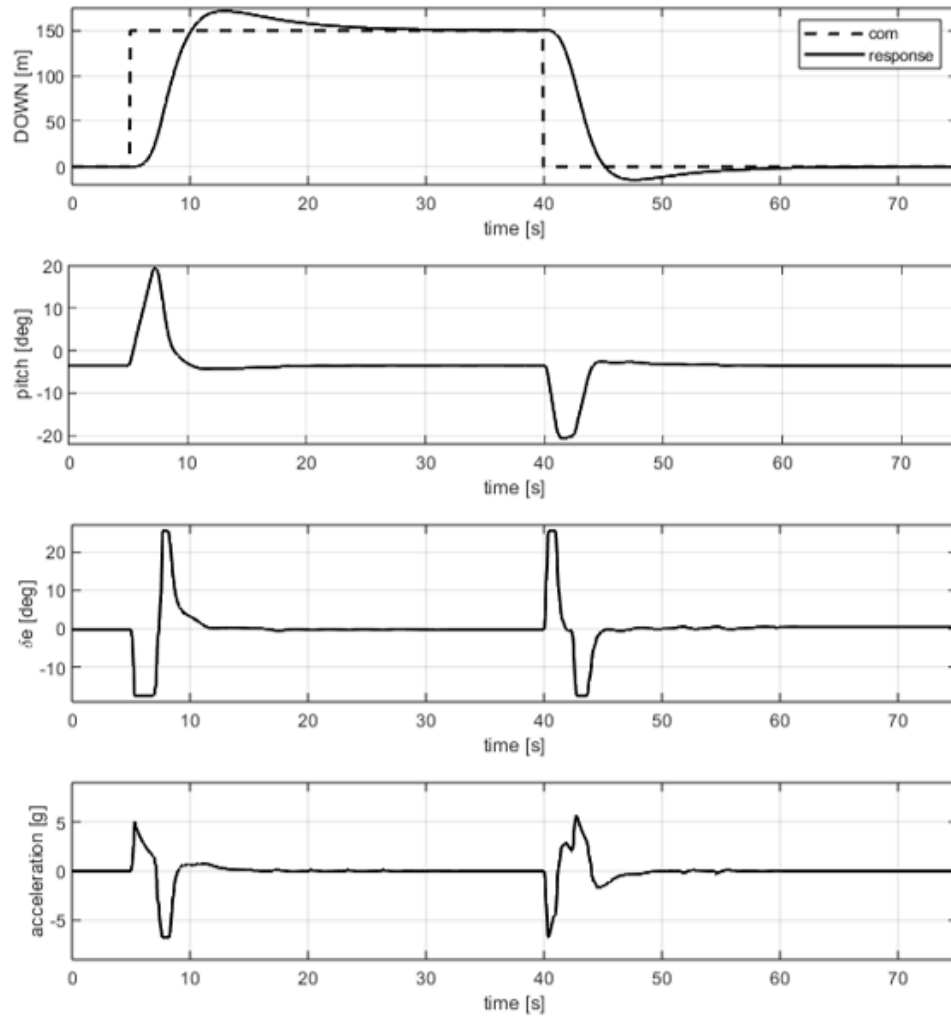


Figura 27: Vertical separation autopilot.

Il controllore di *Vertical Separation* (Figure 27) regola la distanza verticale dal leader con struttura simile all'*Altitude Hold*. La risposta evidenzia prontezza e correzioni aggressive ma stabili.

Risultati numerici: $t_s \approx 9\text{--}11$ s, $\%OS \approx 10\text{--}12\%$, $e_\infty < 3$ m, pitch massimo $|\theta| \approx 18^\circ$, $\delta_e \approx 16\text{--}18^\circ$, $a_x \approx 5\text{--}6$ g.

La Tabella 1 riassume i principali valori numerici per ciascun autopilota.

Autopilota	t_r [s]	t_s [s]	%OS	e_∞	Comando/Attuatore
Pitch Hold	2.5–3.0	4–5	5–8%	$< 0.5^\circ$	$\delta_e \approx 20^\circ$
Bank Hold	~ 2	3–4	3–5%	$< 0.3^\circ$	$\delta_a \approx 3^\circ$
Autothrottle	8–10	12–15 / 25–30	~ 0 –2%	< 2 ft/s	Throttle 40–90%
Heading Hold	–	6–8	5–7%	$< 0.5^\circ$	$\phi \approx 35^\circ$
Altitude Hold	8–10	12–14	3–5%	< 20 ft	$\delta_e \approx 20^\circ$
Lateral Sep.	–	8–10	8–12%	< 2 m	$\phi \approx 35^\circ$
Vertical Sep.	–	9–11	10–12%	< 3 m	$\theta \approx 18^\circ$
Forward Sep.	–	6–9	–	20–40 m RMS	err. max 80–120 m

Tabella 1: Metriche prestazionali degli autopiloti in simulazione.

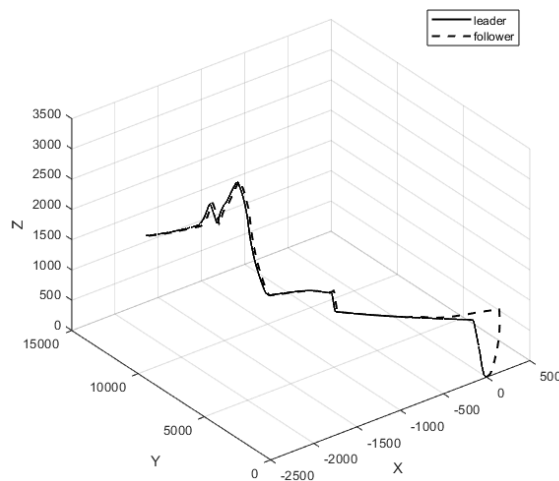


Figura 28: Mission Trajectory - 3D

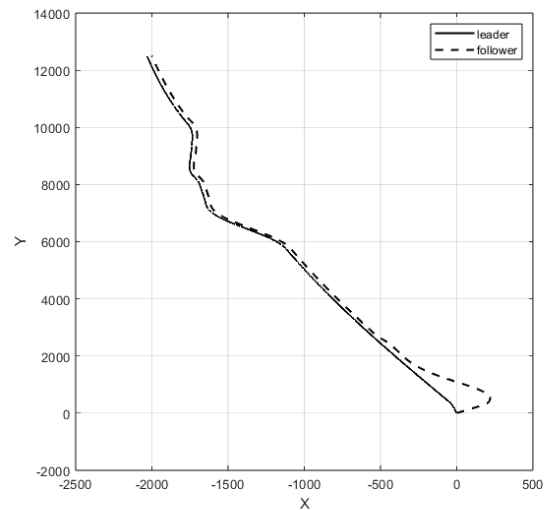


Figura 29: Mission Trajectory - Top

La traiettoria complessiva di missione, mostrata nelle Figure 28 e 29, evidenzia come il follower riesca a replicare con buona fedeltà il profilo del leader. Nella vista tridimensionale si osservano discrepanze limitate ai soli transitori, con rapidi riallineamenti che confermano la stabilità del sistema. La vista dall'alto mette in luce un errore longitudinale contenuto entro margini accettabili, a riprova dell'efficacia dei controllori di separazione nell'assicurare coesione durante l'intera missione.

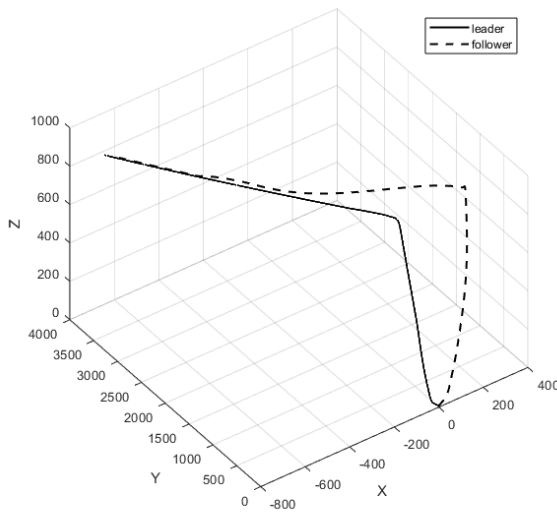


Figura 30: Manovra di decollo e avvicinamento.

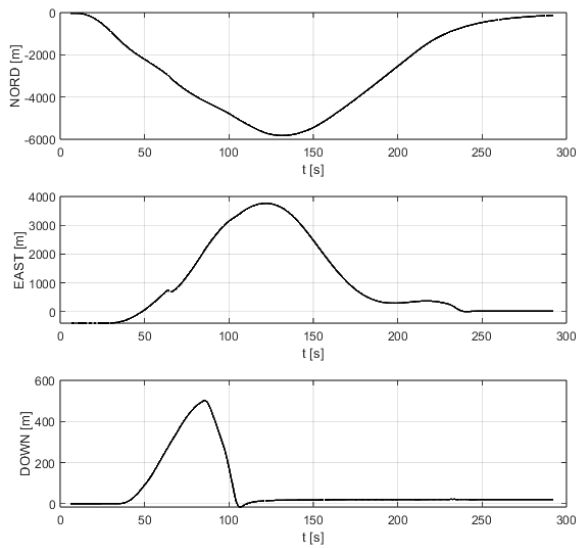


Figura 31: NED decollo e avvicinamento.

Nella fase di decollo e avvicinamento (Figure 30 e 31), il follower riproduce correttamente la sequenza di salita del leader. Nei grafici NED si nota un transitorio iniziale più marcato lungo l'asse verticale, legato alla fase critica di transizione dal rullaggio al volo, ma il sistema converge rapidamente al riferimento. Le traiettorie Nord ed Est risultano invece ben sovrapposte, a conferma di un inseguimento stabile in orizzontale.

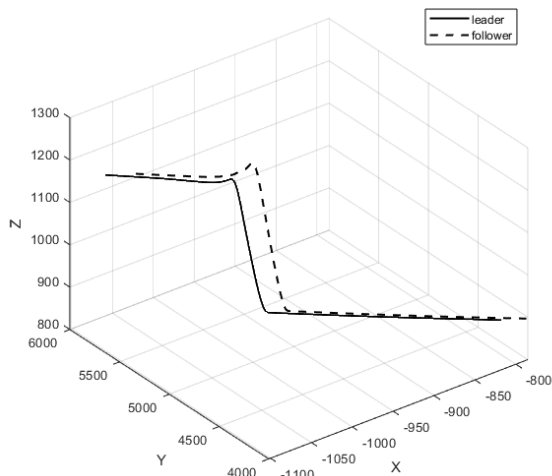


Figura 32: Manovra di richiamata.

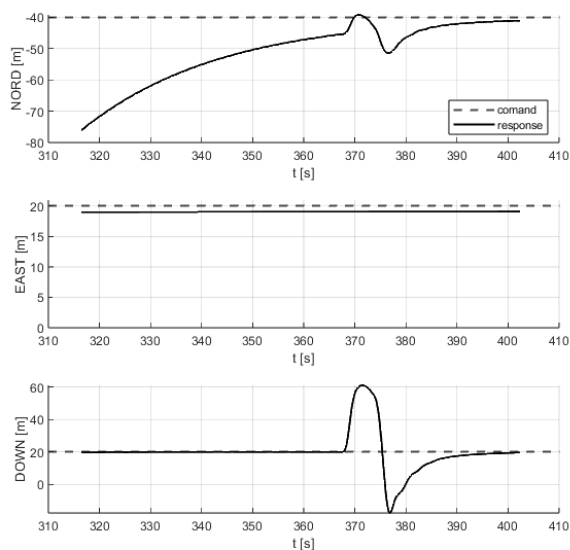


Figura 33: NED richiamata.

Durante la manovra di richiamata (Figure 32 e 33), caratterizzata da un rapido cambiamento di quota, il follower dimostra prontezza di risposta. Nei grafici NED l'errore più evidente si manifesta sul canale Nord, dove si osserva uno scostamento temporaneo, seguito da un rapido riallineamento. I canali Est e Down presentano deviazioni trascurabili, indicando una buona robustezza dei controllori anche in condizioni dinamiche più sollecitanti.

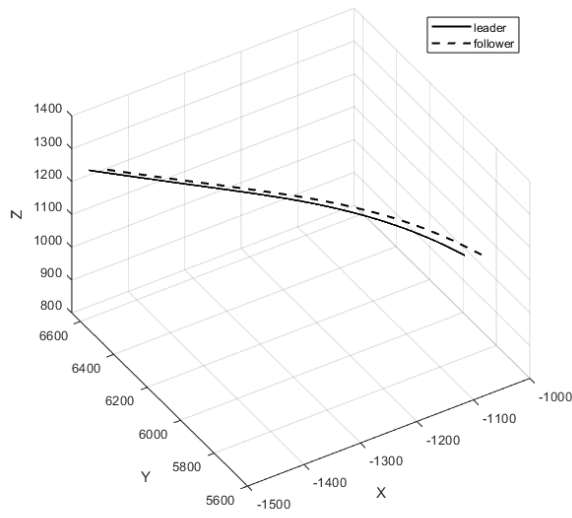


Figura 34: Manovra di virata.

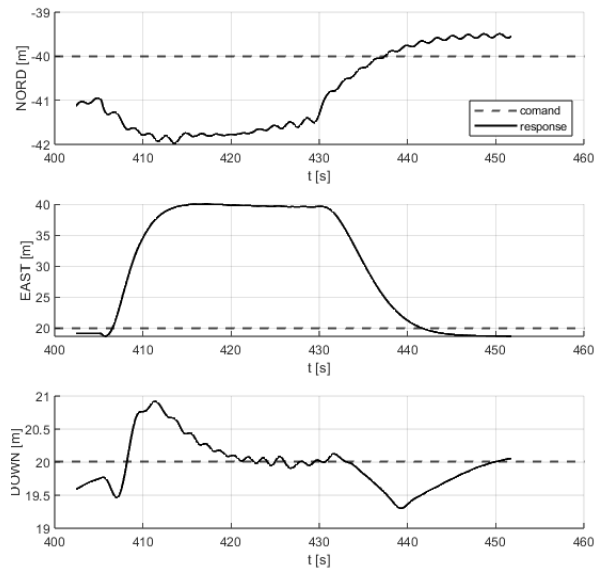


Figura 35: NED virata.

La virata orizzontale (Figure 34 e 35) conferma la capacità del follower di mantenere la posizione relativa durante variazioni di rotta. Il grafico tridimensionale mostra una traiettoria ben aderente a quella del leader, mentre i grafici NED evidenziano un errore iniziale sull'asse Est di pochi metri, recuperato senza oscillazioni persistenti. L'azione coordinata tra heading e bank hold garantisce la stabilizzazione sulla nuova direzione in tempi ridotti.

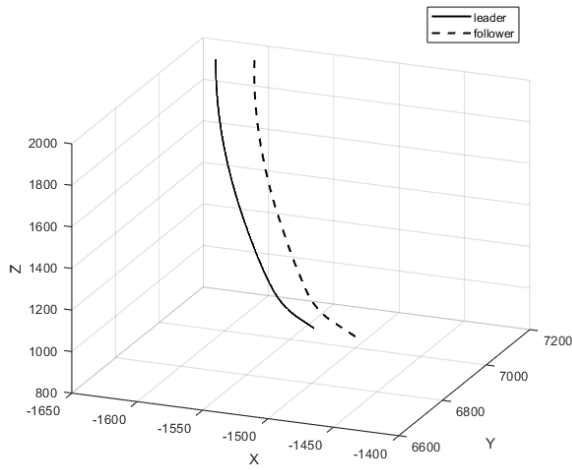


Figura 36: Manovra di virata ascendente.

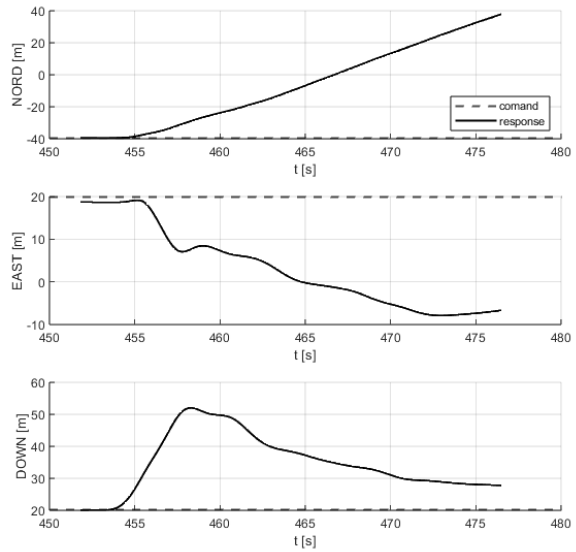


Figura 37: NED virata ascendente.

La virata ascendente (Figure 36 e 37), che combina simultaneamente variazioni di direzione e quota, rappresenta una manovra piú complessa. In questo scenario il follower presenta un overshoot verticale piú evidente nella fase iniziale, che viene tuttavia compensato in maniera efficace, riportando l'assetto entro i valori desiderati. Gli errori lungo gli assi Nord ed Est rimangono contenuti e non compromettono la stabilità della formazione. Tale risultato dimostra la robustezza del sistema anche in condizioni tridimensionali critiche.

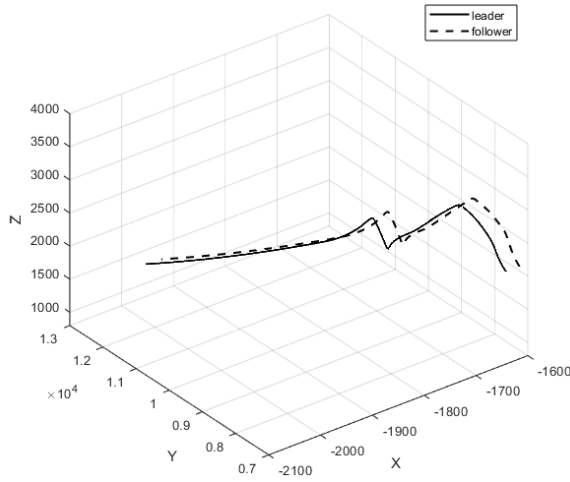


Figura 38: Comandi manuali.

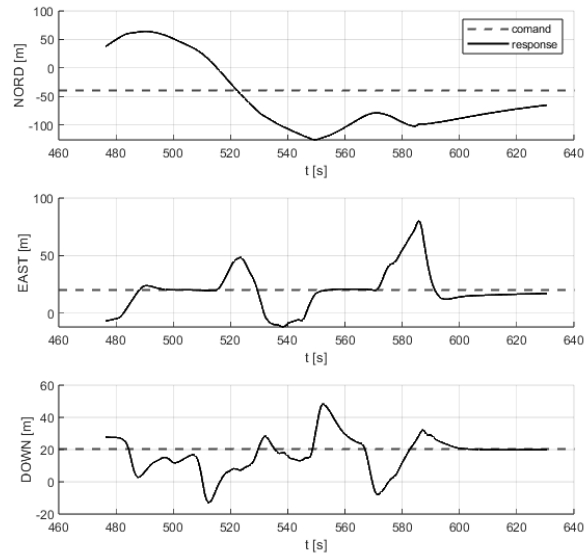


Figura 39: NED comandi manuali.

I comandi manuali (Figure 38 e 39) costituiscono una prova di stress per l'architettura di controllo. In questo caso, le traiettorie risultano meno regolari a causa della natura non ottimizzata degli input. Nei grafici NED emergono oscillazioni più significative nei tre canali, con deviazioni ripetute dai riferimenti. Nonostante ciò, il sistema è in grado di mantenere la stabilità complessiva, impedendo divergenze eccessive e preservando la sicurezza della formazione. Questo esperimento evidenzia la resilienza dell'architettura anche in presenza di condizioni di pilotaggio non ideali.

5.0.2 Caso perturbato

Dopo l'analisi del caso nominale, si é valutata la risposta degli autopiloti in presenza di condizioni meteorologiche avverse. Lo scenario perturbato includeva vento trasversale con raffiche variabili (0–30 nodi), turbolenza moderata e un wind shear localizzato a bassa quota. Tali disturbi rappresentano una condizione critica per la stabilità del velivolo, soprattutto nelle fasi di decollo e atterraggio, e mettono in evidenza la robustezza delle leggi di controllo implementate.

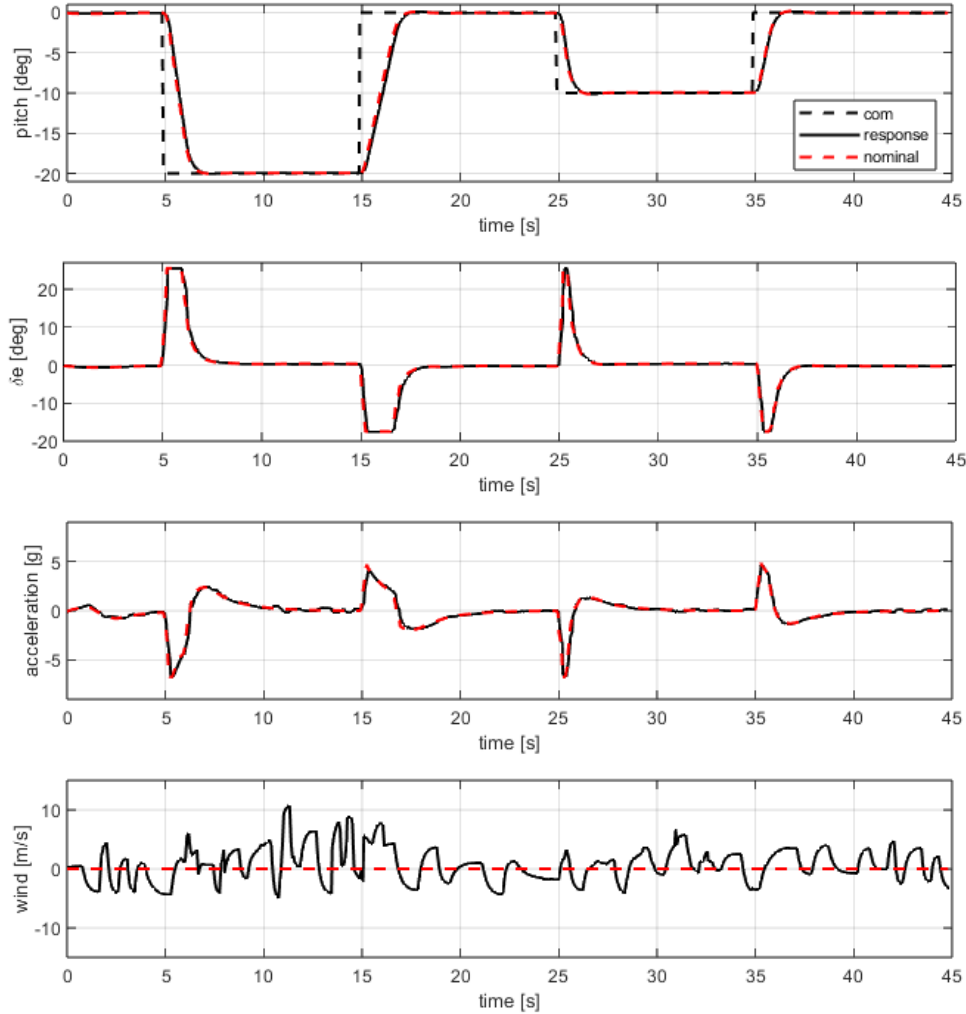


Figura 40: Grafico Pitch hold.

Il controllore di beccheggio ha mostrato una buona capacità di mantenere l'assetto anche in presenza di raffiche improvvise (Figure 40). La struttura *lead-lag* ha permesso di reagire rapidamente alle variazioni indotte dalla turbolenza, limitando l'oscillazione del pitch entro valori accettabili. Si osservano transienti più accentuati rispetto al caso non perturbato, ma l'errore a regime rimane contenuto ($e_{\infty} < 0.5^{\circ}$).

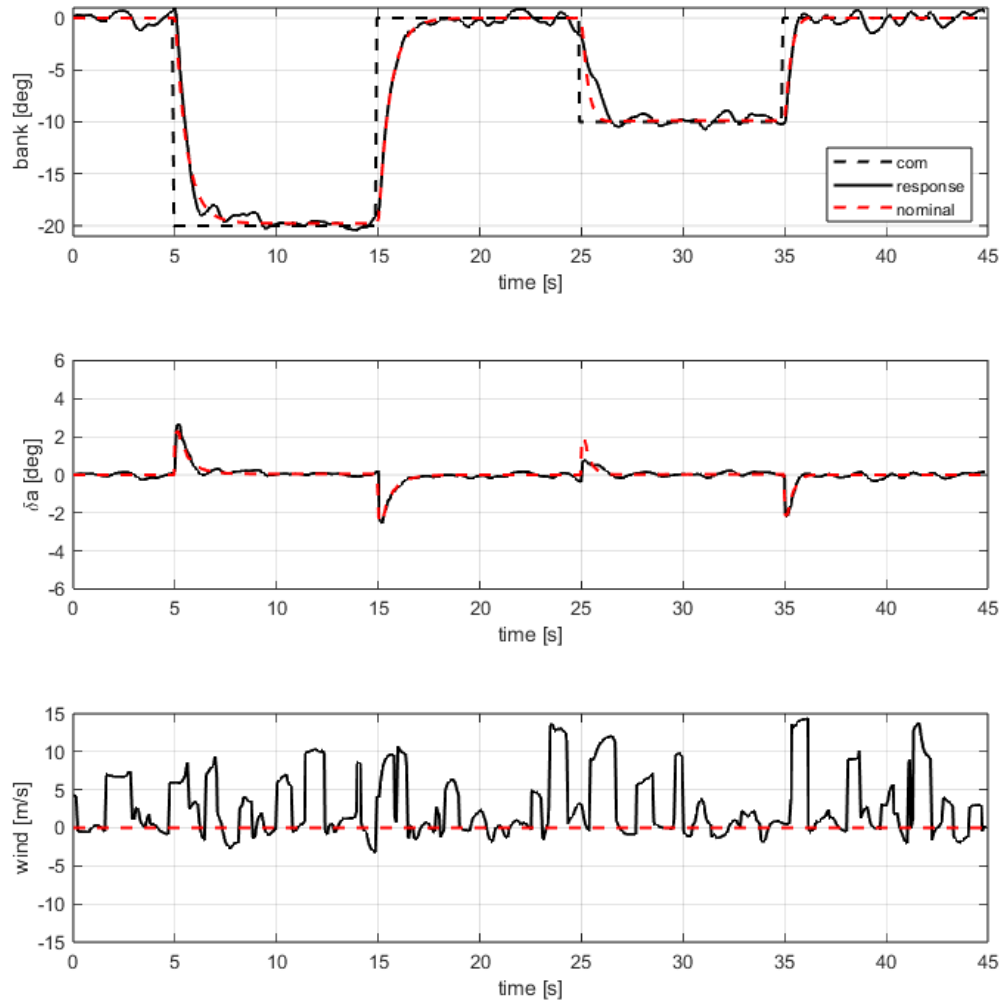


Figura 41: Grafico Bank Hold.

Durante la turbolenza, il canale di rollio é stato quello maggiormente sollecitato. Il controllore *Bank Hold* ha contrastato le deviazioni laterali generate dal vento trasversale, imponendo micro-correzioni continue sugli alettoni (Figure 41). La sovraelongazione non supera il 5% e l'angolo di rollio massimo resta in linea con i limiti operativi ($\delta_a \approx 3^\circ$), a conferma della stabilit  del regolatore.

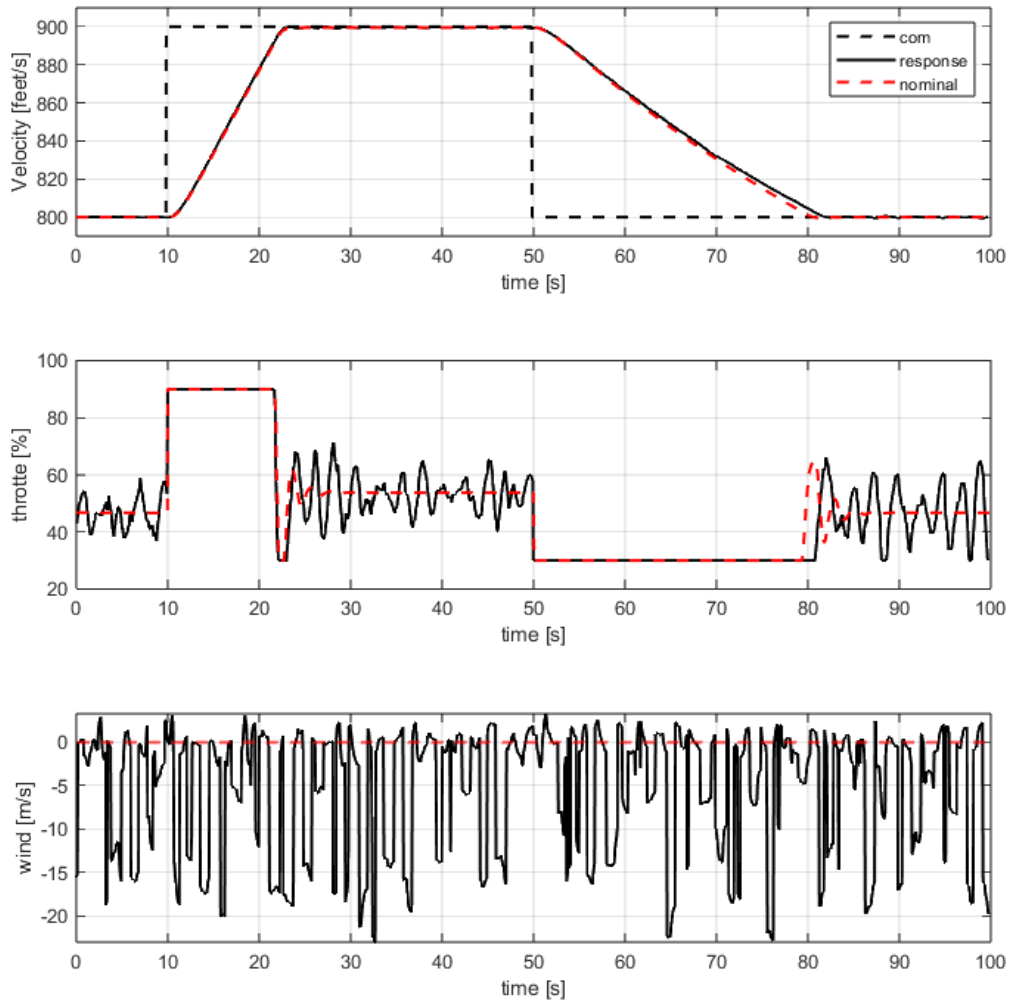


Figura 42: Grafico autothrottle.

Il canale di velocità longitudinale ha evidenziato la maggiore criticità in corrispondenza dell'ingresso nel wind shear, quando la velocità indicata ha subito variazioni repentine. L'autothrottle ha compensato con un rapido incremento della manetta, saturando vicino al 90%, ma è riuscito a ristabilizzare il regime in circa 25–30 s. Nonostante i transitori prolungati, l'errore a regime resta inferiore a 2 ft/s (Figure 42).

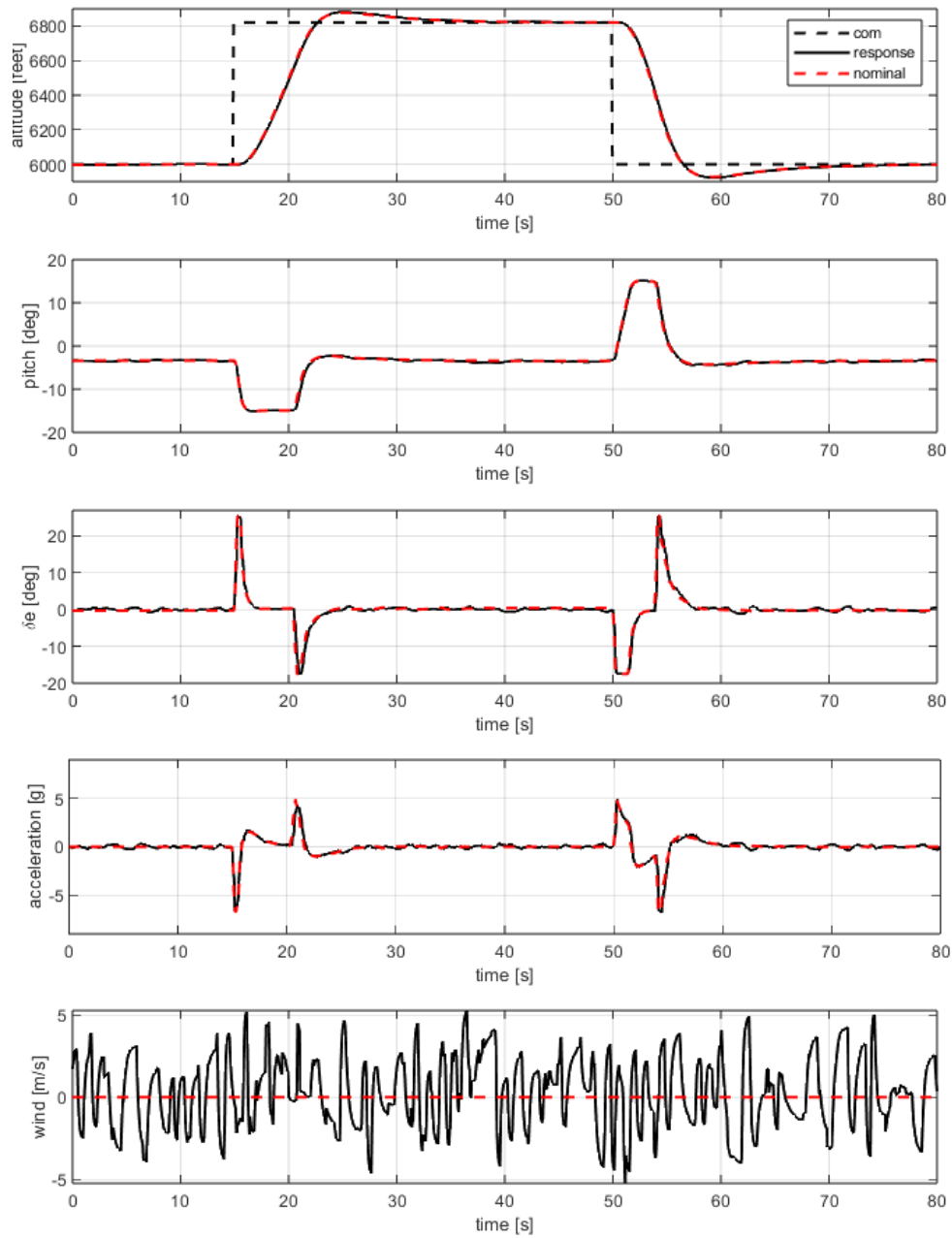


Figura 43: Grafico altitude hold.

Le variazioni verticali indotte dalla turbolenza hanno comportato oscillazioni della quota, corrette dal controllore con comandi aggressivi sull'elevatore (Figure 43). La risposta resta comunque entro i valori nominali ($t_s \approx 12\text{--}14$ s, $e_\infty < 20$ ft), dimostrando la robustezza della struttura PID con retroazione sul pitch.

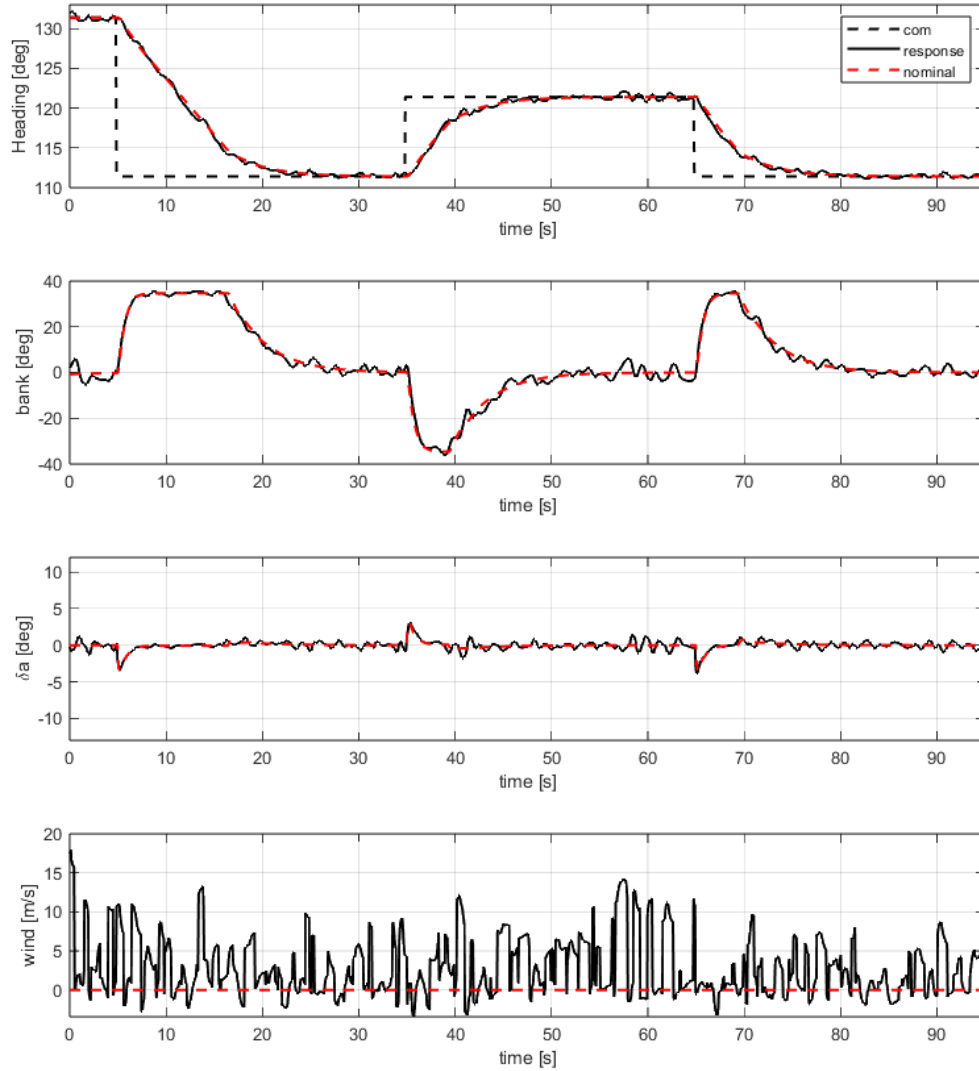


Figura 44: Grafico heading hold.

Il controllore di prua ha mantenuto la rotta desiderata anche sotto l'azione di vento trasversale. L'anello esterno (heading) ha generato banche di riferimento maggiori rispetto al caso nominale, raggiungendo $|\phi| \approx 35^\circ$, ma senza compromettere la stabilità (Figure 44). L'anti-windup ha impedito accumuli significativi, garantendo un errore finale inferiore a 0.5° .

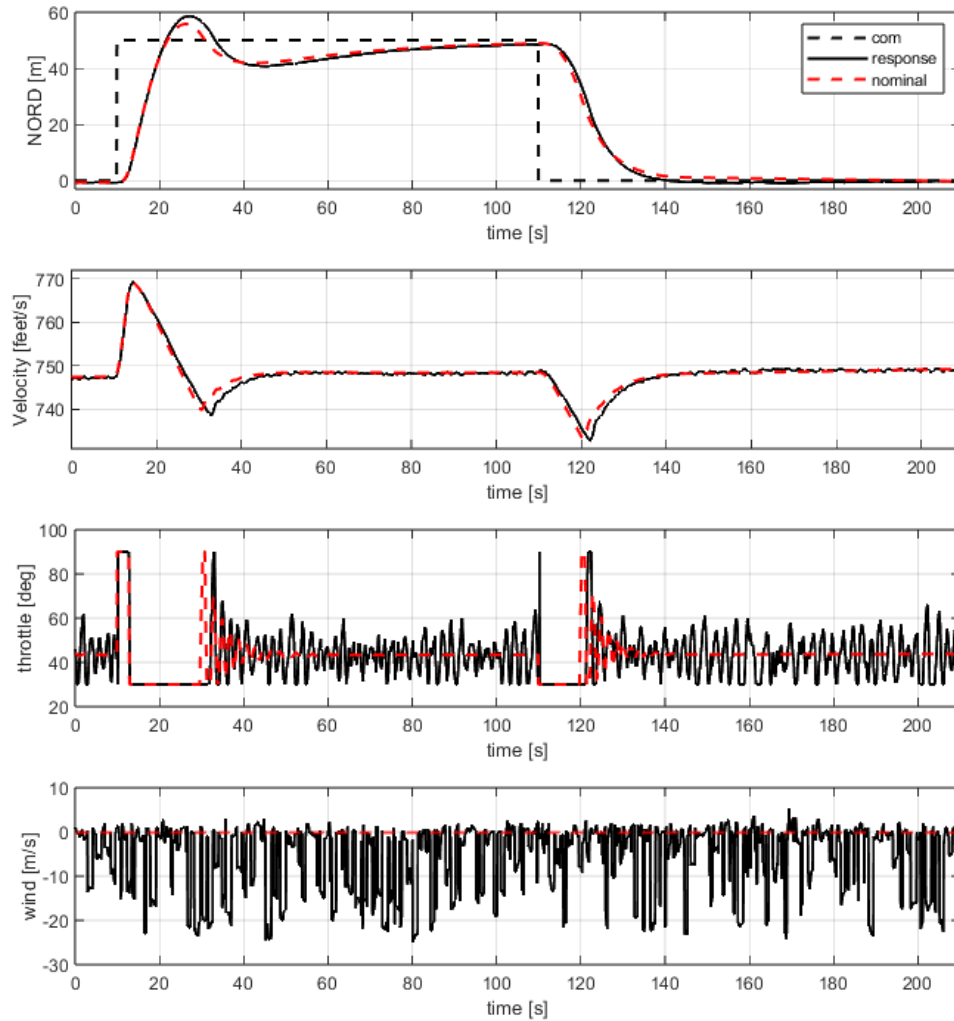


Figura 45: Grafico forward separation autopilot.

Il canale longitudinale ha mostrato oscillazioni dovute al wind shear, con errori massimi durante le manovre rapidi pari a 100 m circa. Tuttavia, il riallineamento con la traiettoria del leader é avvenuto entro 6–9 s, dimostrando che la coordinazione tra autothrottle e assetto longitudinale rimane efficace anche in condizioni perturbate (Figure 45).

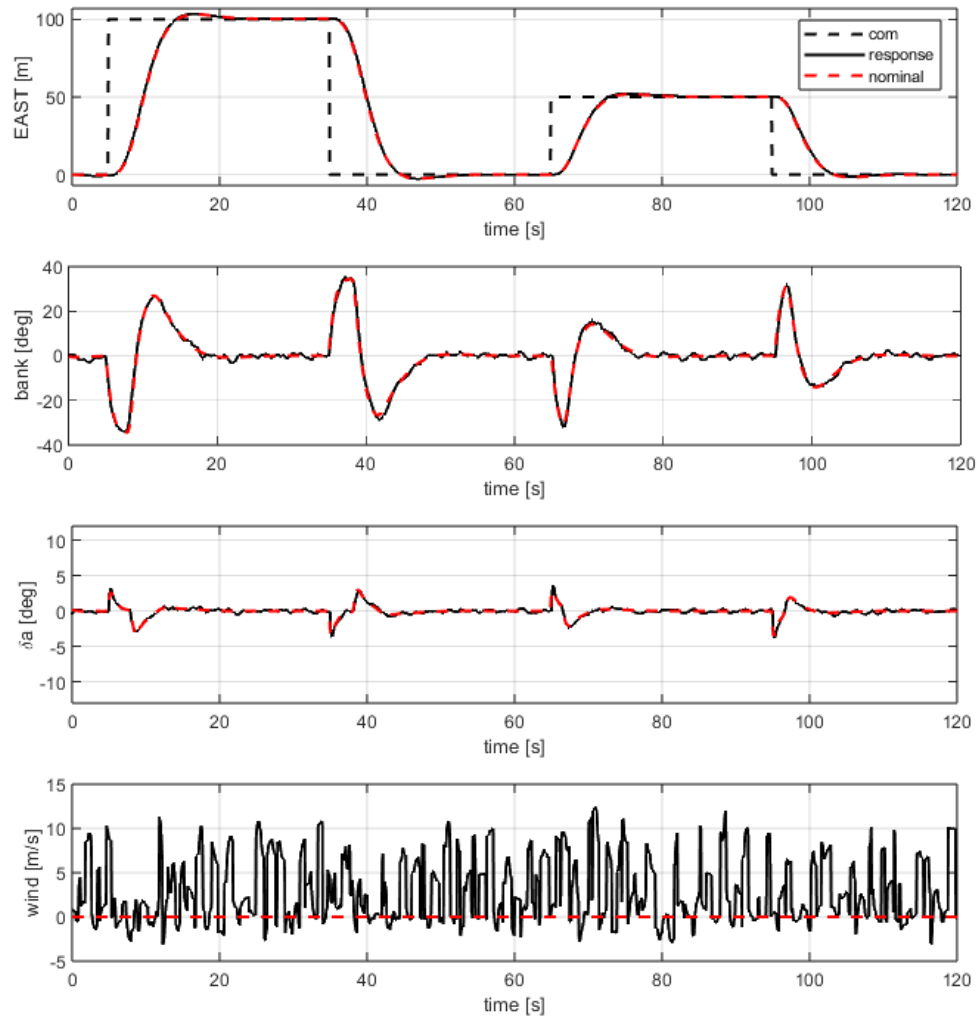


Figura 46: Grafico lateral separation autopilot.

Il mantenimento della separazione laterale dal leader é risultato il piú sensibile alle raffiche di vento trasversale. Il controllore ha generato correzioni rapide in rollio, con picchi di bank comparabili a quelli del caso nominale ($30\text{--}35^\circ$)(Figure 46). La distanza laterale é stata recuperata con un tempo di assestamento inferiore ai 10 s e con errore residuo < 2 m.

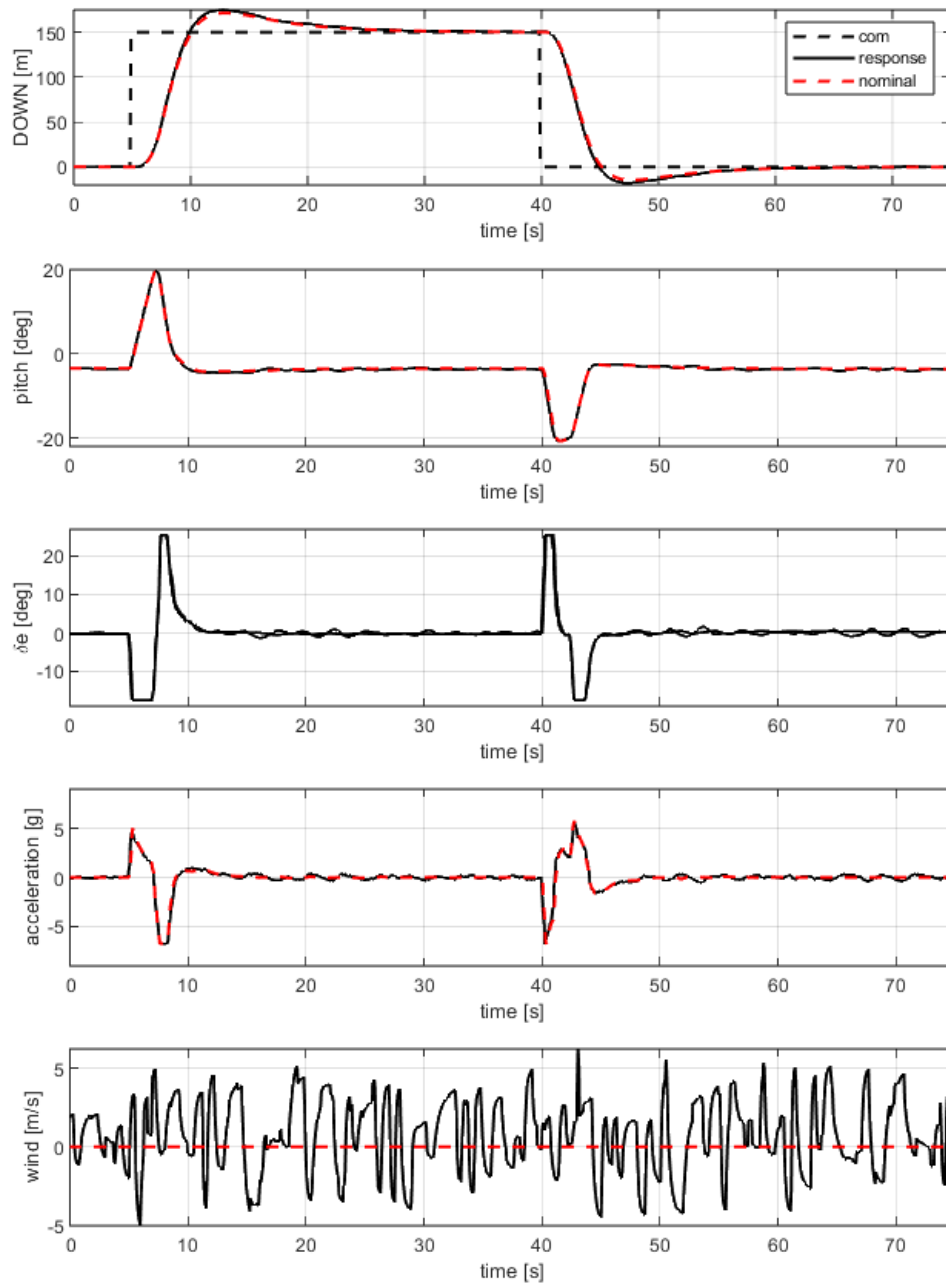


Figura 47: Grafico vertical separation autopilot.

Le perturbazioni verticali hanno introdotto oscillazioni nella distanza dal leader, ma il controllore ha reagito con variazioni di pitch ($|\theta| \approx 18^\circ$) e deflessioni agli elevatori fino a 18° (Figure 47). L'errore massimo non ha superato i 3 m e l'assestamento é avvenuto in circa 10 s.

Nel complesso, i risultati numerici riportati in Tabella 1 rimangono validi anche in condizioni perturbate, pur con transitori più energici e maggiore impegno degli attuatori. L'analisi conferma la robustezza dei controllori progettati, in grado di garantire la sicurezza operativa del volo in formazione anche in presenza di condizioni atmosferiche avverse.

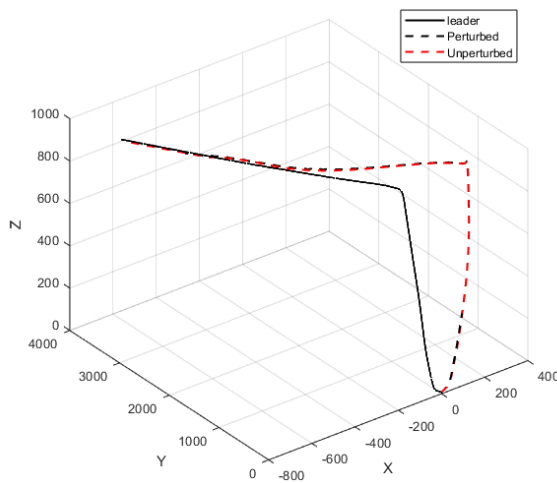


Figura 48: Manovra di decollo e avvicinamento perturbati.

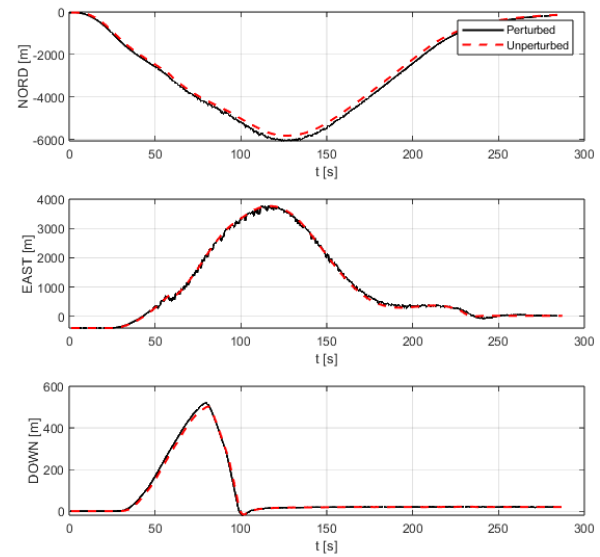


Figura 49: NED decollo e avvicinamento perturbati.

La manovra di decollo e avvicinamento perturbata (Figure 48 e 49) mette in evidenza l'efficacia del sistema di controllo in presenza di disturbi. Il confronto tra traiettorie perturbate e non perturbate mostra come il follower riesca a mantenere la separazione anche in condizioni non nominali. Nei grafici NED, gli scostamenti sono più marcati nella fase iniziale, in particolare lungo l'asse verticale, ma le traiettorie convergono rapidamente, segno di un buon comportamento correttivo. Gli assi Nord ed Est restano sostanzialmente sovrapposti, garantendo una traiettoria stabile e fedele.

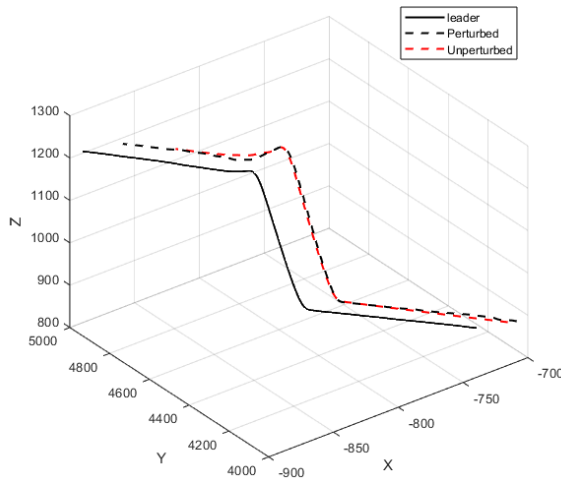


Figura 50: Manovra di richiamata perturbata.

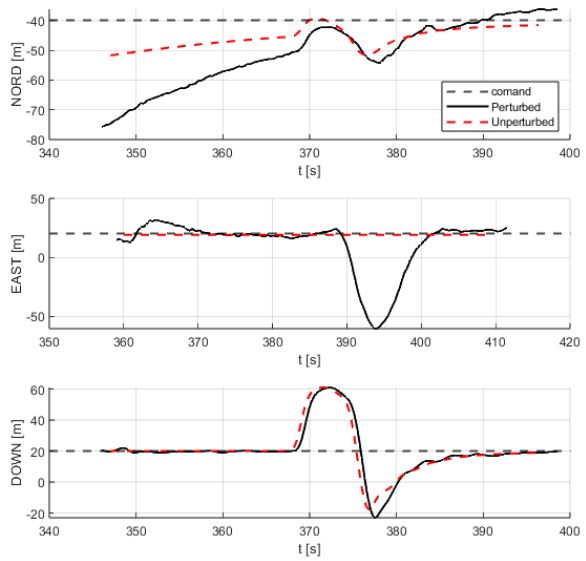


Figura 51: NED richiamata perturbata.

Durante la manovra di richiamata perturbata (Figure 50 e 51), il sistema mostra una maggiore sensibilità ai disturbi, specialmente lungo l'asse Nord. Il follower perturbato evidenzia un errore più ampio e persistente rispetto al caso non perturbato, ma riesce comunque a seguire la variazione di quota imposta dal leader. Sul canale Down, il comportamento rimane controllato, con deviazioni simili tra scenario perturbato e non perturbato, a dimostrazione della robustezza del controllo in quota. Il canale Est evidenzia invece scostamenti ridotti, mantenendo una buona coerenza con il profilo desiderato.

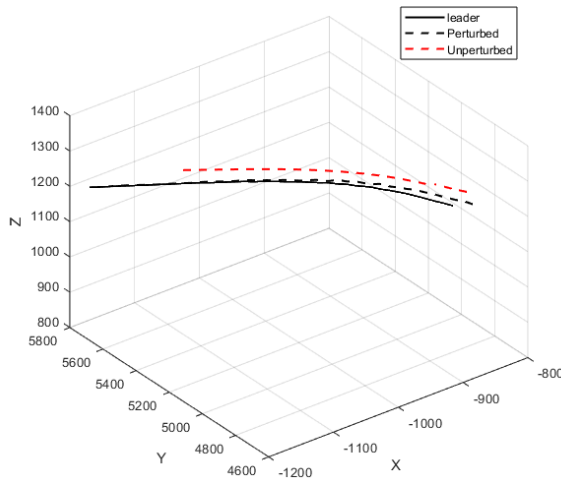


Figura 52: Manovra di Virata perturbata.

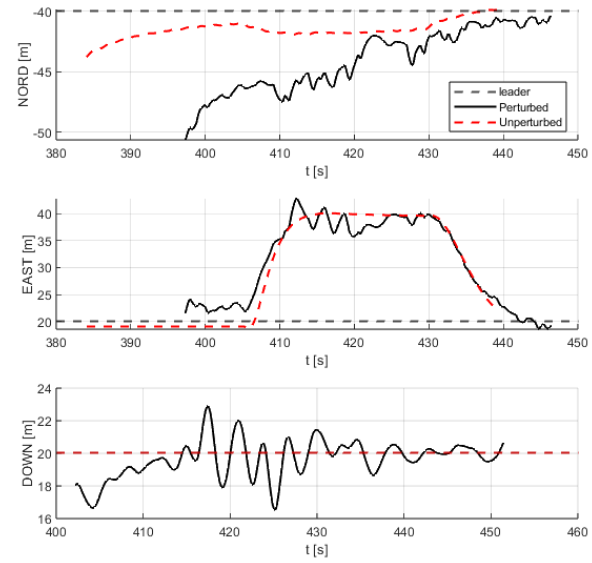


Figura 53: NED virata perturbata.

La virata perturbata (Figure 52 e 53) evidenzia gli effetti più rilevanti dei disturbi sul mantenimento della traiettoria. Nella vista 3D si osserva uno scostamento sistematico tra traiettoria perturbata e non perturbata, in particolare lungo l'asse verticale. Nei grafici NED, il follower perturbato mostra oscillazioni marcate nei tre canali, con deviazioni evidenti sul canale Nord e variazioni significative sul canale Down. Nonostante queste perturbazioni, la traiettoria rimane complessivamente vicina al profilo del leader, indicando che il sistema riesce a contenere gli effetti dei disturbi senza comprometterne la stabilità globale.

5.1 Conclusioni

Il progetto sviluppato ha evidenziato la concreta fattibilità di implementare e validare, all'interno di un ambiente di simulazione, un sistema di autopiloti in grado di gestire missioni cooperative tra velivoli con e senza pilota, in uno scenario di *Manned-Unmanned Teaming* (MUM-T).

Le prove condotte hanno dimostrato, in primo luogo, la robustezza dell'architettura di simulazione: la sincronizzazione in rete tra velivolo leader e velivolo follower si è dimostrata stabile e affidabile non solo in condizioni operative standard, ma anche in presenza di perturbazioni esterne introdotte artificialmente. Parallelamente, i controllori implementati, basati su strutture PID e compensatori di tipo lead-lag, hanno mostrato un comportamento coerente con le aspettative teoriche, garantendo risposte pronte e stabili e mantenendo la formazione anche in scenari caratterizzati da disturbi atmosferici significativi.

Un ulteriore aspetto emerso riguarda la scalabilità del sistema. L'architettura sviluppata è risultata infatti sufficientemente flessibile da permettere un'estensione del numero di velivoli simulati, nonché l'integrazione di moduli software e sensori aggiuntivi senza richiedere modifiche invasive all'infrastruttura di base.

Complessivamente, i risultati ottenuti confermano che la piattaforma proposta costituisce una base solida e adattabile per lo studio di missioni multi-velivolo, rappresentando un punto di partenza concreto per la futura introduzione di capacità autonome più avanzate e per applicazioni operative in scenari complessi e realistici.

5.2 Sviluppi futuri

Il lavoro svolto può essere ulteriormente sviluppato lungo diverse direttrici di ricerca e innovazione tecnologica, capaci di aumentare il livello di autonomia e l'efficacia operativa del sistema.

Un primo ambito di approfondimento riguarda l'introduzione di algoritmi di *collision avoidance*. In scenari caratterizzati dalla presenza di più UAV all'interno di spazi aerei condivisi, risulta fondamentale dotare i velivoli di strategie predittive in grado di anticipare situazioni di potenziale conflitto. L'adozione di approcci come il *Model Predictive Control* (MPC) consentirebbe di calcolare traiettorie sicure in tempo reale, mentre l'integrazione di logiche cooperative ispirate alla *swarm intelligence* permetterebbe di coordinare le manovre evasive sulla base delle informazioni provenienti da sensori e comunicazioni inter-veicolo.

Un secondo filone di ricerca riguarda la *navigazione avanzata*. In questo contesto, lo sviluppo di algoritmi di *sensor fusion* più sofisticati, basati su filtri come l'Extended Kalman Filter (EKF) o l'Unscented Kalman Filter (UKF), consentirebbe di migliorare l'accuratezza e l'affidabilità della stima dello stato del velivolo. In prospettiva, l'integrazione di tecniche di *terrain-aware navigation* permetterebbe inoltre di affrontare scenari complessi e non strutturati, sfruttando mappe dinamiche costantemente aggiornate per adattare il profilo di missione alle condizioni operative.

Infine, un ulteriore sviluppo riguarda gli aspetti di *guidance* e *mission planning*. La progettazione di logiche adattive, capaci di gestire missioni multi-velivolo con obiettivi dinamici, potrebbe essere potenziata mediante l'impiego di algoritmi di *reinforcement learning* e tecniche di pianificazione multi-obiettivo. Tali approcci renderebbero possibile il *re-planning* in tempo reale, consentendo al sistema di rispondere in maniera efficace a eventi imprevisti, co-

me condizioni meteorologiche variabili o perdita temporanea delle comunicazioni, aumentando così la resilienza e l'affidabilità complessiva della missione.

Riferimenti bibliografici

- [1] J. Luo, Y. Tian, and Z. Wang, “Research on unmanned aerial vehicle path planning,” *Drones*, vol. 8, no. 2, p. 51, 2024.
- [2] L. Bai, Z. Zhao, X. Meng, Y. Wang, Q. Rao, and X. Deng, “Research on uav formation simulation and evaluation technology,” in *Proceedings of the Chinese Flight Test Establishment Symposium*, 2018, pp. 1–6.
- [3] *HOTAS Warthog: Hardware User Manual*, Guillemot Corporation S.A., 2010, replica joystick and throttle system manual.
- [4] *Prepar3D SDK v6 Overview*, Lockheed Martin, 2024, prepar 3d SDK.
- [5] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, 3rd ed. John Wiley & Sons, 2016.
- [6] Q. Qiu and S. Wu, “Robust formation flight control system design for multi-uavs,” *International Journal of Control, Automation, and Systems*, pp. 1–8, 2010.
- [7] P. Zhang and J. Liu, “On new uav flight control system based on kalman & pid,” in *Proceedings of International Conference on UAV Systems*, 2012, pp. 1–6.
- [8] G. Testolin, “Controllori pid e tecniche anti wind-up,” Master’s thesis, Università degli Studi di Padova, 2010.
- [9] A. Bemporad, “Controllo di sistemi con saturazione: tecniche di anti-windup,” 2007, dispense del corso di Controllo Digitale, A.A. 2007/08.
- [10] F. Haugen, “Derivation of a discrete-time lowpass filter,” TechTeach, Tech. Rep., 2008.

A Documentazione Aggiuntiva

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:Addon for Simconnect linked to Prepar3D
5 //      It has to be attached to a C2 Plane
6 //-----
7
8 #include "headers/globalvar.h"
9 #include "headers/readwrite.h"
10 #include "headers/dispatchfun.h"
11 #include "headers/autopilots.h"
12 #include "headers/miscellaneous.h"
13
14 //title of the addon
15 const char* TITLE_STRING = "C2 PLANE";
16
17 //main function of the addon
18 int __cdecl _tmain(int argc, _TCHAR* argv[])
19 {
20
21     // Apertura del simconnect
22     if (SUCCEEDED(SimConnect_Open(&hSimConnect, TITLE_STRING, NULL, 0, 0, 0)))
23     {
24         //connesso al simulatore
25         printf("Connected to Prepar3D\n");
26
27         //lettura file configs.txt
28         LetturaConfigs();
29
30         //richiesta test
31         printf("Vuoi eseguire un test dei controlli? ");
32         flag_test = AskYesNo();
33         if (flag_test==1)
```

```
34 {
35     printf("Su cosa vuoi eseguire il test?\n");
36     flag_test = SelectTest();
37
38 }
39
40 //richiesta di creazione e apertura outputfile
41 printf("Vuoi salvare i dati della simulazione? ");
42 flag_stampa=AskYesNo();
43 if (flag_stampa == 1)
44 {
45     CreaFile();
46     ApriFile();
47 }
48
49 //ciclo principale dell'applicazione
50 while (0 == flag_quit)
51 {
52     //funzione che gestisce gli eventi del simulatore
53     SimConnect_CallDispatch(hSimConnect, MyDispatchProc, NULL);
54
55     //controllo se il simulatore è in pausa o meno
56     if (flag_isrunning == 1)
57     {
58         //request data on user
59         hr = SimConnect_RequestDataOnSimObjectType(hSimConnect, REQUEST_1, DEFINITION_1, 0,
60             SIMCONNECT_SIMOBJECT_TYPE_USER);
61
62         //request data on other plane
63         hr = SimConnect_RequestDataOnSimObjectType(hSimConnect, REQUEST_0, DEFINITION_1, 100000,
64             SIMCONNECT_SIMOBJECT_TYPE_AIRCRAFT);
65
66         //ceck if is on ground and correct the position
```

```
65 IsOnGround();
66
67 //controllo del carrello e retrazione
68 GearCheck();
69
70 // se al suolo autopilota C2, manovra di decollo
71 if (flag_initialgroundcheck == 1)
72 {
73     switch (flag_decollo)
74     {
75
76     //inizializzazione
77     case 1:
78     {
79         // manetta al 90%
80         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
            EVENT_THROTTLE_SET, 14743, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
81
82         // rimuovo il freno di stazionamento
83         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
            EVENT_PARKING_BRAKES_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
84         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
            EVENT_RUDDER_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
85         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
            EVENT_AILERON_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
86         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
            EVENT_ELEVATOR_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
87
```

```
88 //passo alla prossima fase
89 flag_decollo = 2;
90
91 //invio il comando al drone per decollare
92 SendCommandTakeOff();
93 }
94 break;
95
96 //fase di rullaggio
97 case 2:
98 {
99     // manetta al 90%
100     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
        EVENT_THROTTLE_SET, 14743, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
        7
        7
101
102 //attivo l'autopilota di heading con controllo di timone
103 double rudder = HeadingTakeoff(initial_heading, UserPlane.heading);
104 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
        EVENT_RUDDER_SET, static_cast<DWORD>(round(rudder)),
        SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
        7
        7
105 //autopilota di controllo del pitch per tenere il velivolo a terra
106 double elevator = Pitchhold(0.8, UserPlane.pitch, UserPlane.pitchrate);
107 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
        EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator)),
        SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
        7
        7
108
109 //comando per richiamata
110 if (UserPlane.velocity > 300)
111 {
112     flag_decollo = 3;
113     // setto l'alettone a 0 (non servirà se implemento l'ari)
114     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
```

```
EVENT_RUDDER_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
    }
}
break;

// fase di richiamata e salita rettilinea
case 3:
{
    // manetta al 90%
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
EVENT_THROTTLE_SET, 14743, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

    //autopilota di mantenimento dell'heading con controllo di alettone
double aileron = Headinghold(initial_heading, UserPlane.heading, 10, UserPlane.bank,
UserPlane.rollrate);
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
EVENT_AILERON_SET, static_cast<DWORD>(round(aileron))),
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

    //autopilota di controllo della quota con controllo dell'equilibratore
double elevator = Pitchhold(-15, UserPlane.pitch, UserPlane.pitchrate);
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator))),
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

    //comando per virata di allineamento alla crociera
if (UserPlane.altitude > quota_crociera / 3)
{
    flag_decollo = 4;
}
}
```



```
166     UserPlane.pitchrate);
167     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
168     EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator)),
169     SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
170 }
171
172 //autopilota di mantenimento dell'heading con controllo di alettone
173 double aileron = Headinghold(heading_crociera, UserPlane.heading, 20, UserPlane.bank,
174     UserPlane.rollrate);
175 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
176     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
177     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
178
179 if (UserPlane.altitude < (quota_crociera + 10.0) && UserPlane.altitude > (quota_crociera -
180     10.0))
181 {
182     if (flag_isincruise == 0)
183     {
184         initial_simtime = UserPlane.simtime;
185         flag_isincruise = 1;
186     }
187     if (UserPlane.simtime - initial_simtime > 5)
188     {
189         flag_decollo = 5;
190         SendCommandReachC2();
191     }
192 }
193
194 }
195
196 else
197 {
198     flag_isincruise = 0;
199 }
200
201 }
```

```
192
193 //fase di crociera a velocità di crociera
194 if (flag_decollo == 5)
195 {
196     // imposto la velocità a 850 feet/sec
197     double throttle = Autothrottle(850, UserPlane.velocityZ, UserPlane.accelerationZ);
198     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
199     static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
200     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
201
202     //autopilota di controllo della quota con controllo dell'equilibratore
203     if (UserPlane.altitude < quota_crociera + pitchang - 250)
204     {
205         double elevator = Pitchhold(-15, UserPlane.pitch, UserPlane.pitchrate);
206         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
207         EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator)),
208         SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
209
210     }
211     else if (UserPlane.altitude > quota_crociera + pitchang + 250)
212     {
213         double elevator = Pitchhold(15, UserPlane.pitch, UserPlane.pitchrate);
214         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
215         EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator)),
216         SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
217
218     }
219     else
220     {
221         double elevator = Altitudehold(quota_crociera + pitchang, UserPlane.altitude, 15,
222         UserPlane.pitch, UserPlane.pitchrate);
223         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
224         EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator)),
225         SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
226     }
227 }
```

```

216
217 //autopilota di mantenimento dell'heading con controllo di alettone
218 double aileron = Headinghold(heading_crociera+bankang, UserPlane.heading, 20,
219 UserPlane.bank, UserPlane.rollrate);
220 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
221 static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
222 SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
223
224 }
225
226 //test pitchhold
227 else if(flag_test == 101)
228 {
229     double throttle = Autothrottle(800, UserPlane.velocityZ, UserPlane.accelerationZ);
230     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
231 static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
232 SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
233     double aileron = Bankhold(0, UserPlane.bank, UserPlane.rollrate);
234     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
235 static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
236 SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
237
238     if (flag_initial == 0 && UserPlane.simtime > 0)
239     {
240         initial_simtime = UserPlane.simtime;
241         flag_initial = 1;
242     }
243     if (UserPlane.simtime - initial_simtime < 5.0)
244     {
245         pitchang = 0;
246         double elevator = Pitchhold(pitchang, UserPlane.pitch, UserPlane.pitchrate);
247         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
248 static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,

```

```
241         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
242     }  
243     else if (UserPlane.simtime - initial_simtime < 15.0)  
244     {  
245         pitchang = -20;  
246         double elevator = Pitchhold(pitchang, UserPlane.pitch, UserPlane.pitchrate);  
247         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2  
248         static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2  
249         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
250     }  
251     else if (UserPlane.simtime - initial_simtime < 25.0)  
252     {  
253         pitchang = 0;  
254         double elevator = Pitchhold(pitchang, UserPlane.pitch, UserPlane.pitchrate);  
255         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2  
256         static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2  
257         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
258     }  
259     else if (UserPlane.simtime - initial_simtime < 35.0)  
260     {  
261         pitchang = -10;  
262         double elevator = Pitchhold(pitchang, UserPlane.pitch, UserPlane.pitchrate);  
263         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2  
264         static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2  
265         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
266     }  
267     else if (UserPlane.simtime - initial_simtime < 45.0)  
268     {  
269         pitchang = 0;  
270         double elevator = Pitchhold(pitchang, UserPlane.pitch, UserPlane.pitchrate);
```

```
267 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2
    static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

268 }
269 else
270 {
271     exit(0);
272 }
273 }
274 }
275
276 //test per il bankhold
277 else if (flag_test == 102)
278 {
279     double throttle = Autothrottle(800, UserPlane.velocityZ, UserPlane.accelerationZ);
280     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET, 2
        static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

281
282     double elevator = Altitudehold(6000, UserPlane.altitude, 20, UserPlane.pitch, 2
        UserPlane.pitchrate);
283     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2
        static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

284
285     if (flag_initial == 0 && UserPlane.simtime > 0)
286     {
287         initial_simtime = UserPlane.simtime;
288         flag_initial = 1;
289     }
290     if (UserPlane.simtime - initial_simtime < 5.0)
291     {
292         bankang = 0;
```

```
293 double aileron = Bankhold(bankang, UserPlane.bank, UserPlane.rollrate);
294 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
    static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
295 }
296 else if (UserPlane.simtime - initial_simtime < 15.0)
297 {
298     bankang = -20;
299     double aileron = Bankhold(bankang, UserPlane.bank, UserPlane.rollrate);
300     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
    static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
301 }
302 else if (UserPlane.simtime - initial_simtime < 25.0)
303 {
304     bankang = 0;
305     double aileron = Bankhold(bankang, UserPlane.bank, UserPlane.rollrate);
306     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
    static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
307 }
308 else if (UserPlane.simtime - initial_simtime < 35.0)
309 {
310     bankang = -10;
311     double aileron = Bankhold(bankang, UserPlane.bank, UserPlane.rollrate);
312     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
    static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
313 }
314 else if (UserPlane.simtime - initial_simtime < 45.0)
315 {
316     bankang = 0;
317     double aileron = Bankhold(bankang, UserPlane.bank, UserPlane.rollrate);
```

```

318 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
    static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
319 }
320 else
321 {
322     exit(0);
323 }
324
325 }
326
327 //test altitude hold
328 else if (flag_test == 103)
329 {
330     double throttle = Autothrottle(800, UserPlane.velocityZ, UserPlane.accelerationZ);
331     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
        static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
332     double aileron = Bankhold(0, UserPlane.bank, UserPlane.rollrate);
333     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
        static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
334
335     if (flag_initial == 0 && UserPlane.simtime > 0)
336     {
337         initial_simtime = UserPlane.simtime;
338         flag_initial = 1;
339     }
340     if (UserPlane.simtime - initial_simtime < 35.0)
341     {
342         altitude = 6000;
343         double elevator = Altitudehold(altitude, UserPlane.altitude, 15, UserPlane.pitch,
            UserPlane.pitchrate);

```

```
344 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,   
    static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,   
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);   
   
345 }   
346   
347 else if (UserPlane.simtime - initial_simtime < 70.0)   
348 {   
349     altitude = 6000+250/0.3048;   
350   
351     double elevator = Altitudehold(altitude, UserPlane.altitude, 15, UserPlane.pitch,   
        UserPlane.pitchrate);   
352     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,   
        static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,   
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);   
   
353 }   
354   
355 else if (UserPlane.simtime - initial_simtime < 105.0)   
356 {   
357     altitude = 6000;   
358   
359     double elevator = Altitudehold(altitude, UserPlane.altitude, 15, UserPlane.pitch,   
        UserPlane.pitchrate);   
360     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,   
        static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,   
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);   
   
361 }   
362   
363 else   
364 {   
365     exit(0);   
366 }   
367   
368 }
```

```

369 //test heading hold
370 else if (flag_test == 104)
371 {
372     double throttle = Autothrottle(850, UserPlane.velocityZ, UserPlane.accelerationZ);
373     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
        static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
374
375     double elevator = Altitudehold(6000, UserPlane.altitude, 15, UserPlane.pitch,
        UserPlane.pitchrate);
376     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
        static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
377
378     if (flag_initial == 0 && UserPlane.simtime > 0)
379     {
380         initial_simtime = UserPlane.simtime;
381         flag_initial = 1;
382         testheading = UserPlane.heading;
383     }
384     if (UserPlane.simtime - initial_simtime < 5.0)
385     {
386         comtestheading = testheading;
387     }
388     double aileron = Headinghold(comtestheading, UserPlane.heading, 35, UserPlane.bank,
        UserPlane.rollrate);
389     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
        static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
390
391     }
392     else if (UserPlane.simtime - initial_simtime < 35.0)
393     {
        comtestheading = testheading +20;
    }

```

```
394
395
396     double aileron = Headinghold(comtestheading, UserPlane.heading, 35, UserPlane.bank,
397     UserPlane.rollrate);
398     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
399     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
400     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
401
402     }
403     else if (UserPlane.simtime - initial_simtime < 65.0)
404     {
405         comtestheading = testheading;
406
407         double aileron = Headinghold(comtestheading, UserPlane.heading, 35, UserPlane.bank,
408         UserPlane.rollrate);
409         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
410         static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
411         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
412
413     }
414     else if (UserPlane.simtime - initial_simtime < 95.0)
415     {
416         comtestheading = testheading +10;
417
418         double aileron = Headinghold(comtestheading, UserPlane.heading, 35, UserPlane.bank,
419         UserPlane.rollrate);
420         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
421         static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
422         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
423
424     }
425     else if (UserPlane.simtime - initial_simtime < 125.0)
426     {
427         comtestheading = testheading;
```

```
418
419     double aileron = Headinghold(comtestheading, UserPlane.heading, 35, UserPlane.bank,
420     UserPlane.rollrate);
421     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
422     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
423     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
424
425     }
426     else
427     {
428         exit(0);
429     }
430
431     }
432     //test autothrottle
433     else if (flag_test == 105)
434     {
435
436         double elevator = Altitudehold(6000, UserPlane.altitude, 15, UserPlane.pitch,
437         UserPlane.pitchrate);
438         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
439         static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
440         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
441         double aileron = Bankhold(0, UserPlane.bank, UserPlane.rollrate);
442         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
443         static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
444         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
445
446         if (flag_initial == 0 && UserPlane.simtime > 0)
447         {
448             initial_simtime = UserPlane.simtime;
```

```
4443     flag_initial = 1;
4444
4445 }
4446 if (UserPlane.simtime - initial_simtime < 30.0)
4447 {
4448     comvelocity = 800;
4449
4450     double throttle = Autothrottle(comvelocity, UserPlane.velocityZ, UserPlane.accelerationZ);
4451     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
4452         static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
4453         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
4454
4455 }
4456 else if (UserPlane.simtime - initial_simtime < 70.0)
4457 {
4458     comvelocity = 900;
4459
4460     double throttle = Autothrottle(comvelocity, UserPlane.velocityZ, UserPlane.accelerationZ);
4461     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
4462         static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
4463         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
4464
4465 }
4466 else if (UserPlane.simtime - initial_simtime < 120.0)
4467 {
4468     comvelocity = 800;
4469
4470     double throttle = Autothrottle(comvelocity, UserPlane.velocityZ, UserPlane.accelerationZ);
4471     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
4472         static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
4473         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
4474
4475 }
```

```
470     else
471     {
472         exit(0);
473     }
474 }
475 //test NORD
476 else if (flag_test == 106)
477 {
478
479     if (UserPlane.longitude != 0)
480     {
481         if (flag_initial == 0 && UserPlane.simtime > 0)
482         {
483             initial_simtime = UserPlane.simtime;
484             flag_initial = 1;
485             TestPlane = UserPlane;
486         }
487
488         if (UserPlane.simtime - previoustime<1)
489         {
490             deltatime = UserPlane.simtime - previoustime;
491             testVel = testVel + (1 / (111000.0 * cos(TestPlane.latitude / 180 * PI))) * 800
492                 * .3048 * deltatime);
493         }
494
495         double elevator = Altitudehold(6000, UserPlane.altitude, 15, UserPlane.pitch,
496             UserPlane.pitchrate);
497         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
498             EVENT_ELEVATOR_SET, static_cast<DWORD>(round(elevator)),
499             SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
500         double aileron = Bankhold(0, UserPlane.bank, UserPlane.rollrate);
501         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
```

```
EVENT_AILERON_SET, static_cast<DWORD>(round(aileron))),  
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
if (UserPlane.simtime - initial_simtime < 210.0)  
{  
    comNORD = 0;  
    testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude + testVel,  
TestPlane.altitude, UserPlane.latitude, UserPlane.longitude, UserPlane.altitude,  
TestPlane.heading);  
  
    double throttle = ForwardSeparation(testNED, comNORD, UserPlane.velocityZ,  
UserPlane.accelerationZ);  
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,  
EVENT_THROTTLE_SET, static_cast<DWORD>(round(throttle)),  
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
}  
else if (UserPlane.simtime - initial_simtime < 310.0)  
{  
    comNORD = 50;  
    testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude + testVel,  
TestPlane.altitude, UserPlane.latitude, UserPlane.longitude, UserPlane.altitude,  
TestPlane.heading);  
    double throttle = ForwardSeparation(testNED, comNORD, UserPlane.velocityZ,  
UserPlane.accelerationZ);  
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,  
EVENT_THROTTLE_SET, static_cast<DWORD>(round(throttle)),  
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
}  
else if (UserPlane.simtime - initial_simtime < 410.0)  
{  
    comNORD = 0;  
    testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude + testVel,
```

```
TestPlane.altitude, UserPlane.latitude, UserPlane.longitude, UserPlane.altitude,
TestPlane.heading);
    double throttle = ForwardSeparation(testNED, comNORD, UserPlane.velocityZ,
UserPlane.accelerationZ);
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
EVENT_THROTTLE_SET, static_cast<DWORD>(round(throttle)),
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

    }
    else
    {
        exit(0);
    }
    printf("com%.5f\ttest %.5f\tuser %.5f\tNORD%.5f\n", comNORD, TestPlane.longitude +
testVel, UserPlane.longitude, testNED[0]);
    previousTime = UserPlane.simtime;
    }

}

//test EAST
else if (flag_test == 107)
{
    if (flag_initial == 0 && UserPlane.simtime > 0)
    {
        initial_simtime = UserPlane.simtime;
        flag_initial = 1;
        TestPlane = UserPlane;
    }
    double throttle = Autothrottle(800, UserPlane.velocityZ, UserPlane.accelerationZ);
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
```

```
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
546 double elevator = Altitudehold(6000, UserPlane.altitude, 15, UserPlane.pitch,
    UserPlane.pitchrate);
547 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
    static_cast<DWORD>(round(elevator))), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

548
549 if (UserPlane.simtime - initial_simtime < 5.0)
550 {
551     comeEAST = 0;
552     testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude, TestPlane.altitude,
553         UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, TestPlane.heading);
554     double aileron = LateralSeparation(testNED, comeEAST, 35, UserPlane.bank,
        UserPlane.rollrate);
555     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
        static_cast<DWORD>(round(aileron))), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

556
557 }
558 else if (UserPlane.simtime - initial_simtime < 35.0)
559 {
560     comeEAST = 100;
561     testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude, TestPlane.altitude,
        UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, TestPlane.heading);
562     double aileron = LateralSeparation(testNED, comeEAST, 35, UserPlane.bank,
        UserPlane.rollrate);
563     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
        static_cast<DWORD>(round(aileron))), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

564
565 }
566 else if (UserPlane.simtime - initial_simtime < 65.0)
```

```
567 {
568     comEAST = 0;
569     testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude, TestPlane.altitude,
570                             UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, TestPlane.heading);
571     double aileron = LateralSeparation(testNED, comEAST, 35, UserPlane.bank,
572                                         UserPlane.rollrate);
573     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
574                                     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
575                                     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
576
577 }
578 else if (UserPlane.simtime - initial_simtime < 95.0)
579 {
580     comEAST = 50;
581     testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude, TestPlane.altitude,
582                             UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, TestPlane.heading);
583     double aileron = LateralSeparation(testNED, comEAST, 35, UserPlane.bank,
584                                         UserPlane.rollrate);
585     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
586                                     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
587                                     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
588
589 }
590 else if (UserPlane.simtime - initial_simtime < 125.0)
591 {
592     comEAST = 0;
593     testNED = EcefToNEDPhi(TestPlane.latitude + comEAST, TestPlane.longitude,
594                             TestPlane.altitude, UserPlane.latitude, UserPlane.longitude, UserPlane.altitude,
595                             TestPlane.heading);
596     double aileron = LateralSeparation(testNED, comEAST, 35, UserPlane.bank,
597                                         UserPlane.rollrate);
598     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
599                                     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
```

```
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

```
588     }
589
590     else
591     {
592         exit(0);
593     }
594
595
596
597
```

```
    }
    //test DOWN
    else if (flag_test == 108)
    {
```

```
600
601     double throttle = Autothrottle(800, UserPlane.velocityZ, UserPlane.accelerationZ);
602     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
        static_cast<DWORD>(round(throttle)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
        7
```

```
603
604     double aileron = Bankhold(0, UserPlane.bank, UserPlane.rollrate);
605     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
        static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
        7
```

```
606
607
608     if (flag_initial == 0 && UserPlane.simtime > 0)
609     {
610         initial_simtime = UserPlane.simtime;
611         flag_initial = 1;
612         TestPlane = UserPlane;
613     }
614     if (UserPlane.simtime - initial_simtime < 35.0)
615     {
```

```
616 comDOWN = 0;
617 testNED= EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude,
618 TestPlane.altitude,UserPlane.latitude,UserPlane.longitude,UserPlane.altitude,TestPlane.heading);
619 double elevator = VerticalSeparation(testNED, comDOWN, 20,
UserPlane.pitch,UserPlane.pitchrate);
620 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
621
622 }
623 else if (UserPlane.simtime - initial_simtime < 70.0)
624 {
625 comDOWN = 150;
626 testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude, TestPlane.altitude,
UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, TestPlane.heading);
627 double elevator = VerticalSeparation(testNED, comDOWN, 20, UserPlane.pitch,
UserPlane.pitchrate);
628 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
629
630 }
631 else if (UserPlane.simtime - initial_simtime < 105.0)
632 {
633 comDOWN = 0;
634 testNED = EcefToNEDPhi(TestPlane.latitude, TestPlane.longitude, TestPlane.altitude,
UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, TestPlane.heading);
635 double elevator = VerticalSeparation(testNED, comDOWN, 20, UserPlane.pitch,
UserPlane.pitchrate);
636 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
```

```
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
  
    }  
    else  
    {  
        exit(0);  
    }  
  
    }  
    //calcolo ned se trovo drone per scriverlo su file  
    if (flag_dronefound == 1)  
    {  
        testNED = EcefToNEDPhi(UserPlane.latitude, UserPlane.longitude, UserPlane.altitude,  
                                OtherPlane.latitude, OtherPlane.longitude, OtherPlane.altitude, UserPlane.heading);  
    }  
  
    //comandi manuali  
    if (flag_decollo == 1001)  
    {  
    }  
  
    //scrivo su file di output  
    if (flag_stampa == 1)  
    {  
        StampaFile();  
    }  
}  
  
//definisco la frequenza dell'add-on  
Sleep(50);
```

```
668     }
669
670     //chiusura applicazione e file output
671     hr = SimConnect_Close(hSimConnect);
672     if (flag_stampa == 1)
673     {
674         ChiudiFile();
675     }
676     printf("\nDisconnected from Prepar3D ");
677     system("pause");
678 }
679 else
680 {
681     printf("\nFailed to Connect to Prepar3D ");
682     system("pause");
683 }
684 system("pause");
685 return 0;
686 }
687
```

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:Addon for Simconnect linked to Prepar3D
5 //      It has to be attached to a DRONE
6 //-----
7
8 #include "headers/globalvar.h"
9 #include "headers/readwrite.h"
10 #include "headers/dispatchfun.h"
11 #include "headers/autopilots.h"
12 #include "headers/miscellaneous.h"
13
14 //title of the addon
15 const char* TITLE_STRING = "DRONE PLANE";
16
17 //main function of the addon
18 int __cdecl _tmain(int argc, _TCHAR* argv[])
19 {
20     //lettura file configs.txt
21     LetturaConfigs();
22
23     // Apertura del simconnect
24     if (SUCCEEDED(SimConnect_Open(&hSimConnect, TITLE_STRING, NULL, 0, 0, 0)))
25     {
26         //connesso al simulatore
27         printf("\nConnected to Prepar3D");
28
29
30         //ciclo principale dell'applicazione
31         while (0 == flag_quit)
32         {
33             //funzione che gestisce gli eventi del simulatore
```

```
34 SimConnect_CallDispatch(hSimConnect, MyDispatchProc, NULL);
35
36
37 //controllo se il simulatore è in pausa o meno
38 if (flag_isrunning == 1)
39 {
40     //request data on user
41     hr = SimConnect_RequestDataOnSimObjectType(hSimConnect, REQUEST_1, DEFINITION_1, 0,
42         SIMCONNECT_SIMOBJECT_TYPE_USER);
43
44     //request data on other plane
45     hr = SimConnect_RequestDataOnSimObjectType(hSimConnect, REQUEST_0, DEFINITION_1, 100000,
46         SIMCONNECT_SIMOBJECT_TYPE_AIRCRAFT);
47
48     //richiesta di comando
49     RecieveCommand();
50
51     //ceck if is on ground and correct the position
52     IsOnGround();
53
54     //controllo del carrello e retrazione
55     GearCheck();
56
57     // autopilota C2, manovra di decollo e immissione in crociera
58     if (flag_initialgroundcheck == 1)
59     {
60         switch (flag_decollo)
61         {
62             //inizializzazione
63             case 1:
64             {
65                 // manetta al 90%
66                 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET, 7
```

```

14743, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

// rimuovo il freno di stazionamento
SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
EVENT_PARKING_BRAKES_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_RUDDER_SET, 0,
SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
0, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
0, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
//passo alla prossima fase
flag_decollo = 2;
}
break;

//fase di rullaggio
case 2:
{
    // manetta al 90%
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
14743, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

    //attivo l'autopilota di heading con controllo di timone
    double rudder = Headingtakeoff(initial_heading, UserPlane.heading);
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_RUDDER_SET,
static_cast<DWORD>(round(rudder)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
    //autopilota di controllo del pitch per tenere il velivolo a terra
    double elevator = Pitchhold(0.8, UserPlane.pitch, UserPlane.pitchrate);
    SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,

```

```
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

88 //comando per richiamata
89 if (UserPlane.velocity > 300)
90 {
91     flag_decollo = 3;
92     // setto l'alettone a 0 (non servirà se implemento l'ari)
93     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER,
94     EVENT_RUDDER_SET, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST,
95     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
96 }
97 break;
98
99 // fase di richiamata e salita rettilinea
100 case 3:
101 {
102     // manetta al 90%
103     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
104     14743, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
105
106     //autopilota di mantenimento dell'heading con controllo di alettone
107     double aileron = Headinghold(initial_heading, UserPlane.heading, 10, UserPlane.bank,
108     UserPlane.rollrate);
109     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
110     static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
111     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

112     //autopilota di controllo della quota con controllo dell'equilibratore
113     double elevator = Pitchhold(-15, UserPlane.pitch, UserPlane.pitchrate);
114     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
115     static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
116     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

```
112
113 //comando per virata di allineamento alla crociera
114 if (UserPlane.altitude > quota_crociera / 3)
115 {
116     flag_decollo = 4;
117 }
118 }
119 break;
120 }
121 }
122
123
124 //fase di immissione in rotta di crociera e mantenimento della quota in crociera
125 if (flag_decollo == 4)
126 {
127     // manetta al 70%
128     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
129                                     static_cast<DWORD>(round(16383 * .7)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
130                                     SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
131
132     //autopilota di controllo della quota con controllo dell'equilibratore
133     if (UserPlane.altitude < quota_crociera - 250)
134     {
135         double elevator = Pitchhold(-15, UserPlane.pitch, UserPlane.pitchrate);
136         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
137                                         static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
138                                         SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
139     }
140     else if (UserPlane.altitude > quota_crociera + 250)
141     {
142         double elevator = Pitchhold(15, UserPlane.pitch, UserPlane.pitchrate);
```

E:\TESI\00 - Prepar3dSDK\PERSONAL\DRONE\main.cpp

```
141 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2
    static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

142 }
143
144 else
145 {
146
147     double elevator = Altitudehold(quota_crociera, UserPlane.altitude, 15, UserPlane.pitch, 2
        UserPlane.pitchrate);
148     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET, 2
        static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
149 }
150
151 //autopilota di mantenimento dell'heading con controllo di alettone
152 double aileron = Headinghold(heading_crociera, UserPlane.heading, 20, UserPlane.bank, 2
    UserPlane.rollrate);
153 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET, 2
    static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST, 2
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

154 }
155
156
157
158 //fase di crociera in quota a velocità ridotta
159 if (flag_decollo == 100)
160 {
161
162     std::vector<double> NED = EcefToNEDPhi(OtherPlane.latitude, OtherPlane.longitude, 2
        OtherPlane.altitude, UserPlane.latitude, UserPlane.longitude, UserPlane.altitude, 2
        OtherPlane.heading);
163     printf("NED NORD %.2f", NED[0]);
```

```
164 printf("EAST %.2f", NED[1]);
165 printf("DOWN %.2f", NED[2]);
166
167 // autopilota di manetta
168 double throttle = ForwardSeparation( NED, -40.0, UserPlane.velocityZ, UserPlane.accelerationZ);
169 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_THROTTLE_SET,
    static_cast<DWORD>(throttle), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
170
171
172 double elevator = VerticalSeparation(NED, 20, 20, UserPlane.pitch, UserPlane.pitchrate);
173 SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_ELEVATOR_SET,
    static_cast<DWORD>(round(elevator)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
174
175 double distance = sqrt(NED[1] * NED[1] + NED[0] * NED[0] + NED[2] * NED[2]);
176 printf("distance %.2f\n", distance);
177 if (distance > abs(1000))
178 {
179     //autopilota di mantenimento dell'heading con controllo di alettone
180     double aileron = LateralSeparationApproach(NED, 20, UserPlane.bank, UserPlane.heading,
        UserPlane.rollrate);
181     //double aileron = LateralSeparation(NED, 20, 35, UserPlane.bank, UserPlane.heading,
        UserPlane.rollrate);
182     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
        static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,
        SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
183
184 }
185 else
186 {
187     double aileron = LateralSeparation( NED, 20, 35, UserPlane.bank, UserPlane.rollrate);
188     SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_AILERON_SET,
```

```
static_cast<DWORD>(round(aileron)), SIMCONNECT_GROUP_PRIORITY_HIGHEST,  
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

```
189         }  
190     }  
191 }  
192 }  
193 }  
194 }  
195 }  
196 //definisco la frequenza dell'add-on  
197 Sleep(50);  
198 }  
199 }  
200 //chiusura applicazione e file output  
201 hr = SimConnect_Close(hSimConnect);  
202 }  
203 printf("\nDisconnected from Prepar3D ");  
204 system("pause");  
205 }  
206 else  
207 {  
208     printf("\nFailed to Connect to Prepar3D ");  
209     system("pause");  
210 }  
211 system("pause");  
212 return 0;  
213 }  
214 }
```

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:header file with the simulation main body
5 //-----
6 #pragma once
7 #include "globalvar.h"
8 #include "communicationmodule.h"
9
10 //funzione di callback per gestire gli eventi del simulatore
11 void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData, void* pContext)
12 {
13     switch (pData->dwID)
14     {
15         //code called on the connection with the simulation
16         case SIMCONNECT_RECV_ID_OPEN:
17         {
18             printf("Inizializzazione\n");
19             // Set up the data definition 1
20             hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Title", NULL,
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
32 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Elevator Deflection", "degrees");
33 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rudder Deflection", "degrees");
34 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Aileron Left Deflection", "degrees");
35 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "General Eng Throttle Lever Position:1", "percent");
36 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "TRANSPONDER CODE:1", "BC016");
37 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Incidence alpha", "degrees");
38 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Incidence beta", "degrees");
39 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rotation velocity body X", "radians per second");
40 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rotation velocity body Z", "radians per second");
41 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Rotation velocity body Y", "radians per second");
42 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Acceleration Body Z", "feet per second squared");
43 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Acceleration Body X", "feet per second squared");
44 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Acceleration Body Y", "feet per second squared");
45 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "RELATIVE WIND VELOCITY BODY X", "feet per second squared");
46 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "RELATIVE WIND VELOCITY BODY Y", "feet per second squared");
47 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "RELATIVE WIND VELOCITY BODY Z", "feet per second squared");
48
49
50 // Set up the data definition 2
51 hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_2, "Plane Heading Degrees True", "degrees");
52
53 //link the event in the simulator to the event in the addon
```

```
54 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_THROTTLE_SET, "THROTTLE_SET");
55 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_ELEVATOR_SET, "ELEVATOR_SET");
56 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_AILERON_SET, "AILERON_SET");
57 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_RUDDER_SET, "RUDDER_SET");
58 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_FLAPS_SET, "FLAPS_SET");
59 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_PARKING_BRAKES_SET, "PARKING_BRAKES");
60 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_LANDING_GEAR, "GEAR_UP");
61 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_TRANSPONDER, "XPNDR_SET");
62
63
64
65 //INPUTS
66 // simulator keys
67 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_2, "HOTAS_KEY_A0");
68 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_2);
69
70 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_3, "HOTAS_KEY_A1");
71 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_3);
72
73 //keyboard keys
74 // Map event, add to notification group map the keys to the private events and activate them
75 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_0);
76 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_0);
77 hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_0, "VK_COMMA", EVENT_0);
78 hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_0, SIMCONNECT_STATE_ON);
79
80 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1);
81 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_1);
82 hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "q", EVENT_1);
83 hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_1, SIMCONNECT_STATE_ON);
84
85 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_5);
86 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_5);
87 hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_3, "w", EVENT_5);
```

```
87 hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_3, SIMCONNECT_STATE_ON);
88
89 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_6);
90 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_6);
91 hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_4, "a", EVENT_6);
92 hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_4, SIMCONNECT_STATE_ON);
93
94 hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_7);
95 hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_7);
96 hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_5, "s", EVENT_7);
97 hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_5, SIMCONNECT_STATE_ON);
98
99
100 // set the notification group priority
101 hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);
102
103 //take some inicial values
104 initial_transponder = UserPlane.transponderCode;
105 initial_simtime = UserPlane.simtime;
106 initial_latitude = UserPlane.latitude;
107 initial_longitude = UserPlane.longitude;
108
109
110 // Subscribe to Pause and Unpause events
111 SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_PAUSED, "Pause");
112
113
114 printf("Inizializazione completata\n");
115
116 }
117 break;
118
119 //code to handle events received in a SIMCONNECT_RECV_EVENT structure.
```

```
120 case SIMCONNECT_RECV_ID_EVENT:
121 {
122     SIMCONNECT_RECV_EVENT* evt = (SIMCONNECT_RECV_EVENT*)pData;
123     switch (evt->uEventID)
124     {
125     case EVENT_SIM_PAUSED:
126     {
127         if (evt->dwData == 1) {
128             printf("Simulatore in Pausa\n");
129             flag_isrunning = 0;
130         }
131         else {
132             printf("Simulatore in Esecuzione\n");
133             flag_isrunning = 1;
134         }
135     }
136     break;
137
138 case EVENT_SIM_UNPAUSED:
139 {
140     }
141     break;
142
143 case EVENT_0:
144 {
145     //event 0 è assegnato alla virgola
146     if(flag_decollo==0)
147     {
148         flag_decollo = 1;
149         printf("Autopilota di immissione in crociera attivo\n");
150     }
151 }
152 }
```

```
153     }
154     else
155     {
156         printf("Comandi Manuali\n");
157         flag_decollo = 1001;
158     }
159 }
160 break;
161
162
163 case EVENT_1:
164 {
165     //event 1 è assegnato alla q
166     printf("q premuto\n");
167
168     if (flag_pitch == 0)
169     {
170
171         flag_pitch = 1;
172         pitchang = 1000;
173     }
174     else
175     {
176
177         flag_pitch = 0;
178         pitchang = 0;
179     }
180 }
181
182 }
183 break;
184
185 case EVENT_5:
```

```
186 {
187     //event 1 è assegnato alla q
188     printf("w premuto\n");
189
190
191
192 }
193 break;
194
195 case EVENT_6:
196 {
197     //event 6 assegnato alla a
198     printf("a premuto\n");
199
200     if (flag_roll == 0)
201     {
202
203         flag_roll = 1;
204         bankang = -20.0;
205     }
206     else
207     {
208
209         flag_roll = 0;
210         bankang = 0;
211     }
212 }
213
214 }
215 break;
216
217 case EVENT_7:
218 {
```

```
219 //event 1 è assegnato alla q
220 printf("s premuto\n");
221
222
223
224 }
225 break;
226
227
228 case EVENT_2:
229 {
230     flag_decollo = 1;
231     printf("Autopilota di immissione in crociera attivo\n");
232     // The message text to be displayed
233     const char* message = "AUTOPILOTA DI DECOLLO E IMMISSIONE IN CROCIERA ATTIVO";
234     hr=SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW,10.0,EVENT_PRINT_1, (DWORD)0
235
236
237
238 }
239 break;
240
241 case EVENT_3:
242 {
243     flag_decollo = 1001;
244     printf("Passaggio a comandi manuali\n");
245     // The message text to be displayed
246     const char* message2 = "COMANDI MANUALI";
247     hr = SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW, 10.0, EVENT_PRINT_2,
248         (DWORD)(strlen(message2) + 1), (void*)message2);
249     hr = SimConnect_ClearNotificationGroup(hSimConnect, GROUP_2);
250 }
```

```
250     break;
251
252
253
254     default:
255     break;
256
257     }
258     break;
259
260     // if a call to SimConnect_RequestDataOnSimObjectType succeed this event opens.
261     case SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE:
262     {
263         SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE* pObjjData = (SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*)pData;
264         switch (pObjjData->dwRequestID)
265         {
266             //richiesta di info su altri velivoli
267             case REQUEST_0:
268             {
269                 DWORD Object_flag = 0;
270                 for (int i = 1; i <= 2; i++)
271                 {
272                     pObjjData->dwentrynumber = i;
273                     DWORD ObjectID = pObjjData->dwObjectID;
274
275                     if (pObjjData->dwObjectID != UserID && pObjjData->dwObjectID != Object_flag)
276                     {
277                         flag_dronefound = 1;
278                         Object_flag = ObjectID;
279                         ObjectID2 = ObjectID;
280                         V2 = (ObjectDataStruct*)&pObjjData->dwData;
281                         if (SUCCEEDED(StringCbLengthA(&V2->title[0], sizeof(V2->title), NULL))) // security
282                             check
```

```

282 {
283     flag_dronefound = 1;
284     OtherPlane = *V2;
285
286     OtherPlane.velocity = sqrt(OtherPlane.velocityX * OtherPlane.velocityX +
287                               OtherPlane.velocityY * OtherPlane.velocityY + OtherPlane.velocityZ *
288                               OtherPlane.velocityZ);
289     //print a schermo delle caratteristiche
290     printf("DR Alt=%f Pitch=%f Bank=%f Heading=%f V=%.1f VX=%.1f VY=%.1f VZ=%.1f Elev=%
291           %.2f Rudd=%.2f Ail=%.2f Thrott=%.1f Transp=%.1f alpha=%.5f beta=%.5f\n", V2->altitude,
292           V2->pitch, V2->bank, V2->heading, OtherPlane.velocity, V2->velocityX, V2->velocityY,
293           V2->velocityZ, V2->elevator, V2->rudder, V2->aileron, V2->throttle, V2-
294           >transponderCode, V2->alpha, V2->beta);
295 }
296
297 }
298
299 }
300 break;
301
302 //richiesta di info sul velivolo principale
303 case REQUEST_1:
304 {
305     UserID = pObjData->dwojectID;
306     V1 = (ObjectDataStruct*)&pObjData->dwoData;
307     if (SUCCEEDED(StringCbLengthA(&V1->title[0], sizeof(V1->title), NULL))) // security check
308     {
309         //passo le variabili ad un vettore utilizzabile
310         UserPlane = *V1;
311         UserPlane.velocity = sqrt(UserPlane.velocityX * UserPlane.velocityX + UserPlane.velocityY *
312                                   UserPlane.velocityY + UserPlane.velocityZ * UserPlane.velocityZ);
313
314         //print a schermo delle caratteristiche
315         printf("C2 Alt=%f Pitch=%f Bank=%f Heading=%f V=%.1f VX=%.1f VY=%.1f VZ=%.1f Elev=%.2f

```

E:\TESI\00 - Prepar3dSDK\PERSONAL\C2 PLANE\headers\dispatchfun.h

11

```
Rudd=%.2f Ail=%.2f Thrott=%.1f Transp=%.5f beta=%.5f\n", V1->altitude, V1->  
>pitch, V1->bank, V1->heading, UserPlane.velocity, V1->velocityX, V1->velocityY, V1->  
>velocityZ, V1->elevator, V1->rudder, V1->aileron, V1->throttle, V1->transponderCode, V1->  
>alpha, V1->beta);
```

```
308     }  
309     }  
310     break;  
311     default:  
312     break;  
313     }  
314     }  
315     break;  
316     }  
317  
318  
319     //code to handle errors received in a SIMCONNECT_RECV_EXCEPTION structure.  
320     case SIMCONNECT_RECV_ID_EXCEPTION:  
321     {  
322         SIMCONNECT_RECV_EXCEPTION* except = (SIMCONNECT_RECV_EXCEPTION*)pData;  
323  
324         switch (except->dwException)  
325         {  
326             case SIMCONNECT_EXCEPTION_ERROR:  
327             {  
328                 printf("\nError");  
329             }  
330             break;  
331             default: {}  
332             break;  
333         }  
334     }  
335     break;  
336 }
```

```
337 //code to handle exiting the application
338 case SIMCONNECT_RECV_ID_QUIT:
339 {
340     flag_quit = 1;
341 }
342 break;
343
344 // code to handle the case where an unexpected message is received
345 default:
346 {
347     printf("\nReceived:%d", pData->dwID);
348 }
349 break;
350 }
351 }
352
```

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:header file with all autopilots functions
5 //-----
6 #pragma once
7 #include "globalvar.h"
8
9 double Headingtakeoff(double comand_heading, double actual_heading);
10
11 double Pitchhold(double comand_pitch, double actual_pitch, double atual_pitch_vel);
12
13 double Bankhold(double max_bank, double actual_bank, double roll_rate);
14
15 double Headinghold(double comand_heading, double actual_heading, double max_bank, double actual_bank, double
    roll_rate);
16
17 double Altitudehold(double comand_altitude, double actual_altitude, double max_pitch, double actual_pitch, double
    actual_pitch_vel);
18
19 double Autothrottle(double comand_velocity, double C2_velocity, double acceleration);
20
21 double VerticalSeparation(std::vector<double> NED, double verticalseparation, double max_pitch, double
    actual_pitch, double pitch_rate);
22
23 double LateralSeparationApproach(std::vector<double> NED, double max_bank, double actual_bank, double
    actual_heading, double roll_rate);
24
25 double LateralSeparation(std::vector<double> NED, double comand_separation, double max_bank, double actual_bank,
    double roll_rate);
26
27 double ForwardSeparation(std::vector<double> NED, double comand_separation, double actual_velocity, double
    acceleration);
```

```
28
29
30 //autopilota dell'angolo di heading con controllo del solo timone
31 double Headingtakeoff(double comand_heading, double actual_heading)
32 {
33
34
35     static double err = 0;
36     static double err_integral = 0;
37     static double err_derivative = 0;
38     static double err_previous = 0;
39     if (actual_heading >= comand_heading)
40     {
41
42         if (actual_heading - comand_heading >= 180)
43         {
44             err = -(actual_heading - comand_heading);
45         }
46         else
47         {
48             err = (actual_heading - comand_heading);
49         }
50
51     }
52     else
53     {
54         if (comand_heading - actual_heading >= 180)
55         {
56             err = -(actual_heading - comand_heading);
57         }
58         else
59         {
60             err = (actual_heading - comand_heading);
```

```
61     }
62 }
63 err_integral += err;
64 err_derivative = err - err_previous;
65 err_previous = err;
66 double max_err_integral = 10000/50.0;
67 if (err_integral > max_err_integral && err_integral > 0)
68 {
69     err_integral = max_err_integral;
70 }
71 else if (err_integral < -max_err_integral && err_integral < 0)
72 {
73     err_integral = -max_err_integral;
74 }
75 double rudder = 1500 * err + 50* err_integral + 0 * err_derivative;
76 if (rudder > 16383)
77 {
78     rudder = 16382;
79 }
80 else if (rudder < -16383)
81 {
82     rudder = -16382;
83 }
84
85
86 return (rudder);
87 }
88
89 double Pitchhold(double comand_pitch, double actual_pitch, double pitch_rate)
90 {
91     static double time = 0;
92     static double previoustime = 0;
93     static double elevator = 0;
```

```
94 static double previouslevator = 0;
95
96 time = UserPlane.simtime;
97
98 static double err = 0;
99 static double err_previous = 0;
100 static double err2 = 0;
101 static double err_previous2 = 0;
102
103 //pitch negativo a cabrare
104 err = comand_pitch - actual_pitch;
105
106 //lead controller
107 double comand_pitch_rate = 9000 * (err + (0.3 * (err - err_previous) / (time - previousime)));
108
109 err2 = comand_pitch_rate - 4500*pitch_rate;
110
111 //lag
112 double alpha = (time - previousime) / ((time - previousime) + 0.8);
113 elevator = alpha * err2 + ((1 - alpha) * previouslevator);
114
115 if (elevator > 16383)
116 {
117     elevator = 16383;
118 }
119 else if (elevator < -16383)
120 {
121     elevator = -16383;
122 }
123
124 err_previous = err;
125 err_previous2 = err2;
126 previousime = time;
```

```
127 if (!std::isnan(elevator))
128 {
129     previous_elevator = elevator;
130 }
131
132 return (elevator);
133 }
134
135 double Bankhold(double commanded_bank, double actual_bank, double roll_rate)
136 {
137     static double time = 0;
138     static double previoustime = 0;
139     static double aileron = 0;
140     static double previousaileron = 0;
141
142     time = UserPlane.simtime;
143
144     static double err = 0;
145     static double err_previous = 0;
146     static double err2 = 0;
147     static double err_previous2 = 0;
148
149
150     err = commanded_bank - actual_bank;
151
152     //lead controller
153     double command_roll_rate = 400 * (err + (0.64* (err - err_previous) / (time - previoustime)));
154
155     err2 = command_roll_rate - 200 * roll_rate;
156
157     //lag
158     double alpha = (time - previoustime) / ((time - previoustime) + 0.7);
159     aileron = alpha * err2 + ((1 - alpha) * previousaileron);
```

```
160
161     if (aileron > 16383)
162     {
163         aileron = 16383;
164     }
165     else if (aileron < -16383)
166     {
167         aileron = -16383;
168     }
169
170     err_previous = err;
171     err_previous2 = err2;
172     previous_time = time;
173     if (!std::isnan(aileron))
174     {
175         previous_aileron = aileron;
176     }
177
178     return (aileron);
179 }
180
181 //autopilota di controllo della quota
182 double Altitudehold(double comand_altitude, double actual_altitude, double max_pitch, double actual_pitch, double pitch_rate)
183 {
184     static double time = 0;
185     static double previous_time = 0;
186     time = UserPlane.simtime;
187
188     static double altitude_err = 0;
189     static double altitude_err_integral = 0;
190     static double altitude_err_derivative = 0;
191     static double altitude_err_previous = 0;
```

```
192 static double comand_pitch = 0;
193 static double saturated_pitch = 0;
194 static double windup = 0;
195
196 altitude_err = actual_altitude - comand_altitude;
197
198 //protezione dall'inizializzazione
199 if ((time - previoustime) > 1)
200 {
201     altitude_err_derivative = 0;
202     altitude_err_integral = 0;
203 }
204 else
205 {
206
207     if (!std::isnan(comand_pitch - saturated_pitch))
208     {
209         windup = (comand_pitch - saturated_pitch);
210
211     }
212     else
213     {
214         windup = 0.0;
215     }
216
217     altitude_err_derivative = (altitude_err - altitude_err_previous) / (time - previoustime);
218     altitude_err_integral += ((altitude_err + altitude_err_previous) * (time - previoustime) / 2) - 1 * windup;
219 }
220
221 altitude_err_previous = altitude_err;
222 comand_pitch = 0.08 * altitude_err + 0.01 * altitude_err_integral + 0.09 * altitude_err_derivative;
223
224
```

```
225
226 if (comand_pitch > max_pitch && comand_pitch > 0)
227 {
228     saturated_pitch = max_pitch;
229 }
230 else if (comand_pitch < -max_pitch && comand_pitch < 0)
231 {
232     saturated_pitch = -max_pitch;
233 }
234 else
235 {
236     saturated_pitch = comand_pitch;
237 }
238
239 static double elevator = 0;
240 static double previouslevator = 0;
241
242 static double err = 0;
243 static double err_previous = 0;
244 static double err2 = 0;
245 static double err_previous2 = 0;
246
247 //pitch negativo a cabrare
248 err = saturated_pitch - actual_pitch;
249
250 //lead controller
251 double comand_pitch_rate = 9000 * (err + (0.3 * (err - err_previous) / (time - previousime)));
252
253 err2 = comand_pitch_rate - 4500 * pitch_rate;
254
255 //lag
256 double alpha = (time - previousime) / ((time - previousime) + 0.8);
257 elevator = alpha * err2 + ((1 - alpha) * previouslevator);
```

```
258
259     if (elevator > 16383)
260     {
261         elevator = 16383;
262     }
263     else if (elevator < -16383)
264     {
265         elevator = -16383;
266     }
267
268     err_previous = err;
269     err_previous2 = err2;
270     previous_time = time;
271     if (!std::isnan(elevator))
272     {
273         previous_elevator = elevator;
274     }
275
276     return (elevator);
277 }
278
279 // autopilota dell'angolo di heading con controllo del solo alettone
280 double Headinghold(double command_heading, double actual_heading, double max_bank, double actual_bank, double
    roll_rate)
281 {
282     static double time = 0;
283     static double previous_time = 0;
284     time = UserPlane.simtime;
285
286     static double heading_err = 0;
287     static double heading_err_derivative = 0;
288     static double heading_err_previous = 0;
289     static double heading_err_integral = 0;
```

```
290 static double comanded_bank = 0;
291 static double saturated_bank = 0;
292 static double windup = 0;
293
294 if (actual_heading >= comand_heading)
295 {
296
297     if (actual_heading - comand_heading >= 180)
298     {
299         heading_err = (comand_heading-actual_heading);
300     }
301     else
302     {
303         heading_err = -(comand_heading - actual_heading);
304     }
305
306 }
307 else
308 {
309     if (comand_heading - actual_heading >= 180)
310     {
311         heading_err = (comand_heading-actual_heading);
312     }
313     else
314     {
315         heading_err = -(comand_heading- actual_heading);
316     }
317 }
318
319 if ((time - previoustime) > 1)
320 {
321     heading_err_derivative = 0;
322     heading_err_integral = 0;
```

```
323     }
324     else
325     {
326
327         if (!std::isnan(comanded_bank - saturated_bank))
328         {
329             windup = (comanded_bank - saturated_bank);
330         }
331         else
332         {
333             windup = 0.0;
334         }
335     }
336     heading_err_derivative = (heading_err - heading_err_previous) / (time - previoustime);
337     heading_err_integral += ((heading_err + heading_err_previous) * (time - previoustime) / 2) - 0.001 * P
338     windup;
339 }
340 heading_err_previous = heading_err;
341 comanded_bank = 10*heading_err + 0.00 * heading_err_integral + 0 * heading_err_derivative;
342
343 if (comanded_bank > max_bank && comanded_bank > 0)
344 {
345     saturated_bank = max_bank;
346 }
347 else if (comanded_bank < -max_bank && comanded_bank < 0)
348 {
349     saturated_bank = -max_bank;
350 }
351 else
352 {
353     saturated_bank = comanded_bank;
354 }
```

```
355
356 static double aileron = 0;
357 static double previousaileron = 0;
358
359 static double err = 0;
360 static double err_previous = 0;
361 static double err2 = 0;
362 static double err_previous2 = 0;
363
364
365 err = saturated_bank - actual_bank;
366
367 //lead controller
368 double comand_roll_rate = 400 * (err + (0.6 * (err - err_previous) / (time - previoustime)));
369
370 err2 = comand_roll_rate - 200 * roll_rate;
371
372 //lag
373 double alpha = (time - previoustime) / ((time - previoustime) + 0.7);
374 aileron = alpha * err2 + ((1 - alpha) * previousaileron);
375
376 if (aileron > 16383)
377 {
378     aileron = 16383;
379 }
380 else if (aileron < -16383)
381 {
382     aileron = -16383;
383 }
384
385 err_previous = err;
386 err_previous2 = err2;
387 previoustime = time;
```

```
388 if (!std::isnan(aileron))
389 {
390     previousaileron = aileron;
391 }
392
393 return (aileron);
394 }
395
396
397
398
399 double Autothrottle(double comand_velocity, double actual_velocity, double acceleration)
400 {
401     static double time = 0;
402     static double previoustime = 0;
403     static double throttle = 0;
404     static double previousthrottle = 0;
405
406     time = UserPlane.simtime;
407
408     static double err = 0;
409     static double err_previous = 0;
410     static double err2 = 0;
411     static double err_previous2 = 0;
412
413
414     err = comand_velocity - actual_velocity;
415
416     //lead controller
417     double comand_throttle = 3000 * (err + (0.27 * (err - err_previous) / (time - previoustime)));
418
419     err2 = comand_throttle - 1500 * acceleration;
420
```

```
421 //lag
422 double alpha = (time - previoustime) / ((time - previoustime) + 0);
423 throttle = alpha * err2 + ((1 - alpha) * previousthrottle);
424
425 if (throttle > 16383*0.9)
426 {
427     throttle = 16383*0.9;
428 }
429 else if (throttle < 16383*0.3)
430 {
431     throttle = 16383*0.3;
432 }
433 err_previous = err;
434 err_previous2 = err2;
435 previoustime = time;
436 if (!std::isnan(throttle))
437 {
438     previousthrottle = throttle;
439 }
440
441 return (throttle);
442 }
443
444 //autopilota di controllo della quota separazione verticale positiva = drone + in basso
445 double VerticalSeparation(std::vector<double> NED, double verticalseparation, double max_pitch, double
    actual_pitch, double pitch_rate)
446 {
447     static double time = 0;
448     static double previoustime = 0;
449     time = UserPlane.simtime;
450
451     static double altitude_err = 0;
452     static double altitude_err_integral = 0;
```

```
453 static double altitude_err_derivative = 0;
454 static double altitude_err_previous = 0;
455 static double comand_pitch = 0;
456 static double saturated_pitch = 0;
457 static double windup = 0;
458
459 altitude_err = -(NED[2] - verticalseparation)* 3.28084;
460
461 //protezione dall'inizializzazione
462 if ((time - previoustime) > 1)
463 {
464     altitude_err_derivative = 0;
465     altitude_err_integral = 0;
466 }
467 else
468 {
469
470     if (!std::isnan(comand_pitch - saturated_pitch))
471     {
472         windup = (comand_pitch - saturated_pitch);
473     }
474     else
475     {
476         windup = 0;
477     }
478
479     altitude_err_derivative = (altitude_err - altitude_err_previous) / (time - previoustime);
480     altitude_err_integral += ((altitude_err + altitude_err_previous) * (time - previoustime) / 2) - 1 * windup;
481 }
482
483 altitude_err_previous = altitude_err;
484 comand_pitch = 0.08 * altitude_err + 0.01 * altitude_err_integral + 0.09 * altitude_err_derivative;
485
```

```
486 if (comand_pitch > max_pitch && comand_pitch > 0)
487 {
488     saturated_pitch = max_pitch;
489 }
490 else if (comand_pitch < -max_pitch && comand_pitch < 0)
491 {
492     saturated_pitch = -max_pitch;
493 }
494 else
495 {
496     saturated_pitch = comand_pitch;
497 }
498
499 static double elevator = 0;
500 static double previouslevator = 0;
501
502 static double err = 0;
503 static double err_previous = 0;
504 static double err2 = 0;
505 static double err_previous2 = 0;
506
507 //pitch negativo a cabrare
508 err = saturated_pitch - actual_pitch;
509
510 //lead controller
511 double comand_pitch_rate = 9000 * (err + (0.2 * (err - err_previous) / (time - previous_time)));
512
513 err2 = comand_pitch_rate - 4500 * pitch_rate;
514
515 //lag
516 double alpha = (time - previous_time) / ((time - previous_time) + 0.8);
517 elevator = alpha * err2 + ((1 - alpha) * previouslevator);
518
```

```
519     if (elevator > 16383)
520     {
521         elevator = 16383;
522     }
523     else if (elevator < -16383)
524     {
525         elevator = -16383;
526     }
527
528     err_previous = err;
529     err_previous2 = err2;
530     previous_time = time;
531     if (!std::isnan(elevator))
532     {
533         previous_elevator = elevator;
534     }
535
536     return (elevator);
537 }
538
539
540 double LateralSeparationApproach(std::vector<double> NED, double max_bank, double actual_bank, double
    actual_heading, double roll_rate)
541 {
542     double command_heading;
543     double commanded_bank;
544     double distance;
545     distance = sqrt(NED[1] * NED[1] + NED[0] * NED[0] + NED[2] * NED[2]);
546     double forward_dist = 0;
547     forward_dist = abs(NED[0]);
548
549     static double heading_err = 0;
550     static double heading_err_derivative = 0;
```

```
551 static double heading_err_previous = 0;
552 static double heading_err_integral = 0;
553
554 comand_heading = -atan((NED[1]) / forwardist) * (180.0 / PI);
555
556
557 if (comand_heading < 0)
558 {
559     comand_heading = comand_heading + 360;
560 }
561
562 if (actual_heading >= comand_heading)
563 {
564
565     if (actual_heading - comand_heading >= 180)
566     {
567         heading_err = -(actual_heading - comand_heading);
568     }
569     else
570     {
571         heading_err = (actual_heading - comand_heading);
572     }
573
574 }
575 else
576 {
577     if (comand_heading - actual_heading >= 180)
578     {
579         heading_err = -(actual_heading - comand_heading);
580     }
581     else
582     {
583         heading_err = (actual_heading - comand_heading);
```

```
584     }
585 }
586
587 static double time = 0;
588 static double previoustime = 0;
589 time = UserPlane.simtime;
590
591
592
593
594
595 heading_err_derivative = (heading_err - heading_err_previous) / (time - previoustime);
596 heading_err_integral += (heading_err + heading_err_previous) * (time - previoustime) / 2;
597 heading_err_previous = heading_err;
598
599 comanded_bank = 2 * heading_err + 0 * heading_err_integral + 0 * heading_err_derivative;
600
601
602
603 if (comanded_bank > max_bank && comanded_bank > 0)
604 {
605     comanded_bank = max_bank;
606 }
607 else if (comanded_bank < -max_bank && comanded_bank < 0)
608 {
609     comanded_bank = -max_bank;
610 }
611 static double aileron = 0;
612 static double previousaileron = 0;
613
614
615 static double err = 0;
616 static double err_previous = 0;
617 static double err2 = 0;
```

```
617 static double err_previous2 = 0;
618
619 //pitch negativo a cabrare
620 err = comanded_bank - actual_bank;
621
622
623 //lead controller
624 double comand_roll_rate = 250 * (err + (0.1 * (err - err_previous) / (time - previoustime)));
625
626
627
628 err2 = comand_roll_rate - 125 * roll_rate;
629
630
631 //lag
632 double alpha = (time - previoustime) / ((time - previoustime) + 1);
633
634 aileron = alpha * err2 + ((1 - alpha) * previousaileron);
635
636
637 if (aileron > 16383)
638 {
639     aileron = 16383;
640 }
641 else if (aileron < -16383)
642 {
643     aileron = -16383;
644 }
645
646 err_previous = err;
647 err_previous2 = err2;
648 previoustime = time;
649 if (!std::isnan(aileron))
```

```
650 {
651     previousaileron = aileron;
652 }
653
654 return (aileron);
655 }
656
657 double LateralSeparation(std::vector<double> NED, double comand_separation, double max_bank, double actual_bank,
double roll_rate)
658 {
659     static double previousned = 0;
660     static double comand_bank=0;
661     static double saturated_bank = 0;
662     static double lateral_err = 0;
663     static double lateral_err_derivative = 0;
664     static double lateral_err_previous = 0;
665     static double lateral_err_integral = 0;
666     static double windup = 0;
667     static double aileron = 0;
668     static double previousaileron = 0;
669
670
671
672     static double time = 0;
673     static double previoustime = 0;
674     time = UserPlane.simtime;
675
676     lateral_err = -(comand_separation - NED[1]);
677     if ((time - previoustime) > 1)
678     {
679         lateral_err_derivative = 0;
680         lateral_err_integral = 0;
681     }
```

```
682     else
683     {
684
685         if (!std::isnan(comanded_bank - saturated_bank))
686         {
687             windup = (comanded_bank - saturated_bank);
688         }
689         else
690         {
691             windup = 0;
692         }
693     }
694     lateral_err_derivative = (lateral_err - lateral_err_previous) / (time - previoustime);
695     lateral_err_integral += ((lateral_err + lateral_err_previous) * (time - previoustime) / 2) - 0.001 * ⤵
        windup;
696     }
697
698     lateral_err_previous = lateral_err;
699     comanded_bank = 1 * lateral_err + 0.001 * lateral_err_integral + 3.6 * lateral_err_derivative;
700
701     if (comanded_bank > max_bank && comanded_bank > 0)
702     {
703         saturated_bank = max_bank;
704     }
705     else if (comanded_bank < -max_bank && comanded_bank < 0)
706     {
707         saturated_bank = -max_bank;
708     }
709     else
710     {
711         saturated_bank = comanded_bank;
712     }
713     static double err = 0;
```

```
714 static double err_previous = 0;
715 static double err2 = 0;
716 static double err_previous2 = 0;
717
718
719 err = saturated_bank - actual_bank;
720
721 //lead controller
722 double comand_roll_rate = 400 * (err + (0.6 * (err - err_previous) / (time - previoustime)));
723
724 err2 = comand_roll_rate - 200 * roll_rate;
725
726 //lag
727 double alpha = (time - previoustime) / ((time - previoustime) + 0.7);
728 aileron = alpha * err2 + ((1 - alpha) * previousaileron);
729
730 if (aileron > 16383)
731 {
732     aileron = 16383;
733 }
734 else if (aileron < -16383)
735 {
736     aileron = -16383;
737 }
738
739 err_previous = err;
740 err_previous2 = err2;
741 previoustime = time;
742 if (!std::isnan(aileron))
743 {
744     previousaileron = aileron;
745 }
746
```

```
747     return (aileron);
748 }
749
750 double ForwardSeparation(std::vector<double> NED, double comand_separation, double actual_velocity, double
    acceleration)
751 {
752
753     static double time = 0;
754     static double previoustime = 0;
755     time = UserPlane.simtime;
756
757     static double forward_err = 0;
758     static double forward_err_integral = 0;
759     static double forward_err_derivative = 0;
760     static double forward_err_previous = 0;
761     static double windup = 0;
762     static double throttle = 0;
763     static double comand_velocity = 0;
764     static double saturated_velocity = 0;
765     static double previousthrottle = 0;
766     forward_err = comand_separation - NED[0];
767
768     //protezione dall'inizializzazione
769     if ((time - previoustime) > 1)
770     {
771         forward_err_derivative = 0;
772         forward_err_integral = 0;
773     }
774     else
775     {
776
777         if (!std::isnan(comand_velocity - saturated_velocity))
778         {
```

```
779     windup = (comand_velocity - saturated_velocity);
780 }
781 else
782 {
783     windup = 0;
784 }
785 }
786 forward_err_derivative = (forward_err - forward_err_previous) / (time - previoustime);
787 forward_err_integral += ((forward_err + forward_err_previous) * (time - previoustime) / 2) - 0.5* windup;
788 }
789 forward_err_previous = forward_err;
790
791
792 comand_velocity = 1 * forward_err + 0.03 * forward_err_integral + 2 * forward_err_derivative;
793
794
795 if (comand_velocity > 1060)
796 {
797     saturated_velocity = 1060;
798 }
799 else if (comand_velocity < 600)
800 {
801     saturated_velocity = 600;
802 }
803 else
804 {
805     saturated_velocity = comand_velocity;
806 }
807
808 printf("time %.1f\t err%.5f\t int%.5f\t comand_velocity%.5f\n", UserPlane.simtime, forward_err,
      forward_err_integral, comand_velocity);
809
810
```

```
811 static double err = 0;
812 static double err_previous = 0;
813 static double err2 = 0;
814 static double err_previous2 = 0;
815
816
817 err = saturated_velocity - actual_velocity;
818
819 //lead controller
820 double comand_throttle = 3000 * (err + (0.27 * (err - err_previous) / (time - previoustime)));
821
822 err2 = comand_throttle - 1500 * acceleration;
823
824 //lag
825 double alpha = (time - previoustime) / ((time - previoustime) + 0);
826 throttle = alpha * err2 + ((1 - alpha) * previousthrottle);
827
828 if (throttle > 16383 * 0.9)
829 {
830     throttle = 16383 * 0.9;
831 }
832 else if (throttle < 16383 * 0.3)
833 {
834     throttle = 16383 * 0.3;
835 }
836 printf("\nthrottle %f", throttle);
837 err_previous = err;
838 err_previous2 = err2;
839 previoustime = time;
840 if (!std::isnan(throttle))
841 {
842     previousthrottle = throttle;
843 }
```

E:\TESI\00 - Prepar3dSDK\PERSONAL\C2 PLANE\headers\autopilots.h

27

844

845 return (throttle);

846 }

847

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:header file with all communication functions
5 //-----
6 #pragma once
7 #include "globalvar.h"
8 void SendCommandTakeOff();
9 void SendCommandReachC2();
10 void SendCommandFormation();
11
12 void SendCommandTakeOff()
13 {
14     if (UserPlane.transponderCode != 0x0001)
15     {
16         hr = SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_TRANSPONDER,
17             static_cast<DWORD>(0x0001), SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
18         printf("\n Transponder code set to 1");
19     }
20 }
21 void SendCommandReachC2()
22 {
23     if (UserPlane.transponderCode != 0x0002)
24     {
25         hr = SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_TRANSPONDER,
26             static_cast<DWORD>(0x0002), SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
27         printf("\n Transponder code set to 2");
28     }
29 }
30 void SendCommandFormation()
31 {
```

E:\TESI\00 - Prepar3dSDK\PERSONAL\C2 PLANE\headers\communicationmodule.h

```
32  hr = SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_TRANSPONDER,  
    static_cast<DWORD>(0x0003), SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
33  printf("\n Transponder code set to 3");  
34  }  
35  
36  
37  
38  
39  
40  
41
```

2



```

1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:header file with all additional functions
5 //-----
6 #pragma once
7 #include "globalvar.h"
8
9 void IsOnGround();
10
11 void GearCheck();
12
13 std::vector<double> RelativePos(double C2_latitude, double C2_longitude, double C2_altitude, double C2_roll, double C2_heading);
14
15 //TRANSFORM the info on latitude longitude and altitude to the NED reference frame centered in C2 and rotated by
16   heading,pitch and roll
17 std::vector<double> EcefToBody(double C2_latitude, double C2_longitude, double C2_altitude, double C2_heading);
18
19 //TRANSFORM the info on latitude longitude and altitude to the NED reference frame centered in C2
20 std::vector<double> EcefToNEDPhi(double C2_latitude, double C2_longitude, double C2_altitude, double
21   C2_latitude, double C2_longitude, double C2_heading);
22
23 void IsOnGround()
24 {
25     if (UserPlane.velocity < 50 && flag_initialgroundcheck == 0 && UserPlane.simtime>0.1)
26     {
27         //correct the heading on the take off runway
28         initialHeading = initial_heading;
29         hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION_2, SIMCONNECT_OBJECT_ID_USER, 0, 0, sizeof
30           (initial), &initial);
31         flag_initialgroundcheck = 1;

```

```
29     }
30     else if (UserPlane.velocity > 50 && flag_initialgroundcheck == 0 && UserPlane.simtime > 0.1)
31     {
32         flag_initialgroundcheck = 1;
33         flag_decollo = 4;
34     }
35 }
36
37 void GearCheck()
38 {
39     if (flag_landgear == 0 && UserPlane.velocity > 350)
40     {
41         //retrazione del carrello
42         SimConnect_TransmitClientEvent(hSimConnect, SIMCONNECT_OBJECT_ID_USER, EVENT_LANDING_GEAR, 0,
43             SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
44         flag_landgear = 1;
45     }
46
47
48 // Funzione per trasformare le differenze di coordinate ECEF nel sistema body
49 std::vector<double> EcefToBody(double C2_latitude, double C2_longitude, double C2_altitude, double DRONE_latitude,
50     double DRONE_longitude, double DRONE_altitude, double C2_pitch, double C2_roll, double C2_heading)
51 {
52     // Converti gli angoli dell'origine in radianti
53     double phi0 = C2_latitude * PI / 180.0;
54     double lambda0 = C2_longitude * PI / 180.0;
55     double h0 = C2_altitude * .3048;
56     double phi = DRONE_latitude * PI / 180.0;
57     double lambda = DRONE_longitude * PI / 180.0;
58     double h = DRONE_altitude * .3048;
59     double pitch0 = C2_pitch * PI / 180.0;
```

```
60 double roll0 = C2_roll * PI / 180.0;
61 double heading0 = C2_heading * PI / 180.0;
62
63 //calcolo l'eccentricità al quadrato
64 double e2 = 1 - (WGS84_B * WGS84_B) / (WGS84_A * WGS84_A);
65
66 //trasformo in sistema ECEF C2Plane
67 double N_C2_ECEF = WGS84_A / sqrt(1 - e2 * sin(phi0) * sin(phi0));
68 double X_C2_ECEF = (N_C2_ECEF + h0) * cos(phi0) * cos(lambda0);
69 double Y_C2_ECEF = (N_C2_ECEF + h0) * cos(phi0) * sin(lambda0);
70 double Z_C2_ECEF = (N_C2_ECEF * (pow(WGS84_B, 2) / pow(WGS84_A, 2)) + h0) * sin(phi0);
71 //trasformo in sistema ECEF DRONE
72 double N_DRONE_ECEF = WGS84_A / sqrt(1 - e2 * sin(phi) * sin(phi));
73 double X_DRONE_ECEF = (N_DRONE_ECEF + h) * cos(phi) * cos(lambda);
74 double Y_DRONE_ECEF = (N_DRONE_ECEF + h) * cos(phi) * sin(lambda);
75 double Z_DRONE_ECEF = (N_DRONE_ECEF * (pow(WGS84_B, 2) / pow(WGS84_A, 2)) + h) * sin(phi);
76 //calcolo le differenze in ecef
77 double DeltaX = X_DRONE_ECEF - X_C2_ECEF;
78 double DeltaY = Y_DRONE_ECEF - Y_C2_ECEF;
79 double DeltaZ = Z_DRONE_ECEF - Z_C2_ECEF;
80
81
82
83
84 // Matrice di rotazione ECEF a NED
85 double R[3][3] = {
86     { -sin(phi0) * cos(lambda0), -sin(phi0) * sin(lambda0), cos(phi0) },
87     { -sin(lambda0), cos(lambda0), 0 },
88     { -cos(phi0) * cos(lambda0), -cos(phi0) * sin(lambda0), -sin(phi0) }
89 };
90
91 // Vettore delle differenze di coordinate ECEF
92 double dECEF[3] = { DeltaX, DeltaY, DeltaZ };
```

```
93
94 // Vettore delle coordinate NED risultanti
95 std::vector<double> NED(3, 0.0);
96
97 // Calcola le coordinate NED
98 for (int i = 0; i < 3; ++i) {
99     for (int j = 0; j < 3; ++j) {
100         NED[i] += R[i][j] * dECEF[j];
101     }
102 }
103
104
105 //ruotare attorno ai vari assi
106 // Matrice di rotazione per heading
107 double R_heading[3][3] = {
108     { cos(-heading0), -sin(-heading0), 0 },
109     { sin(-heading0), cos(-heading0), 0 },
110     { 0, 0, 1 }
111 };
112
113 // Matrice di rotazione per pitch
114 double R_pitch[3][3] = {
115     { cos(pitch0), 0, sin(pitch0) },
116     { 0, 1, 0 },
117     { -sin(pitch0), 0, cos(pitch0) }
118 };
119
120 // Matrice di rotazione per roll
121 double R_roll[3][3] = {
122     { 1, 0, 0 },
123     { 0, cos(roll0), -sin(roll0) },
124     { 0, sin(roll0), cos(roll0) }
125 };
```

```
126
127
128 // Applicare le rotazioni al sistema NED
129
130 // Rotazione per heading
131 std::vector<double>bodyAxes(3, 0.0);
132 for (int i = 0; i < 3; ++i) {
133     for (int j = 0; j < 3; ++j) {
134         bodyAxes[i] += R_heading[i][j] * NED[j];
135     }
136 }
137
138 // Rotazione per pitch
139 std::vector<double> bodyAxes_roll(3, 0.0);
140 for (int i = 0; i < 3; ++i) {
141     for (int j = 0; j < 3; ++j) {
142         bodyAxes_roll[i] += R_pitch[i][j] * bodyAxes[j];
143     }
144 }
145
146 // Rotazione per roll
147 std::vector<double> bodyAxes_final(3, 0.0);
148 for (int i = 0; i < 3; ++i) {
149     for (int j = 0; j < 3; ++j) {
150         bodyAxes_final[i] += R_roll[i][j] * bodyAxes_roll[j];
151     }
152 }
153 return bodyAxes_final;
154 }
155
156 // Funzione per trasformare le differenze di coordinate ECEF nel sistema body
157 std::vector<double> EcefToNEDPhi(double C2_latitude, double C2_longitude, double C2_altitude, double
```

```

158 {
159
160     // Converti gli angoli dell'origine in radianti
161     double phi0 = C2_latitude * PI / 180.0;
162     double lambda0 = C2_longitude * PI / 180.0;
163     double h0 = C2_altitude * .3048;
164     double phi = DRONE_latitude * PI / 180.0;
165     double lambda = DRONE_longitude * PI / 180.0;
166     double h = DRONE_altitude * .3048;
167     double heading0 = C2_heading * PI / 180.0;
168
169     //calcolo l'eccentricità al quadrato
170     double e2 = 1 - (WGS84_B * WGS84_B) / (WGS84_A * WGS84_A);
171
172     //trasformo in sistema ECEF C2Plane
173     double N_C2_ECEF = WGS84_A / sqrt(1 - e2 * sin(phi0) * sin(phi0));
174     double X_C2_ECEF = (N_C2_ECEF + h0) * cos(phi0) * cos(lambda0);
175     double Y_C2_ECEF = (N_C2_ECEF + h0) * cos(phi0) * sin(lambda0);
176     double Z_C2_ECEF = (N_C2_ECEF * (pow(WGS84_B, 2) / pow(WGS84_A, 2)) + h0) * sin(phi0);
177     //trasformo in sistema ECEF DRONE
178     double N_DRONE_ECEF = WGS84_A / sqrt(1 - e2 * sin(phi) * sin(phi));
179     double X_DRONE_ECEF = (N_DRONE_ECEF + h) * cos(phi) * cos(lambda);
180     double Y_DRONE_ECEF = (N_DRONE_ECEF + h) * cos(phi) * sin(lambda);
181     double Z_DRONE_ECEF = (N_DRONE_ECEF * (pow(WGS84_B, 2) / pow(WGS84_A, 2)) + h) * sin(phi);
182     //calcolo le differenze in ecef
183     double DeltaX = X_DRONE_ECEF - X_C2_ECEF;
184     double DeltaY = Y_DRONE_ECEF - Y_C2_ECEF;
185     double DeltaZ = Z_DRONE_ECEF - Z_C2_ECEF;
186
187
188
189     // Matrice di rotazione ECEF a NED
190

```

```
191 double RI[3][3] = {
192     { -sin(phi0) * cos(lambda0), -sin(phi0) * sin(lambda0), cos(phi0) },
193     { -sin(lambda0), cos(lambda0), 0 },
194     { -cos(phi0) * cos(lambda0), -cos(phi0) * sin(lambda0), -sin(phi0) }
195 };
196
197 // Vettore delle differenze di coordinate ECEF
198 double dECEF[3] = { DeltaX, DeltaY, DeltaZ };
199
200 // Vettore delle coordinate NED risultanti
201 std::vector<double> NED(3, 0.0);
202
203 // Calcola le coordinate NED
204 for (int i = 0; i < 3; ++i) {
205     for (int j = 0; j < 3; ++j) {
206         NED[i] += RI[i][j] * dECEF[j];
207     }
208 }
209
210
211 //ruotare attorno ai vari assi
212 // Matrice di rotazione per heading
213 double R_heading[3][3] = {
214     { cos(-heading0), -sin(-heading0), 0 },
215     { sin(-heading0), cos(-heading0), 0 },
216     { 0, 0, 1 }
217 };
218
219
220 // Applicare le rotazioni al sistema NED
221
222 // Rotazione per heading
223 std::vector<double> bodyAxes(3, 0.0);
```

```

224     for (int i = 0; i < 3; ++i) {
225         for (int j = 0; j < 3; ++j) {
226             bodyAxes[i] += R_heading[i][j] * NED[j];
227         }
228     }
229     return bodyAxes;
230 }
231
232 // Funzione per trasformare l
233 std::vector<double> RelativePos(double C2_latitude, double C2_longitude, double C2_altitude, double DRONE_latitude, double DRONE_longitude, double DRONE_altitude, double C2_pitch, double C2_roll, double C2_heading)
234 {
235     // Converti gli angoli dell'origine in radianti
236     double phi0 = C2_latitude * PI / 180.0;
237     double lambda0 = C2_longitude * PI / 180.0;
238     double pitch0 = C2_pitch * PI / 180.0;
239     double roll0 = C2_roll * PI / 180.0;
240     //double heading0 = C2_heading * PI / 180.0;
241     //asse z rivolto verso l'alto e x avanti, y sinistra
242     double heading0 = -phi0 * PI / 180.0;
243
244     std::vector<double> NED(3, 0.0);
245     NED[0] = DRONE_latitude - C2_latitude;
246     NED[1] = DRONE_longitude - C2_longitude;
247     NED[2] = DRONE_altitude - C2_altitude;
248
249     //ruotare attorno ai vari assi
250     // Matrice di rotazione per heading
251     double R_heading[3][3] = {
252         { cos(heading0), -sin(heading0), 0 },
253         { sin(heading0), cos(heading0), 0 },
254         { 0, 0, 1 }
255     };

```

```
256
257 // Matrice di rotazione per pitch
258 double R_pitch[3][3] = {
259     { cos(pitch0), 0, sin(pitch0) },
260     { 0, 1, 0 },
261     { -sin(pitch0), 0, cos(pitch0) }
262 };
263
264 // Matrice di rotazione per roll
265 double R_roll[3][3] = {
266     { 1, 0, 0 },
267     { 0, cos(roll0), -sin(roll0) },
268     { 0, sin(roll0), cos(roll0) }
269 };
270
271
272 // Applicare le rotazioni al sistema NED
273
274 // Rotazione per heading
275 std::vector<double>bodyAxes(3, 0.0);
276 for (int i = 0; i < 3; ++i) {
277     for (int j = 0; j < 3; ++j) {
278         bodyAxes[i] += R_heading[i][j] * NED[j];
279     }
280 }
281
282 // Rotazione per pitch
283 std::vector<double> bodyAxes_roll(3, 0.0);
284 for (int i = 0; i < 3; ++i) {
285     for (int j = 0; j < 3; ++j) {
286         bodyAxes_roll[i] += R_pitch[i][j] * bodyAxes[j];
287     }
288 }
```

```
289
290 // Rotazione per roll
291 std::vector<double> bodyAxes_final(3, 0.0);
292 for (int i = 0; i < 3; ++i) {
293     for (int j = 0; j < 3; ++j) {
294         bodyAxes_final[i] += R_roll[i][j] * bodyAxes_roll[j];
295     }
296 }
297 return bodyAxes;
298 }
299
```

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:header file with all write/read functions
5 //-----
6 #pragma once
7 #include "globalvar.h"
8
9 void LetturaConfigs();
10 void StampaConfigs(double daticonfigs[2]);
11 void CreaFile();
12 void ApriFile();
13 void StampaFile();
14 void ChiudiFile();
15
16 int AskYesNo();
17 int SelectTest();
18
19
20
21 //funzione che legge i dati dal file input
22 void LetturaConfigs()
23 {
24     configs_file;
25     err0 = fopen_s(&configs_file, "inputs/configs.txt", "r");
26     if (err0 != 0)
27     {
28         printf("Error opening configs file.\n");
29         system("pause");
30         exit(1);
31     }
32     else {
33         printf("Config file correctly loaded.\n");
```

```
34 int i = 0, max_length = 200, max_data_configs = 3;
35 char buf[200];
36 double x;
37 while (fgets(buf, max_length, configs_file) != NULL && i < max_data_configs)
38 {
39     if (buf[0] == '*')
40     {
41         continue;
42     }
43     else if (buf[0] != '*')
44     {
45         int k = sscanf_s(buf, "%lf", &x);
46         daticonfigs[i] = x;
47         i++;
48     }
49 }
50 quota_crociera = daticonfigs[0];
51 heading_crociera = daticonfigs[1];
52 initial_heading = daticonfigs[2];
53 fclose(configs_file);
54 }
55
56 }
57
58 //funzione che stampa su console i dati di configurazione
59 void StampaConfigs(double daticonfigs[2])
60 {
61     int max_data_configs = 3;
62     printf("\nData configs:\n");
63     for (int i = 0; i < max_data_configs; i++)
64     {
65         printf("%lf\n", daticonfigs[i]);
66     }
```

```
67 }
68
69 //crea due file di output uno per C2 e uno per il drone con la data e ora attuali
70 void CreaFile()
71 {
72     const char* filepath = "outputs/";
73     time_t timestamp;
74     struct tm datetime;
75     time(&timestamp);
76     char output_date[50]{};
77     if (localtime_s(&datetime, &timestamp) == 0)
78     {
79         strftime(output_date, 50, "%F_%T", &datetime);
80     }
81     strcpy_s(filepath, filepath);
82     strcat_s(filepath, "C2_");
83     strcat_s(filepath, output_date);
84     strcat_s(filepath, ".txt");
85     for (char& ch : filepath)
86     {
87         if (ch == ':')
88         {
89             ch = '-';
90         }
91     }
92     strcpy_s(filepath2, filepath);
93     strcat_s(filepath2, "DRONE_");
94     strcat_s(filepath2, output_date);
95     strcat_s(filepath2, ".txt");
96     for (char& ch : filepath2)
97     {
98         if (ch == ':')
99         {
```

```
100     ch = '-';
101 }
102 }
103 flag_created = 1;
104 }
105
106 //Apro i due file di output precedentemente creati
107 void ApriFileC()
108 {
109     output_file;
110     err = fopen_s(&output_file, fileTitle, "w");
111     if (err == 0)
112     {
113         printf("Creazione del file output: ");
114         printf("%s\n", fileTitle);
115         fprintf(output_file, "%*SimTime\\tLatitude\\tLongitude\\tAltitude\\tComAltitude\\tPitch\\t\\tCompPitch\\tBank\\t\\tComBank\\t\\tHeading\\t\\tComHeading\\tElevator\\tRudder\\t\\tVelocity\\tComVelocity\\tDOWN\\t\\tcomDOWN\\t\\tEAST\\t\\tcomeEAST\\t\\tNORTH\\t\\tcomNORTH\\t\\tthrottle\\taccelerationX\\taccelerationY\\taccelerationZ\\twindX\\twindY\\twindZ");
116
117     }
118     output_file2;
119     err2 = fopen_s(&output_file2, fileTitle2, "w");
120     if (err2 == 0)
121     {
122         printf("Creazione del file output: ");
123         printf("%s\n", fileTitle);
124         fprintf(output_file2, "%*SimTime\\tLatitude\\tLongitude\\tAltitude\\tComAltitude\\tPitch\\t\\tCompPitch\\tBank\\t\\tComBank\\t\\tHeading\\t\\tComHeading\\tElevator\\tRudder\\t\\tVelocity\\tComVelocity\\tDOWN\\t\\tcomDOWN\\t\\tEAST\\t\\tcomeEAST\\t\\tNORTH\\t\\tcomNORTH\\t\\tthrottle\\taccelerationX\\taccelerationY\\taccelerationZ\\twindX\\twindY\\twindZ");
125     }
126 }
```



```
146  if (err == 0)
147  {
148      fclose(output_file);
149  }
150  if (err2 == 0)
151  {
152      fclose(output_file2);
153  }
154  }
155
156  int AskYesNo()
157  {
158      int flag_true = 0;
159      char input[2] = { 'U', '\0' };
160      printf("(Y/N): ");
161      while (input[0] == 'U') {
162
163          if (scanf_s("%1s", input, (unsigned)_countof(input)) == 1) {
164              // Converti l'input in maiuscolo per gestire diverse maiuscole/minuscole
165              input[0] = toupper(input[0]);
166
167              if (strcmp(input, "Y") == 0) {
168
169                  flag_true = 1;
170              }
171              else if (strcmp(input, "N") == 0) {
172
173                  flag_true = 0;
174              }
175              else {
176                  input[0] = 'U';
177                  printf("Input non valido, (Y/N):");
178              }
```

```
179     }
180     else {
181         // Gestisci l'errore se scanf_s non riesce a leggere l'input
182         printf("Errore nella lettura dell'input.\n");
183         // Svuota il buffer di input
184         int c;
185         while ((c = getchar()) != '\n' && c != EOF);
186         input[0] = 'U';
187     }
188 }
189
190 return flag_true;
191 }
192 int SelectTest()
193 {
194     int flag = 0;
195     char input[2] = { 'U', '\0' };
196
197     while (input[0] == 'U') {
198         printf("[1] PITCH \n[2] BANK\n[3] ALTITUDE\n[4] HEADING\n[5] VELOCITY\n[6] NORTH\n[7] EAST\n[8] DOWN\n");
199         if (scanf_s("%1s", input, (unsigned)_countof(input)) == 1) {
200             // Converti l'input in maiuscolo per gestire diverse maiuscole/minuscole
201             input[0] = toupper(input[0]);
202             if (strcmp(input, "1") == 0) {
203                 printf("Avviamento test autopilota di mantenimento del pitch\n");
204                 flag = 101;
205             }
206             else if (strcmp(input, "2") == 0) {
207                 printf("Avviamento test autopilota di mantenimento del bank\n");
208                 flag = 102;
209             }
210             else if (strcmp(input, "3") == 0) {
211                 printf("Avviamento test autopilota di mantenimento della quota\n");
```

```
212     flag = 103;
213 }
214 else if (strcmp(input, "4") == 0) {
215     printf("Avviamento test autopilota di mantenimento della direzione\n");
216     flag = 104;
217 }
218 else if (strcmp(input, "5") == 0) {
219     printf("Avviamento test autopilota di mantenimento della velocita\n");
220     flag = 105;
221 }
222 else if (strcmp(input, "6") == 0) {
223     printf("Avviamento test autopilota di mantenimento della posizione relativa NORD\n");
224     flag = 106;
225 }
226 else if (strcmp(input, "7") == 0) {
227     printf("Avviamento test autopilota di mantenimento della posizione relativa EST\n");
228     flag = 107;
229 }
230 else if (strcmp(input, "8") == 0) {
231     printf("Avviamento test autopilota di mantenimento della posizione relativa DOWN\n");
232     flag = 108;
233 }
234 else {
235     input[0] = 'U';
236     printf("Input non valido, Ripetere la scelta:\n");
237 }
238 }
239 else {
240     // Gestisci l'errore se scanf_s non riesce a leggere l'input
241     printf("Errore nella lettura dell'input.\n");
242     // Svuota il buffer di input
243     int c;
244     while ((c = getchar()) != '\n' && c != EOF);
```

E:\TESI\00 - Prepar3dSDK\PERSONAL\C2 PLANE\headers\readwrite.h

245 input[0] = 'U';

246 }

247 }

248

249 return flag;

250 }

```
1 //-----
2 //      Title:MASTER DEGREE THESIS by ANTONIO SCAZZI
3 //
4 //      Description:header file with all global variables
5 //-----
6 #pragma once
7 #include <windows.h>
8 #include <tchar.h>
9 #include <stdio.h>
10 #include <strsafe.h>
11 #include <vector>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <cmath>
15 #include <fstream>
16 #include <ctime>
17 #include <cstring>
18 #include <math.h>
19 #include "SimConnect.h"
20 #include <iostream>
21
22
23 //define the event group
24 enum GROUP_ID {
25     GROUP_0, GROUP_1, GROUP_2
26 };
27 //define the event id
28 enum EVENT_ID {
29     EVENT_SIM_PAUSED, EVENT_SIM_UNPAUSED, EVENT_SIM_START, EVENT_SIM_STOP, EVENT_THROTTLE_SET, EVENT_RUDDER_SET,
30     EVENT_AILERON_SET, EVENT_ELEVATOR_SET, EVENT_FLAPS_SET, EVENT_PARKING_BRAKES_SET, EVENT_LANDING_GEAR,
31     EVENT_0, EVENT_1, EVENT_2, EVENT_3, EVENT_4, EVENT_5, EVENT_6, EVENT_7, EVENT_TRANSPONDER, EVENT_PRINT_1,
32     EVENT_PRINT_2
33 };
34 }
```

```
31 //define the input id
32 enum INPUT_ID {
33     INPUT_0, INPUT_1, INPUT_2, INPUT_3, INPUT_4, INPUT_5
34 };
35 //define data structure od the sim data
36 enum DATA_DEFINE_ID {
37     DEFINITION_1, DEFINITION_2
38 };
39 //define how many data request we need
40 enum DATA_REQUEST_ID {
41     REQUEST_1, REQUEST_0, REQUEST_2, REQUEST_3
42 };
43 // define the data structure
44 struct ObjectDataStruct
45 {
46     char title[256];
47     double simtime;
48     double altitude;
49     double latitude;
50     double longitude;
51     double pitch;
52     double bank;
53     double heading;
54     double velocity;
55     double velocityX;
56     double velocityY;
57     double velocityZ;
58     double elevator;
59     double rudder;
60     double aileron;
61     double throttle;
62     double transponderCode;
63     double alpha;
```

```
64 double beta;
65 double pitchrate;
66 double rollrate;
67 double yawrate;
68 double accelerationZ;
69 double accelerationX;
70 double accelerationY;
71 double windX;
72 double windY;
73 double windZ;
74 };
75
76 //flags
77 int flag_created = 0, flag_quit = 0, flag_throttle = 0, flag_flap = 0, flag_isrunning = 0, flag_parkingbrake = 0,
    flag_initial = 0, flag_output = 0, flag_decollo = 0, flag_landgear = 0, flag_dronefound = 0, flag_partenzadrone = 0,
    flag_isincruise = 0, flag_stampa = 0, flag_initialgroundcheck = 0, flag_roll = 0, flag_pitch = 0,
    flag_comandoverride = 0, flag_test=0, flag_timer=0;
78 //definisco pi greco
79 double PI = 3.14159265358979323846;
80
81 //variabili da pulire
82 double Vz;
83 double heading;
84 double initial_latitude, initial_longitude;
85 double vs;
86
87 //variabili globali configurazione
88 double daticonfigs[3];
89 double quota_crociera = 0;
90 double heading_crociera = 0;
91 double initial_heading = 0;
92 double initial_transponder = 0;
93 double initial_simtime = 0, initial_simtime2=0;
```

```
94
95
96
97 //title of output file
98 char fileTitle[50];
99 char fileTitle2[50];
100
101 FILE* configs_file;
102 FILE* output_file;
103 FILE* output_file2;
104 errno_t err;
105 errno_t err2;
106
107 errno_t err0;
108
109 //other variables
110 ObjectDataStruct* V1;
111 ObjectDataStruct* V2;
112 ObjectDataStruct UserPlane, OtherPlane;
113 HANDLE hSimConnect = NULL;
114 HRESULT hr;
115 DWORD UserID;
116 DWORD ObjectID2;
117
118
119
120 const double WGS84_A = 6378137.0; // Earth's semi-major axis in meters
121 const double WGS84_B = 6356752.3; // Earth's semi-minor axis in meters
122
123 struct AircraftPosition
124 {
125     double Heading;
126 };
```

E:\TESI\00 - Prepar3dSDK\PERSONAL\C2 PLANE\headers\globalvar.h

127 **AircraftPosition** initial;

128

129 **double** bankang = 0, pitchang = 0, altitude = 0, testheading = 0, comtestheading = 0,
comvelocity=0,comDOWN=0,comEAST = 0,comNORD = 0;

130 **ObjectDataStruct** TestPlane;

131 **std::vector<double>** testNED(3, 0.0);

132

133 **double** deltetime = 0, previoustime = 0, testVel = 0;