Politecnico Di Bari

Dipartimento di Ingegneria Elettrica e dell'Informazione

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Tema d'Anno in
Progettazione dei Sistemi Avionici

Simulazione e Monitoraggio di Strumentazione di Bordo per Velivoli in Ambiente MATLAB

Professore:
Prof. Antonio Satriano

Studente:
Angelo Mastrangelo

Anno Accademico 2024–2025

Indice

1 Introduzione

La simulazione e il monitoraggio della strumentazione di bordo rappresentano un pilastro fondamentale nello sviluppo e nella comprensione dei sistemi avionici, che costituiscono lo scheletro della tecnologica degli aeromobili moderni. Questi sistemi, essenziali per la sicurezza e l'efficienza del volo, forniscono ai piloti informazioni in tempo reale su parametri critici come altitudine, velocità, tasso di salita, direzione e assetto. In un'epoca in cui la tecnologia avanza a ritmi incalzanti, la possibilità di riprodurre il comportamento di tali strumenti in un ambiente virtuale offre un'opportunità unica per testare, analizzare e ottimizzare le loro prestazioni senza i rischi e i costi associati alle prove reali. Questo progetto, si propone di sviluppare un simulatore in ambiente MATLAB capace di modellare e monitorare la strumentazione di bordo di un velivolo leggero, con particolare attenzione a strumenti chiave come l'altimetro, il velocimetro, il variometro, la bussola magnetica e l'indicatore di assetto (pitch). Attraverso un'interfaccia grafica interattiva, una traiettoria 3D in tempo reale e un'architettura software modulare, il simulatore consente di visualizzare il comportamento degli strumenti in scenari di volo dinamici, offrendo uno strumento didattico e analitico.

Il progetto mira a integrare competenze teoriche e pratiche per affrontare sfide complesse, con un focus sull'apprendimento pratico delle dinamiche di volo e sull'uso di MATLAB come piattaforma di simulazione. La scelta di MATLAB come ambiente di sviluppo è motivata dalla sua potenza computazionale, dalla versatilità nella gestione di calcoli matriciali e dalla capacità di creare interfacce grafiche intuitive tramite strumenti come `uifigure` e `uigauge`. Inoltre, MATLAB consente di implementare facilmente modelli fisici semplificati e di visualizzare dati in tempo reale, caratteristiche fondamentali per un simulatore avionico destinato a scopi didattici.

1.1 Obiettivi del progetto

L'obiettivo principale del progetto è sviluppare un simulatore in MATLAB che consenta di modellare e monitorare in tempo reale la strumentazione di bordo di un velivolo, con particolare attenzione a cinque strumenti fondamentali: l'altimetro, il velocimetro, il variometro, la bussola magnetica e l'indicatore di assetto. Il sistema è progettato per riprodurre il comportamento di questi strumenti in tre scenari di volo distinti:

- **Modalità "Manuale"**: consente all'utente di controllare il velivolo tramite input da tastiera (frecce per pitch e direzione, tasti M/N per velocità), con una durata indefinita finché non viene interrotta manualmente tramite il pulsante "Stop".
- **Modalità "Standard"**: simula un volo realistico di 30 secondi, suddiviso in decollo graduale (0–10 s, altitudine fino a 24 m), crociera stabile con una virata di 10° (10–20 s), e discesa controllata (20–30 s, ritorno a 0 m).
- **Modalità "Turbolenza"**: introduce variazioni casuali e sinusoidali nei parametri di volo (es. tasso di salita, pitch, direzione) per simulare condizioni turbolente, anch'essa limitata a 30 secondi.

La simulazione si basa su modelli fisici semplificati, che descrivono le dinamiche del velivolo, con un passo temporale di 0.1 s per garantire aggiornamenti fluidi. L'interfaccia grafica, realizzata con `uifigure` e `uigauge`, presenta un layout che richiama un cockpit reale, con gauge analogici per ciascun strumento e un grafico 3D che visualizza la

traiettoria del velivolo in tempo reale in funzione delle coordinate x, y e altitudine. Un'analisi post-simulazione genera grafici 3D per parametri come altitudine, pitch e velocità, consentendo una valutazione approfondita delle prestazioni.

Ulteriori obiettivi includono l'implementazione di funzionalità avanzate per arricchire la simulazione. I dati di volo vengono registrati in una tabella MATLAB (`flightLog`) contenente parametri come tempo, altitudine, velocità, direzione e coordinate, permettendo un'analisi dettagliata post-simulazione. L'architettura software è modulare, con un main script (`main.m`) che coordina le operazioni e funzioni annidate che gestiscono compiti specifici, garantendo chiarezza e facilità di manutenzione.

2 Cenni sui sistemi avionici

I sistemi avionici rappresentano il cuore tecnologico degli aeromobili moderni, integrando elettronica, informatica e ingegneria aeronautica per garantire la sicurezza, l'efficienza e la precisione durante il volo. Questi sistemi comprendono una vasta gamma di strumenti e dispositivi, dai sensori che acquisiscono dati ambientali agli schermi che presentano informazioni ai piloti, fino ai computer che elaborano comandi in tempo reale. La strumentazione di bordo, come l'altimetro, il velocimetro, il variometro, la bussola magnetica e l'indicatore di assetto (pitch), è progettata per fornire informazioni critiche sullo stato del velivolo, permettendo ai piloti di prendere decisioni informate anche in condizioni avverse.

L'avionica moderna si distingue per la sua capacità di integrare tecnologie avanzate, come il *fly-by-wire*, che sostituisce i comandi meccanici con segnali elettrici, e il *fly-by-light*, che utilizza fibre ottiche per una trasmissione dati più rapida e resistente alle interferenze. Questi progressi hanno reso gli aeromobili più leggeri, efficienti e sicuri, ma hanno anche aumentato la complessità dei sistemi di controllo. La simulazione gioca un ruolo cruciale in questo contesto, consentendo di testare il comportamento degli strumenti in scenari virtuali prima della loro implementazione reale. MATLAB, con la sua versatilità, potenza computazionale e capacità di creare interfacce grafiche tramite `uifigure` e `uigauge`, si è affermato come uno strumento indispensabile per modellare e analizzare questi sistemi, grazie alla gestione di calcoli complessi e alla visualizzazione intuitiva di dati in tempo reale, come dimostrato dal simulatore di questo progetto.

2.1 Evoluzione e classificazione dei sistemi avionici

L'evoluzione dei sistemi avionici è strettamente connessa allo sviluppo dell'aviazione moderna. Nei primi aeromobili, la strumentazione era limitata a semplici indicatori meccanici. Con il progredire della tecnologia elettronica e informatica, si è passati a strumenti elettromeccanici e successivamente a sistemi digitali integrati. Oggi, i moderni aerei commerciali e militari adottano architetture basate su avionica integrata (*Integrated Modular Avionics*, IMA), in cui diverse funzionalità sono consolidate in moduli centralizzati, riducendo peso, consumo energetico e complessità del cablaggio.

I sistemi avionici possono essere classificati in diverse categorie funzionali:

- **Navigazione**: comprendono GPS, radiofari VOR/DME e sistemi inerziali.
- **Comunicazione**: includono radio VHF/UHF e trasponder.
- **Controllo di volo**: coinvolgono il *fly-by-wire* e gli attuatori.
- **Monitoraggio**: sistemi che riportano lo stato del motore, carburante, temperatura, ecc.
- **Display e interfaccia uomo-macchina (HMI)**: dai classici strumenti analogici agli schermi multifunzione (MFD).

2.2 Utilizzi della strumentazione avionica

La strumentazione avionica trova applicazione in una vasta gamma di contesti, dalla navigazione aerea alla gestione delle emergenze. L'altimetro, simulato nel progetto con un

range di 0–200 m, è essenziale per determinare l'altitudine del velivolo, fornendo dati cruciali durante il decollo, la crociera e l'atterraggio. Il velocimetro, con un range di 0–100 m/s, misura la velocità relativa all'aria, permettendo ai piloti di mantenere il velivolo entro i limiti operativi. L'indicatore di assetto (pitch), con limiti di $-30°$ a $30°$, è indispensabile in condizioni di scarsa visibilità, indicando l'inclinazione del velivolo. La bussola magnetica, con un range di $0°$–$360°$, fornisce informazioni sulla direzione, mentre il variometro, con limiti di -10 a 10 m/s, misura il tasso di salita o discesa, aiutando a controllare la traiettoria verticale.

Questi strumenti non si limitano all'aviazione commerciale, ma trovano impiego anche in aerei militari, droni e velivoli leggeri. Ad esempio, nei droni, la strumentazione è integrata con sistemi di controllo automatico per missioni autonome, mentre nell'aviazione militare gli strumenti devono garantire prestazioni elevate in ambienti estremi. Il simulatore MATLAB di questo progetto replica tali dinamiche, permettendo di esplorare queste applicazioni in un ambiente controllato.

2.3 Componenti della strumentazione avionica

La strumentazione avionica si basa su un'integrazione sinergica di componenti hardware e software, ciascuno progettato per svolgere un ruolo specifico. I sensori, come il barometro per l'altimetro o i giroscopi MEMS per il pitch, acquisiscono dati ambientali (pressione, accelerazione, orientamento). I dati vengono elaborati da microprocessori che eseguono algoritmi in tempo reale, presentati tramite display analogici o digitali. Nel simulatore, questi componenti sono emulati con modelli fisici semplificati , visualizzati tramite una GUI con `uigauge`. L'alimentazione e la robustezza sono emulate con un design che garantisce continuità operativa, mentre il software gestisce l'interfaccia utente e il logging dei dati in `flightLog`. La strumentazione di bordo indica gli strumenti utilizzati dal pilota al fine di ricevere le principali informazioni necessarie al mantenimento in volo del velivolo sulla rotta di navigazione, includendo dunque i dispositivi di comunicazione per la gestione del traffico aereo che permettono di interagire con il velivolo. Gli strumenti di bordo possono essere suddivisi in tre gruppi principali:
1. Strumenti di volo, i quali indicano assetto, quota e velocità necessarie ad eseguire il volo anche in condizioni di scarsa visibilità esterna;
2. Strumenti dell'apparato propulsore, grazie ai quali i piloti ricevono informazioni in tempo reale, riguardanti lo stato di funzionamento dei motori;
3. Strumenti di navigazione, che svolgono la funzione di mantenere l'aeroplano sulla rotta desiderata.

2.3.1 Altimetro

L'altimetro è un barometro che misura la pressione atmosferica, trasformando la lettura di pressione in misura di quota grazie al rapporto tra pressione atmosferica e altezza sul livello del mare alla quale viene misurata. Esso è costituito da una cassa a tenuta stagna collegata all'esterno tramite una presa statica, entro la quale si trova una capsula chiusa ermeticamente. All'interno della capsula viene creata una depressione che è contrastata da una molla, la quale spinge contro le pareti, in modo da tenere la capsula nella posizione di riposo e segnare zero quando si trova in condizioni di atmosfera standard al livello del mare. All'aumentare della quota, la pressione diminuisce e la capsula si dilata e tramite un sistema di leve e ingranaggi avviene la trasformazione del moto rettilineo di dilatazione della capsula in moto rotatorio degli indici.

2.3.2 L'orizzonte artificiale

È lo strumento fondamentale per condurre un volo strumentale, ovvero privo di visibilità dell'orizzonte naturale. Esso è costituito da un giroscopio a tre gradi di libertà posto davanti al pilota con l'asse di rotazione verticale e rotore parallelo alla superficie terrestre indipendentemente dall'assetto assunto dall'aeroplano. Al rotore è applicata la linea dell'orizzonte che divide il "cielo" segnato in azzurro dalla "terra" segnata in marrone. Una sagoma che simboleggia il velivolo, solidale con l'involucro dello strumento, e una serie di marcature, consentono al pilota di visualizzare gli assetti via via assunti dall'aeroplano; quindi, di manovrarlo come se visualizzassero l'orizzonte naturale.

2.3.3 Strumenti giroscopici

Parte fondamentale della famiglia degli strumenti di volo è costituita dagli strumenti giroscopici, così chiamati perché il giroscopio è l'elemento meccanico alla base del loro funzionamento. Essi sono l'orizzonte artificiale o indicatore di assetto, il direzionale o indicatore di prua e l'indicatore di virata. I rotori dei loro giroscopi sono mantenuti in rotazione da getti d'aria generati da una pompa o da motorini elettrici. Tali strumenti hanno visto un netto miglioramento tecnologico: a partire dagli anni '80, i giroscopi meccanici sono stati sostituiti con quelli ad anello laser. Ciò prevedeva che non si muovesse più un rotore, bensì un fascio di luce polarizzata, che tramite un sistema di specchi posti ai vertici di un triangolo, effettuava un percorso triangolare venendo quindi riflessa. Ogni minimo movimento intorno ai suoi tre assi veniva rilevato dal computer e inviato ai display degli strumenti, divenuti così di gran lunga più semplici, leggeri ed affidabili.

Figura 1: Orizzonte artificiale

Figura 2: temperatura dei gas di scarico

Figura 3: Altimetro

Figura 4: velocita dell'aria

2.4 Leggi fisiche della strumentazione

Il funzionamento della strumentazione si basa su principi fisici. L'altimetro utilizza la formula barometrica

$$P = P_0 \cdot \exp\left(-\frac{g \cdot M \cdot h}{R \cdot T}\right), \tag{1}$$

semplificata nel simulatore con un modello lineare basato su tasso di salita. Il velocimetro si basa su

$$v = \sqrt{\frac{2 \cdot (P_1 - P_0)}{\rho}}, \tag{2}$$

approssimato con una variazione lineare. L'indicatore di assetto usa equazioni di Euler per rollio e beccheggio, mentre il variometro calcola

$$V_v = \frac{dh}{dt}, \tag{3}$$

tutti adattati per un velivolo leggero nel simulatore.

3 Cenni sulla simulazione in MATLAB

MATLAB, acronimo di Matrix Laboratory, è un ambiente di sviluppo consolidato e ampiamente utilizzato per la simulazione, l'analisi e la progettazione di sistemi complessi in ambito scientifico e ingegneristico. La sua capacità di eseguire operazioni matriciali con elevata efficienza, svolgere calcoli numerici avanzati e produrre visualizzazioni grafiche dettagliate lo rende uno strumento ideale per applicazioni avioniche, dove precisione e rapidità computazionale sono fondamentali. In questo progetto, MATLAB è stato scelto per sviluppare un simulatore avionico in grado di modellare e monitorare la strumentazione di bordo di un velivolo, con particolare attenzione a strumenti come l'altimetro, il velocimetro, il variometro, la bussola magnetica e l'indicatore di assetto (pitch). Questo capitolo esplora il ruolo di MATLAB nel progetto, analizzandone i fondamenti teorici e pratici, la struttura del simulatore, le sue funzionalità operative e le potenziali applicazioni. L'obiettivo è dimostrare come MATLAB non solo faciliti la creazione di simulazioni realistiche, ma rappresenti anche un ponte tra teoria accademica e applicazioni pratiche nel campo dell'avionica.

L'adozione di MATLAB è motivata dalla sua capacità di integrare diverse funzionalità in un unico ambiente: dalla modellazione matematica alla programmazione strutturata, dalla gestione di dati complessi alla creazione di interfacce grafiche interattive (GUI). Funzionalità come la visualizzazione in tempo reale della traiettoria in 3D, il logging dei dati e la possibilità di simulare scenari dinamici arricchiscono il simulatore, rendendolo rappresentativo delle complessità dei sistemi avionici reali.

3.1 Fondamenti di MATLAB per l'avionica

MATLAB si distingue per la sua sintassi intuitiva e per la vasta gamma di strumenti integrati che ne estendono le funzionalità a settori specifici come l'aeronautica. Sebbene questo progetto non utilizzi direttamente toolbox specializzate come l'Aerospace Toolbox o l'Aerospace Blockset, sfrutta le capacità computazionali di MATLAB per implementare modelli fisici semplificati che descrivono il comportamento degli strumenti di bordo e per creare un'interfaccia grafica interattiva che simula un cockpit virtuale. La simulazione si basa su un approccio numerico, in cui i dati degli strumenti (altitudine, velocità, tasso di salita, direzione e assetto) vengono aggiornati a ogni passo temporale, riflettendo le variazioni dinamiche del velivolo in risposta agli scenari di volo simulati.

Un aspetto fondamentale di MATLAB è la sua efficienza nella gestione delle operazioni matriciali, cruciale per simulare sistemi avionici che coinvolgono variabili interdipendenti. Ad esempio, il calcolo della traiettoria del velivolo in 3D richiede l'aggiornamento continuo delle coordinate x, y e z (altitudine) in base alla velocità e alla direzione (heading), operazioni che MATLAB esegue rapidamente grazie alla sua natura matriciale. Inoltre, MATLAB supporta la creazione di GUI attraverso strumenti come `uifigure` e `uigauge`, utilizzati in questo progetto per visualizzare gli strumenti in tempo reale. Queste funzionalità permettono di simulare un'esperienza di volo immersiva, con aggiornamenti dinamici che riflettono le condizioni di volo.

Dal punto di vista teorico, MATLAB si basa su un paradigma di programmazione strutturata, conforme al teorema di Böhm-Jacopini, secondo cui ogni algoritmo può essere costruito utilizzando i costrutti fondamentali di sequenza, selezione e iterazione. Nel simulatore, questo principio è evidente nella struttura del codice: un main script (`main.m`) coordina le operazioni, mentre funzioni annidate gestiscono compiti specifici, come l'ag-

giornamento degli strumenti o la simulazione degli scenari di volo. La modularità del codice, unita a una documentazione chiara, garantisce la manutenibilità e la scalabilità del sistema, consentendo future espansioni, come l'aggiunta di nuovi strumenti o scenari.

3.2 Struttura e funzionamento del simulatore

Il simulatore è stato progettato con un'architettura modulare che separa le componenti logiche, computazionali e grafiche, garantendo chiarezza e flessibilità. Il cuore del sistema è il main script (`main.m`), che inizializza i parametri, gestisce gli scenari di volo e coordina l'interfaccia grafica. Funzioni annidate all'interno del main script si occupano di attività specifiche, come l'elaborazione dei comandi manuali, la simulazione degli scenari automatici e l'aggiornamento dei grafici. Il simulatore supporta tre modalità operative: "Manuale", "Standard" e "Turbolenza", ciascuna con caratteristiche distinte.
Il main script si articola in diverse sezioni logiche:

1. Inizializzazione della GUI: Viene creata una finestra grafica (`uifigure`) con dimensioni, che ospita cinque strumenti: altimetro, velocimetro, variometro, bussola e l'indicatore di assetto. Questi strumenti sono implementati utilizzando `uigauge`, configurati con limiti realistici per un velivolo leggero. Un menu a tendina (`uidropdown`) permette di selezionare lo scenario di volo.

2. Definizione degli scenari di volo: Modalità "Standard" e "Turbolenza":

La simulazione dura esattamente 30 s, con un passo temporale di 0.1 s, per un totale di 300 iterazioni. Un ciclo `for` aggiorna i parametri di volo in base al tempo. Nel caso dello scenario "Standard", il volo è strutturato in tre fasi:

- Decollo (0–10 s): L'altitudine aumenta con un tasso di salita che varia linearmente da $2\,\mathrm{m\,s^{-1}}$ a $2.8\,\mathrm{m\,s^{-1}}$, raggiungendo circa 24 m. La velocità cresce a $2\,\mathrm{m\,s^{-2}}$, arrivando a $20\,\mathrm{m\,s^{-1}}$. Il beccheggio segue un andamento sinusoidale, con un massimo di 5°.

- Crociera (10–20 s): L'altitudine si stabilizza a 24 m, la velocità aumenta a $40\,\mathrm{m\,s^{-1}}$, e il velivolo esegue una virata graduale di 10° (a $1\,^\circ\,\mathrm{s^{-1}}$). Il beccheggio rimane nullo.

-Discesa (20–30 s): L'altitudine diminuisce con un tasso di discesa da $-2\,\mathrm{m\,s^{-1}}$ a $-2.8\,\mathrm{m\,s^{-1}}$, tornando a 0 m. La velocità raggiunge $60\,\mathrm{m\,s^{-1}}$, e il beccheggio varia sinusoidalmente fino a −5°.

- Modalità "Turbolenza": La simulazione dura 30 s, con variazioni casuali e sinusoidali nei parametri di volo, simulando condizioni di turbolenza. Ad esempio, il tasso di salita varia secondo $\mathrm{roc} = 2\sin(0.5t) + 0.8 \cdot \mathrm{randn}$, e il beccheggio oscilla con un'ampiezza di ±10°.

-Modalità "Manuale": La simulazione non ha una durata predefinita e continua indefinitamente finché non viene interrotta dall'utente tramite il pulsante "Stop". L'utente controlla il velivolo tramite input da tastiera (frecce per beccheggio e direzione, tasti M/N per la velocità).

3. Visualizzazione grafica in tempo reale: Un grafico 3D (`uiaxes`) mostra la traiettoria del velivolo in tempo reale, con coordinate x, y e z (altitudine) calcolate in base alla velocità e alla direzione. La traiettoria è aggiornata a ogni iterazione utilizzando `plot3`, con limiti degli assi adattati dinamicamente per riflettere il percorso.

4. Logging dei dati: I parametri di volo (tempo, altitudine, velocità, tasso di salita, beccheggio, direzione, coordinate x e y) sono salvati in un array durante la simulazione e, al termine, esportati in una tabella MATLAB (`table`) denominata `flightLog`, accessibile dall'ambiente di lavoro per ulteriori analisi.

5. Analisi post-simulazione: Al termine della simulazione, una figura separata con cinque grafici 3D mostra l'evoluzione dei parametri principali (altitudine, beccheggio, tasso di salita, direzione, velocità) in funzione delle coordinate x e y, consentendo un'analisi dettagliata del volo.

Le funzioni annidate, come `scenarioLogic`, implementano i modelli fisici per gli scenari automatici. Ad esempio, nello scenario "Standard", l'altitudine è calcolata integrando il tasso di salita:

$$h(t) = h(t - \Delta t) + \mathrm{roc}(t) \cdot \Delta t \tag{4}$$

dove $\mathrm{roc}(t)$ varia in base alla fase del volo. La posizione in 2D (x, y) è aggiornata utilizzando:

$$x(t) = x(t - \Delta t) + v(t) \cdot \cos(\theta(t)) \cdot \Delta t, \quad y(t) = y(t - \Delta t) + v(t) \cdot \sin(\theta(t)) \cdot \Delta t \tag{5}$$

dove $v(t)$ è la velocità e $\theta(t)$ è la direzione (heading). Questi modelli, pur semplificati, sono adeguati per scopi didattici e possono essere estesi con equazioni più complesse, come la formula barometrica per l'altitudine:

$$P = P_0 \cdot \exp\left(-\frac{g \cdot M \cdot h}{R \cdot T}\right) \tag{6}$$

L'interfaccia grafica è un elemento chiave del simulatore. I gauge circolari, implementati con `uigauge`, offrono una rappresentazione visiva realistica degli strumenti, aggiornati a ogni passo temporale (0.1 s). La sincronizzazione tra simulazione e visualizzazione è garantita dal comando `pause(dt)`, che introduce un ritardo per simulare un'esperienza in tempo reale.

Per ottimizzare le prestazioni, il codice utilizza il comando `drawnow limitrate` per limitare la frequenza di aggiornamento grafico, riducendo il carico computazionale senza compromettere la fluidità della visualizzazione. Tuttavia, su hardware meno potente, un passo temporale più ampio (es. 0.2 s) potrebbe migliorare ulteriormente le prestazioni.

3.3 Applicazioni del simulatore

Il simulatore offre molteplici applicazioni, sia in ambito didattico che professionale, grazie alla sua versatilità e alle sue funzionalità avanzate. La modalità "Standard" consente di osservare un volo realistico con fasi di decollo, crociera e discesa, mentre la modalità "Turbolenza" introduce variazioni casuali che simulano condizioni operative più complesse. La modalità "Manuale" permette di sperimentare il controllo diretto del velivolo, affinando la loro comprensione delle interazioni tra input e output degli strumenti.

In ambito professionale, il simulatore può essere utilizzato come piattaforma di test per algoritmi di controllo o per analizzare il comportamento degli strumenti in scenari realistici. Ad esempio, la modalità "Turbolenza" potrebbe essere estesa per includere modelli più dettagliati di turbolenza atmosferica, utilizzando dati reali per calibrare le variazioni casuali. L'aggiunta di un pilota automatico, basato su tecniche di controllo come il PID, potrebbe migliorare la stabilità in presenza di perturbazioni, rendendo il simulatore utile per lo sviluppo di sistemi avionici.

Il logging dei dati in `flightLog` facilita l'analisi post-simulazione, permettendo di generare report dettagliati o di confrontare i risultati con dati reali. Questo aspetto è

particolarmente utile per la validazione di strumenti avionici o per la formazione di tecnici, che possono esercitarsi nella lettura degli strumenti e nella diagnosi di anomalie. Inoltre, i grafici 3D post-simulazione offrono un'analisi visiva approfondita, utile per identificare tendenze o anomalie nel comportamento del velivolo.

Un'applicazione potenziale è l'integrazione con simulatori di volo più avanzati, come X-Plane, che supportano l'interfacciamento con MATLAB tramite protocolli come UDP. Questo consentirebbe di testare gli strumenti in un ambiente virtuale più complesso, con dinamiche di volo realistiche e interazioni con il pilota. Dal punto di vista della ricerca, il simulatore potrebbe essere utilizzato per esplorare nuove tecnologie avioniche, come sensori MEMS o display head-up, modificando i modelli fisici per includere effetti aerodinamici o variazioni climatiche.

4 Simulazione e Report Tecnico

Questo capitolo rappresenta il culmine del progetto, presentando una descrizione dettagliata della simulazione animata della strumentazione di bordo durante il volo, dell'interfaccia grafica interattiva (GUI) che riproduce un report tecnico che analizza i modelli fisici, l'architettura del codice, gli scenari di volo simulati e le possibili estensioni o criticità. Il simulatore, sviluppato in ambiente MATLAB, è stato progettato per modellare e monitorare strumenti avionici come l'altimetro, il velocimetro, il variometro, la bussola magnetica e l'indicatore di assetto (pitch), offrendo uno strumento didattico e analitico per esplorare le dinamiche di volo. Attraverso l'integrazione di animazioni in tempo reale, un'interfaccia grafica intuitiva e un'analisi tecnica approfondita.

4.1 Simulazione animata della strumentazione in volo

La simulazione animata rappresenta uno degli elementi più significativi del progetto, consentendo di visualizzare in tempo reale il comportamento della strumentazione di bordo durante scenari di volo dinamici. L'animazione è stata implementata utilizzando un ciclo di aggiornamento che sincronizza i dati degli strumenti con un passo temporale di 0.1 s, garantendo una rappresentazione fluida e realistica delle variazioni dei parametri di volo, come altitudine, velocità, tasso di salita/discesa, direzione e assetto (pitch). Questo approccio riflette le dinamiche tipiche di un velivolo leggero, con particolare attenzione agli scenari "Manuale", "Standard" e "Turbolenza".

Il cuore dell'animazione è un ciclo che aggiorna i valori degli strumenti a ogni iterazione, utilizzando modelli fisici semplificati. Ad esempio, nella modalità "Manuale", l'utente controlla il velivolo tramite input da tastiera, con gli strumenti che rispondono in tempo reale: l'altimetro mostra un'altitudine iniziale di 0 m, il velocimetro indica una velocità di $0\,\mathrm{m\,s^{-1}}$, e il pitch è regolabile con le frecce. Nella modalità "Standard", il volo segue un profilo realistico: decollo con un tasso di salita da $2\,\mathrm{m\,s^{-1}}$ a $2.8\,\mathrm{m\,s^{-1}}$ (raggiungendo 24 m in 10 secondi), crociera stabile a 24 m con una virata di 10°, e discesa controllata a 0 m con un tasso di discesa da $-2\,\mathrm{m\,s^{-1}}$ a $-2.8\,\mathrm{m\,s^{-1}}$. L'aggiornamento grafico è sincronizzato con `pause(0.1)`, simulando un'esperienza in tempo reale.

La Figura illustra un frame della modalità "Manuale" durante l'esecuzione, con gli strumenti inizializzati e la traiettoria 3D che inizia a formarsi. La traiettoria 3D in tempo reale, visibile a destra, mostra l'evoluzione dell'altitudine in funzione delle coordinate x e y, con un percorso iniziale piatto che riflette l'assenza di input manuali. Questo frame evidenzia la reattività della simulazione agli input dell'utente, che può modificare velocità e assetto a piacere fino alla pressione del pulsante "Stop".

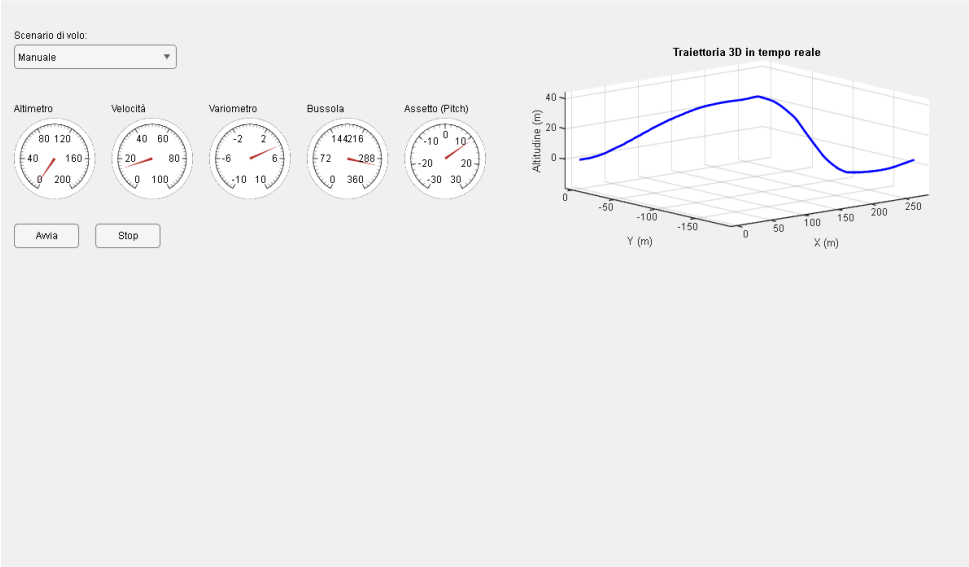

Figura 5: Frame dell'animazione nella modalità "Manuale": gli strumenti mostrano valori iniziali (altitudine 0 m, velocità 0 m s^{-1}, pitch 0°), con la traiettoria 3D che inizia a evolversi.

4.2 Report tecnico

Il report tecnico fornisce un'analisi dettagliata degli aspetti chiave del simulatore, includendo i modelli fisici utilizzati, l'architettura del codice, gli scenari di volo simulati e una discussione su possibili estensioni e criticità. Questa sezione offre una panoramica completa del progetto, evidenziando sia i risultati ottenuti che le aree di miglioramento.

4.2.1 Modelli fisici usati per simulare ogni strumento

La simulazione si basa su modelli fisici semplificati, progettati per scopi didattici. Di seguito, i modelli per ciascun strumento:

-Altimetro: L'altitudine $h(t)$ è calcolata come $h(t) = h(t-\Delta t)+\text{roc}(t)\cdot\Delta t$, dove roc$(t)$ è il tasso di salita/discesa. Nella modalità "Standard", roc varia da 2 m s^{-1} a 2.8 m s^{-1} in decollo, è nullo in crociera, e da -2 m s^{-1} a -2.8 m s^{-1} in discesa.

-Velocimetro: La velocità $v(t)$ segue $v(t) = \min(80, 2\cdot t)$ nella modalità "Standard", con un'accelerazione di 2 m s^{-2}, raggiungendo 60 m s^{-1} in 30 secondi.

-Variometro: Il tasso di salita/discesa roc(t) è determinato da `scenarioLogic`, con variazioni sinusoidali nella modalità "Turbolenza" (roc $= 2\sin(0.5t) + 0.8\cdot$ randn).

-Bussola magnetica: La direzione $\theta(t)$ si aggiorna con $\theta(t) = \theta(t-\Delta t) + \Delta\text{hdg}\cdot\Delta t$, dove Δhdg $= 1\,^\circ$ s^{-1} in crociera "Standard".

-Assetto (Pitch): Il pitch varia sinusoidalmente nella modalità "Standard" (es. pitch $= 5\sin(\pi/2\cdot t/10)$ in decollo), con un massimo di 5°.

Questi modelli sono semplificati ma adeguati per la simulazione, con potenziale per includere la formula barometrica o equazioni aerodinamiche in future estensioni.

4.2.2 Architettura del codice

Il codice è strutturato attorno a un singolo script main.m con un'interfaccia GUI sviluppata tramite uifigure e uigauge. Le principali sezioni sono:

- Inizializzazione: creazione interfaccia e strumenti
- Controlli utente: gestione input da tastiera per lo scenario "Manuale"
- Simulazione: tre modalità operative, ognuna con logica dedicata
- Visualizzazione: grafico 3D live della traiettoria
- Logging: salvataggio dati in una tabella flightLog
- Analisi finale: generazione automatica di grafici 3D dei parametri

4.2.3 Scenari di volo simulato

-Standard (0–30 s): Decollo (0–10 s, 24 m), crociera (10–20 s, virata 10°), discesa (20–30 s, 0 m).

La Figura mostra l'analisi post-simulazione per "Standard", con altitudine che raggiunge 24 m e pitch oscillante.

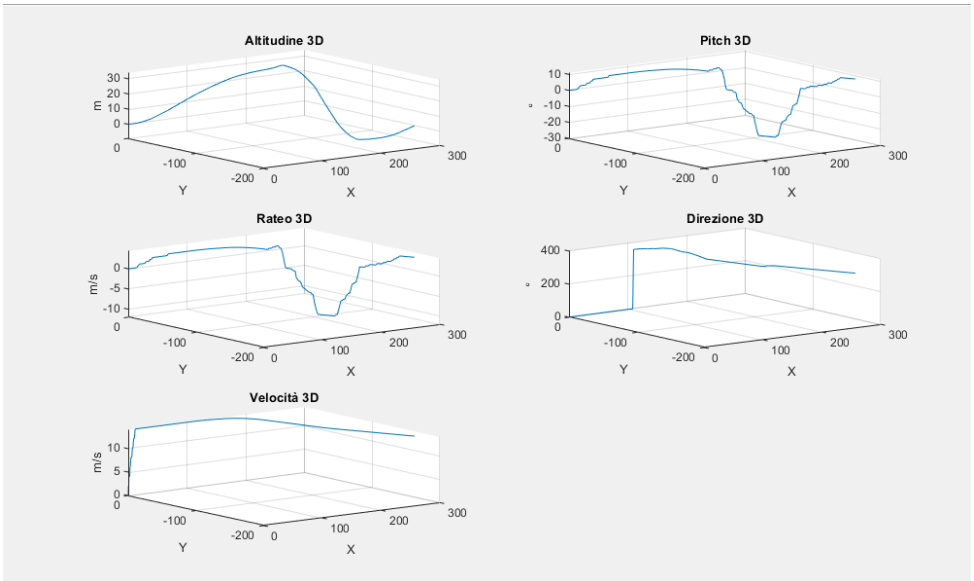

Figura 6: Analisi post-simulazione 3D per lo scenario "Standard": grafici di altitudine, pitch, tasso di salita, direzione e velocità in funzione di x e y.

Durante la simulazione i parametri (tempo, altitudine, velocità, pitch, heading, ROC, x, y) vengono memorizzati in flightLog, una tabella .csv.

Colonna	Descrizione
`Time`	Tempo trascorso dalla partenza (in secondi)
`Altitude`	Altitudine del velivolo (in metri o unità arbitrarie)
`Speed`	Velocità istantanea (m/s o unità definite nello scenario)
`ROC`	Rate of Climb: tasso di salita/discesa (positivo = salita)
`Pitch`	Inclinazione longitudinale (naso su/giù) espressa in radianti
`Heading`	Intestazione (direzione della bussola), in gradi o radianti
`X`	Coordinata X orizzontale del volo (movimento longitudinale)
`Y`	Coordinata Y orizzontale (spostamento laterale)

Tabella 1: Significato delle colonne del file `flightLog.csv` generato dalla simulazione.

5 Conclusioni

Il progetto di sviluppo di un simulatore avionico, attraverso l'implementazione di un'interfaccia grafica interattiva (GUI) che riproduce una simulazione animata in tempo reale e un'analisi tecnica approfondita, il simulatore ha raggiunto gli obiettivi prefissati, offrendo uno strumento didattico e analitico per esplorare le dinamiche di volo e il funzionamento degli strumenti avionici. Questo progetto dimostra l'efficacia di MATLAB come piattaforma versatile per la simulazione, integrando modellazione fisica, programmazione strutturata e visualizzazione grafica.

L'ottimizzazione del codice, con un passo temporale di 0.1 s e l'uso di `drawnow limitrate`, garantisce una simulazione fluida e performante, salvando i dati in una tabella `flightLog` per ulteriori studi.

Il simulatore si è rivelato coerente con i risultati attesi, permettendo di comprendere le interazioni tra input di controllo e parametri di volo, e come piattaforma di sperimentazione, grazie alla sua modularità e scalabilità. Questo progetto non solo consolida le competenze acquisite durante il corso, ma getta le basi per ulteriori sviluppi, contribuendo al campo dell'avionica con un approccio pratico e innovativo.

6 Appendice A

6.1 Codice MATLAB

Di seguito si riporta il codice completo.

```
% Main Script: Simulazione e Monitoraggio Strumentazione
% Autore: Angelo Mastrangelo, Matricola 587675

clc; clear; close all;

function main()


    % === Inizializzazione GUI ===
    fig = uifigure('Name','Cockpit Simulatore','Position',[100 100 1200 700]);
    % Crea una finestra grafica (uifigure) intitolata "Cockpit Simulatore" con dimensioni 1200x700 pixel, posizionata a (100,100) sullo schermo.

    ax = uiaxes(fig, 'Position', [650 400 500 250]);
    % Crea un'area per grafici 3D (uiaxes) nella finestra, posizionata a (650,400) con dimensioni 500x250 pixel.

    % Dropdown scenario
    uilabel(fig,'Position',[20 650 150 22],'Text','Scenario di volo:');
    % Aggiunge un'etichetta testuale "Scenario di volo:" a (20,650) con dimensioni 150x22 pixel.

    scenarioDrop = uidropdown(fig, ...
        'Items',{'Manuale','Standard','Turbolenza'}, ...
        'Position',[20 620 200 30]);
    % Crea un menu a tendina (dropdown) con opzioni "Manuale", "Standard", "Turbolenza", posizionato a (20,620) con dimensioni 200x30 pixel.

    % === Strumenti ===
    uilabel(fig,'Text','Altimetro','Position',[20 560 100 22]);
    % Aggiunge un'etichetta "Altimetro" a (20,560) con dimensioni 100x22 pixel.

    altGauge = uigauge(fig,'circular','Position',[20 460 100 100],'Limits',[0 200]);
    % Crea un indicatore circolare (gauge) per l'altimetro a (20,460), dimensioni 100x100 pixel, con limiti 0-200 metri.

    uilabel(fig,'Text','Velocit ','Position',[140 560 100 22]);
    % Aggiunge un'etichetta "Velocit " a (140,560) con dimensioni 100x22 pixel.

```

```
35      spdGauge = uigauge(fig,'circular','Position',[140 460 100
            100],'Limits',[0 100]);
36      % Crea un indicatore circolare per la velocit  a (140,460),
            dimensioni 100x100 pixel, con limiti 0-100 m/s.
37
38      uilabel(fig,'Text','Variometro','Position',[260 560 100 22]);
39      % Aggiunge un'etichetta "Variometro" a (260,560) con
            dimensioni 100x22 pixel.
40
41      varioGauge = uigauge(fig,'circular','Position',[260 460 100
            100],'Limits',[-10 10]);
42      % Crea un indicatore circolare per il variometro a (260,460),
             dimensioni 100x100 pixel, con limiti -10 a 10 m/s.
43
44      uilabel(fig,'Text','Bussola','Position',[380 560 100 22]);
45      % Aggiunge un'etichetta "Bussola" a (380,560) con dimensioni
            100x22 pixel.
46
47      compassGauge = uigauge(fig,'circular','Position',[380 460 100
             100],'Limits',[0 360]);
48      % Crea un indicatore circolare per la bussola a (380,460),
            dimensioni 100x100 pixel, con limiti 0-360 gradi.
49
50      uilabel(fig,'Text','Assetto␣(Pitch)','Position',[500 560 100
            22]);
51      % Aggiunge un'etichetta "Assetto (Pitch)" a (500,560) con
            dimensioni 100x22 pixel.
52
53      pitchGauge = uigauge(fig,'circular','Position',[500 460 100
            100],'Limits',[-30 30]);
54      % Crea un indicatore circolare per l'assetto (pitch) a
            (500,460), dimensioni 100x100 pixel, con limiti -30 a 30
            gradi.
55
56      % Bottoni
57      startBtn = uibutton(fig,'Text','Avvia','Position',[20 400 80
            30]);
58      % Crea un pulsante "Avvia" a (20,400) con dimensioni 80x30
            pixel.
59
60      stopBtn = uibutton(fig,'Text','Stop','Position',[120 400 80
            30]);
61      % Crea un pulsante "Stop" a (120,400) con dimensioni 80x30
            pixel.
62
63      % Stato iniziale
64      pitchOffset = 0;
65      % Inizializza la variabile pitchOffset a 0 (usata per
            controllare l'assetto in modalit  manuale).
66
67      heading = 0;
```

```
68      % Inizializza la direzione (heading) a 0 gradi.
69
70      yawRate = 0;
71      % Inizializza la velocit  di rotazione (yaw rate) a 0 gradi/
            s.
72
73      speed = 0;
74      % Inizializza la velocit  a 0 m/s.
75
76      isRunning = false;
77      % Inizializza lo stato della simulazione a false (non in
            esecuzione).
78
79      % Controllo tastiera
80      fig.KeyPressFcn = @(~,event) handleKeys(event);
81      % Assegna la funzione handleKeys come callback per gli eventi
            di pressione dei tasti sulla finestra.
82
83      function handleKeys(event)
84          % Definisce la funzione per gestire gli input da tastiera
                .
85
86          if strcmp(scenarioDrop.Value, 'Manuale')
87              % Controlla se lo scenario selezionato    "Manuale".
88
89              switch event.Key
90                  % Inizia uno switch per gestire diversi tasti
                        premuti.
91
92                  case 'uparrow'
93                      % Se premuta la freccia su:
94                      pitchOffset = pitchOffset + 1;
95                      % Incrementa l'assetto (pitchOffset) di 1
                            grado.
96
97                  case 'downarrow'
98                      % Se premuta la freccia gi  :
99                      pitchOffset = pitchOffset - 1;
100                     % Decrementa l'assetto (pitchOffset) di 1
                            grado.
101
102                 case 'leftarrow'
103                     % Se premuta la freccia sinistra:
104                     yawRate = -20;
105                     % Imposta la velocit  di rotazione (yawRate)
                            a -20 gradi/s (virata a sinistra).
106
107                 case 'rightarrow'
108                     % Se premuta la freccia destra:
109                     yawRate = 20;
110                     % Imposta la velocit  di rotazione (yawRate)
```

```
                    a 20 gradi/s (virata a destra).
111
112             case {'add', 'equal'}
113                 % Se premuto il tasto "+" o "=":
114                 speed = min(100, speed + 2);
115                 % Aumenta la velocit  di 2 m/s, senza
                        superare il limite di 100 m/s.
116
117             case 'subtract'
118                 % Se premuto il tasto "-":
119                 speed = max(0, speed - 2);
120                 % Diminuisce la velocit  di 2 m/s, senza
                        scendere sotto 0 m/s.
121
122             case 'm'
123                 % Se premuto il tasto "m":
124                 speed = min(100, speed + 2);
125                 % Aumenta la velocit  di 2 m/s, senza
                        superare il limite di 100 m/s.
126
127             case 'n'
128                 % Se premuto il tasto "n":
129                 speed = max(0, speed - 2);
130                 % Diminuisce la velocit  di 2 m/s, senza
                        scendere sotto 0 m/s.
131         end
132     end
133 end
134
135 % Callback bottoni
136 startBtn.ButtonPushedFcn = @(~,~) simulate();
137 % Assegna la funzione simulate come callback per il pulsante
        "Avvia".
138
139 stopBtn.ButtonPushedFcn = @(~,~) setappdata(fig,'stop',true);
140 % Assegna una funzione che imposta la propriet  'stop' a
        true quando si preme il pulsante "Stop".
141
142 function simulate()
143     % Definisce la funzione per eseguire la simulazione.
144
145     if isRunning, return; end
146     % Se la simulazione    gi   in corso (isRunning=true),
            esce dalla funzione.
147
148     isRunning = true;
149     % Imposta lo stato della simulazione a in esecuzione.
150
151     setappdata(fig,'stop',false);
152     % Inizializza la propriet  'stop' della figura a false.
153
```

```
154         scenario = scenarioDrop.Value;
155         % Memorizza il valore dello scenario selezionato dal menu
                a tendina.
156
157         dt = 0.1;
158         % Definisce il passo temporale della simulazione (0.1
               secondi).
159
160         t = 0;
161         % Inizializza il tempo della simulazione a 0 secondi.
162
163         % Inizializzazione dati
164         alt = []; spd = []; pitch = []; hdg = []; roc = []; x =
               []; y = [];
165         % Inizializza array vuoti per altitudine, velocit ,
               assetto, direzione, rateo di salita, coordinate x e y.
166
167         time = [];
168         % Inizializza un array vuoto per il tempo.
169
170         heading = 0;
171         % Reimposta la direzione iniziale a 0 gradi.
172
173         speed = strcmp(scenario,'Manuale') * 0;
174         % Imposta la velocit  iniziale a 0 solo per lo scenario
               "Manuale" (altrimenti usa valori dello scenario).
175
176         if strcmp(scenario,'Manuale')
177             % Se lo scenario    "Manuale":
178             uialert(fig, 'Usa␣le␣frecce␣per␣manovrare.␣M/N␣per␣la
                    ␣velocit .␣Premi␣Stop␣per␣terminare.', 'Controlli␣
                    Manuali');
179             % Mostra un messaggio con le istruzioni per i comandi
                    manuali.
180         end
181
182         % Grafico 3D
183         cla(ax); view(ax,3); grid(ax,'on');
184         % Pulisce l'area del grafico (ax), imposta la vista 3D e
               attiva la griglia.
185
186         xlabel(ax,'X␣(m)'); ylabel(ax,'Y␣(m)'); zlabel(ax,'
               Altitudine␣(m)');
187         % Imposta le etichette degli assi del grafico 3D: X, Y e
               Altitudine.
188
189         title(ax,'Traiettoria␣3D␣in␣tempo␣reale');
190         % Imposta il titolo del grafico 3D.
191
192         hold(ax,'on');
193         % Abilita la modalit  di sovrascrittura per aggiungere
```

```
    pi  tracciati al grafico.
194
195 p3 = plot3(ax, NaN, NaN, NaN, 'b', 'LineWidth', 2);
196 % Crea un grafico 3D vuoto con linea blu spessa 2 pixel
    per la traiettoria.
197
198 if strcmp(scenario, 'Manuale')
199     % Se lo scenario    "Manuale":
200     xlim(ax, [-5000 5000]); % Limiti pi  ampi per
        simulazione indefinita
201     % Imposta limiti dell'asse X da -5000 a 5000 metri.
202     ylim(ax, [-5000 5000]);
203     % Imposta limiti dell'asse Y da -5000 a 5000 metri.
204 else
205     % Per scenari "Standard" o "Turbolenza":
206     xlim(ax, [-1000 1000]); % Limiti per 30 secondi
207     % Imposta limiti dell'asse X da -1000 a 1000 metri.
208     ylim(ax, [-1000 1000]);
209     % Imposta limiti dell'asse Y da -1000 a 1000 metri.
210 end
211 zlim(ax, [0 200]); % Altitudine massima definita dal
    gauge
212 % Imposta limiti dell'asse Z (altitudine) da 0 a 200
    metri.
213
214 % Indice per salvare dati
215 i = 1;
216 % Inizializza l'indice per salvare i dati a 1.
217
218 if strcmp(scenario, 'Manuale')
219     % Se lo scenario    "Manuale":
220     % Simulazione indefinita per Manuale
221     while ~getappdata(fig,'stop')
222         % Ciclo che continua finch  la propriet  'stop'
              non    true.
223
224         t = t + dt;
225         % Incrementa il tempo della simulazione di dt
            (0.1 secondi).
226
227         time(i) = t;
228         % Salva il tempo corrente nell'array time.
229
230         roc(i) = pitchOffset * 0.4;
231         % Calcola il rateo di salita (roc) come 0.4 *
            pitchOffset.
232
233         pitch(i) = pitchOffset;
234         % Salva l'assetto corrente (pitchOffset) nell'
            array pitch.
235
```

```
236             heading = heading + yawRate * dt;
237             % Aggiorna la direzione (heading) aggiungendo
                    yawRate * dt.
238
239             spd(i) = speed;
240             % Salva la velocit  corrente nell'array spd.
241
242             yawRate = 0; % resetta ogni ciclo
243             % Reimposta yawRate a 0 per il ciclo successivo.
244
245             heading = mod(heading, 360);
246             % Normalizza la direzione tra 0 e 360 gradi.
247
248             hdg(i) = heading;
249             % Salva la direzione corrente nell'array hdg.
250
251             if i == 1
252                 % Se    il primo ciclo:
253                 alt(i) = 0; x(i) = 0; y(i) = 0;
254                 % Inizializza altitudine e coordinate x, y a
                        0.
255             else
256                 % Per i cicli successivi:
257                 alt(i) = alt(i-1) + roc(i)*dt;
258                 % Calcola la nuova altitudine sommando il
                        rateo di salita * dt.
259
260                 dx = cosd(heading) * spd(i) * dt;
261                 % Calcola lo spostamento in x usando la
                        direzione (in gradi) e la velocit .
262
263                 dy = sind(heading) * spd(i) * dt;
264                 % Calcola lo spostamento in y usando la
                        direzione (in gradi) e la velocit .
265
266                 x(i) = x(i-1) + dx;
267                 % Aggiorna la coordinata x sommando lo
                        spostamento dx.
268
269                 y(i) = y(i-1) + dy;
270                 % Aggiorna la coordinata y sommando lo
                        spostamento dy.
271             end
272
273             % Aggiorna strumenti
274             altGauge.Value = alt(i);
275             % Aggiorna il gauge dell'altimetro con l'
                    altitudine corrente.
276
277             spdGauge.Value = spd(i);
278             % Aggiorna il gauge della velocit  con la
```

```
    velocit  corrente.
279
280             varioGauge.Value = roc(i);
281             % Aggiorna il gauge del variometro con il rateo
                    di salita corrente.
282
283             compassGauge.Value = heading;
284             % Aggiorna il gauge della bussola con la
                    direzione corrente.
285
286             pitchGauge.Value = pitch(i);
287             % Aggiorna il gauge dell'assetto con il pitch
                    corrente.
288
289             % Grafico live 3D
290             if isvalid(p3)
291                 % Se il grafico 3D    valido:
292                 set(p3, 'XData', x(1:i), 'YData', y(1:i), '
                        ZData', alt(1:i));
293                 % Aggiorna i dati del grafico 3D con le
                        coordinate x, y e altitudine fino al punto
                        corrente.
294
295                 if i > 1
296                     % Se ci sono almeno due punti:
297                     xlim(ax, [min(x(1:i))-10, max(x(1:i))
                            +10]);
298                     % Aggiorna i limiti dell'asse X in base
                            ai valori di x, con un margine di 10
                            metri.
299
300                     ylim(ax, [min(y(1:i))-10, max(y(1:i))
                            +10]);
301                     % Aggiorna i limiti dell'asse Y in base
                            ai valori di y, con un margine di 10
                            metri.
302
303                     zlim(ax, [min(alt(1:i))-10, max(alt(1:i))
                            +10]);
304                     % Aggiorna i limiti dell'asse Z in base
                            ai valori di altitudine, con un margine
                             di 10 metri.
305                 end
306                 drawnow limitrate;
307                 % Aggiorna il grafico in tempo reale,
                        limitando il tasso di aggiornamento per
                        migliorare le prestazioni.
308             end
309             pause(dt);
310             % Pausa di dt secondi per simulare il tempo reale
                .
```

```
311
312                 i = i + 1;
313                 % Incrementa l'indice per il prossimo ciclo.
314             end
315         else
316             % Per scenari "Standard" o "Turbolenza":
317             simTime = 30;
318             % Imposta la durata della simulazione a 30 secondi.
319
320             time = 0:dt:simTime;
321             % Crea un vettore di tempi da 0 a 30 secondi con
                   passo dt.
322
323             N = length(time);
324             % Calcola il numero di iterazioni basato sul vettore
                   time.
325
326             for i = 1:N
327                 % Ciclo per ogni istante di tempo:
328                 if getappdata(fig,'stop'), break; end
329                 % Interrompe il ciclo se la propriet  'stop'
                       true.
330
331                 t = time(i);
332                 % Imposta il tempo corrente al valore di time(i).
333
334                 [roc(i), pitch(i), deltaHdg] = scenarioLogic(t,
                       scenario);
335                 % Chiama la funzione scenarioLogic per ottenere
                       rateo di salita, pitch e variazione di
                       direzione.
336
337                 heading = heading + deltaHdg * dt;
338                 % Aggiorna la direzione aggiungendo la variazione
                        di direzione * dt.
339
340                 if strcmp(scenario, 'Standard')
341                     % Se lo scenario    "Standard":
342                     spd(i) = min(80, 2 * t); % Accelerazione a 2
                           m/s^2
343                     % Imposta la velocit  come 2*t, limitata a
                           80 m/s.
344                 else
345                     % Se lo scenario    "Turbolenza":
346                     spd(i) = min(80, i * 0.8); % Per Turbolenza
347                     % Imposta la velocit  come 0.8*i, limitata a
                            80 m/s.
348                 end
349
350                 heading = mod(heading, 360);
351                 % Normalizza la direzione tra 0 e 360 gradi.
```

```
352
353             hdg(i) = heading;
354             % Salva la direzione corrente nell'array hdg.
355
356             if i == 1
357                 % Se    il primo ciclo:
358                 alt(i) = 0; x(i) = 0; y(i) = 0;
359                 % Inizializza altitudine e coordinate x, y a
                        0.
360             else
361                 % Per i cicli successivi:
362                 alt(i) = alt(i-1) + roc(i)*dt;
363                 % Calcola la nuova altitudine sommando il
                        rateo di salita * dt.
364
365                 dx = cosd(heading) * spd(i) * dt;
366                 % Calcola lo spostamento in x usando la
                        direzione e la velocit .
367
368                 dy = sind(heading) * spd(i) * dt;
369                 % Calcola lo spostamento in y usando la
                        direzione e la velocit .
370
371                 x(i) = x(i-1) + dx;
372                 % Aggiorna la coordinata x sommando lo
                        spostamento dx.
373
374                 y(i) = y(i-1) + dy;
375                 % Aggiorna la coordinata y sommando lo
                        spostamento dy.
376             end
377
378             % Aggiorna strumenti
379             altGauge.Value = alt(i);
380             % Aggiorna il gauge dell'altimetro con l'
                    altitudine corrente.
381
382             spdGauge.Value = spd(i);
383             % Aggiorna il gauge della velocit  con la
                    velocit  corrente.
384
385             varioGauge.Value = roc(i);
386             % Aggiorna il gauge del variometro con il rateo
                    di salita corrente.
387
388             compassGauge.Value = heading;
389             % Aggiorna il gauge della bussola con la
                    direzione corrente.
390
391             pitchGauge.Value = pitch(i);
392             % Aggiorna il gauge dell'assetto con il pitch
```

```
                corrente.
393
394             % Grafico live 3D
395             if isvalid(p3)
396                 % Se il grafico 3D    valido:
397                 set(p3, 'XData', x(1:i), 'YData', y(1:i), '
                        ZData', alt(1:i));
398                 % Aggiorna i dati del grafico 3D con le
                        coordinate x, y e altitudine fino al punto
                        corrente.
399
400                 if i > 1
401                     % Se ci sono almeno due punti:
402                     xlim(ax, [min(x(1:i))-10, max(x(1:i))
                            +10]);
403                     % Aggiorna i limiti dell'asse X in base
                            ai valori di x, con margine di 10 metri
                            .
404
405                     ylim(ax, [min(y(1:i))-10, max(y(1:i))
                            +10]);
406                     % Aggiorna i limiti dell'asse Y in base
                            ai valori di y, con margine di 10 metri
                            .
407
408                     zlim(ax, [min(alt(1:i))-10, max(alt(1:i))
                            +10]);
409                     % Aggiorna i limiti dell'asse Z in base
                            ai valori di altitudine, con margine di
                             10 metri.
410                 end
411                 drawnow limitrate;
412                 % Aggiorna il grafico in tempo reale,
                        limitando il tasso di aggiornamento.
413             end
414             pause(dt);
415             % Pausa di dt secondi per simulare il tempo reale
                    .
416         end
417     end
418
419     isRunning = false;
420     % Imposta lo stato della simulazione a non in esecuzione
            al termine.
421
422     % Log finale
423     flightLog = table(time', alt', spd', roc', pitch', hdg',
            x', y', ...
424         'VariableNames', {'Time','Altitude','Speed','ROC','
                Pitch','Heading','X','Y'});
425     % Crea una tabella con i dati della simulazione (tempo,
```

```
    altitudine, velocit , rateo di salita, pitch,
    direzione, x, y).

assignin('base','flightLog',flightLog);
% Esporta la tabella flightLog nell'ambiente base di
    MATLAB.

uialert(fig,'Simulazione completata. Dati salvati in
    flightLog.','Fine');
% Mostra un messaggio di conferma al termine della
    simulazione.

% Analisi 3D
figure('Name','Analisi Post-Flight 3D','Position',[200
    100 1000 600]);
% Crea una nuova finestra per l'analisi post-volo,
    dimensioni 1000x600 pixel, posizionata a (200,100).

tiledlayout(3,2);
% Crea un layout a griglia 3x2 per i grafici di analisi.

nexttile; plot3(flightLog.X, flightLog.Y, flightLog.
    Altitude); grid on;
% Crea un grafico 3D della traiettoria (X, Y, Altitudine)
     nella prima casella.

title('Altitudine 3D'); xlabel('X'); ylabel('Y'); zlabel(
    'm');
% Imposta titolo ed etichette per il grafico dell'
    altitudine.

nexttile; plot3(flightLog.X, flightLog.Y, flightLog.Pitch
    ); grid on;
% Crea un grafico 3D di X, Y e Pitch nella seconda
    casella.

title('Pitch 3D'); xlabel('X'); ylabel('Y'); zlabel('  ')
    ;
% Imposta titolo ed etichette per il grafico del pitch.

nexttile; plot3(flightLog.X, flightLog.Y, flightLog.ROC);
     grid on;
% Crea un grafico 3D di X, Y e rateo di salita nella
    terza casella.

title('Rateo 3D'); xlabel('X'); ylabel('Y'); zlabel('m/s'
    );
% Imposta titolo ed etichette per il grafico del rateo di
     salita.

nexttile; plot3(flightLog.X, flightLog.Y, flightLog.
```

```
            Heading); grid on;
459         % Crea un grafico 3D di X, Y e direzione nella quarta
                casella.
460
461         title('Direzione 3D'); xlabel('X'); ylabel('Y'); zlabel('
                ');
462         % Imposta titolo ed etichette per il grafico della
                direzione.
463
464         nexttile; plot3(flightLog.X, flightLog.Y, flightLog.Speed
                ); grid on;
465         % Crea un grafico 3D di X, Y e velocit  nella quinta
                casella.
466
467         title('Velocit  3D'); xlabel('X'); ylabel('Y'); zlabel('
                m/s');
468         % Imposta titolo ed etichette per il grafico della
                velocit .
469     end
470
471     % Logica scenario automatico
472     function [roc, pitch, deltaHdg] = scenarioLogic(t, type)
473         % Definisce la funzione per la logica degli scenari
                automatici ("Standard" e "Turbolenza").
474
475         switch type
476             % Inizia uno switch basato sul tipo di scenario.
477
478             case 'Standard'
479                 % Per lo scenario "Standard":
480                 if t < 10 % Decollo graduale
481                     % Per i primi 10 secondi (fase di decollo):
482                     roc = 2 + 0.8 * (t/10); % Da 2 a 2.8 m/s
483                     % Calcola il rateo di salita che aumenta
                            linearmente da 2 a 2.8 m/s.
484
485                     pitch = 5 * sin((pi/2) * (t/10)); % Da 0   a
                            5   e torna a 0
486                     % Calcola il pitch che varia sinusoidalmente
                            da 0   a 5   e torna a 0  .
487
488                     deltaHdg = 0; % Nessuna virata
489                     % Imposta la variazione di direzione a 0 (
                            nessuna virata).
490
491                 elseif t < 20 % Crociera stabile
492                     % Per i successivi 10 secondi (fase di
                            crociera):
493                     roc = 0; % Altitudine stabile
494                     % Imposta il rateo di salita a 0 (altitudine
                            costante).
```

```
495
496                     pitch = 0; % Assetto livellato
497                     % Imposta il pitch a 0 (assetto livellato).
498
499                     deltaHdg = 1; % Gira lentamente di 10   in
                           totale
500                     % Imposta una variazione di direzione di 1
                           grado/s.
501
502                 else % Discesa controllata
503                     % Per gli ultimi 10 secondi (fase di discesa)
                           :
504                     roc = -2 - 0.8 * ((t-20)/10); % Da -2 a -2.8
                           m/s
505                     % Calcola il rateo di discesa che aumenta
                           linearmente da -2 a -2.8 m/s.
506
507                     pitch = -5 * sin((pi/2) * ((t-20)/10)); % Da
                           0   a -5   e torna a 0
508                     % Calcola il pitch che varia sinusoidalmente
                           da 0   a -5   e torna a 0  .
509
510                     deltaHdg = 0; % Nessuna virata
511                     % Imposta la variazione di direzione a 0 (
                           nessuna virata).
512                 end
513             case 'Turbolenza'
514                 % Per lo scenario "Turbolenza":
515                 roc = 2*sin(0.5*t) + 0.8*randn();
516                 % Calcola un rateo di salita variabile con una
                       componente sinusoidale e rumore casuale.
517
518                 pitch = 10*sin(0.4*t);
519                 % Calcola il pitch che varia sinusoidalmente con
                       ampiezza 10  .
520
521                 deltaHdg = 5*sin(0.2*t);
522                 % Calcola la variazione di direzione sinusoidale
                       con ampiezza 5  /s.
523
524             otherwise
525                 % Per qualsiasi altro caso:
526                 roc = 0; pitch = 0; deltaHdg = 0;
527                 % Imposta rateo di salita, pitch e variazione di
                       direzione a 0.
528         end
529     end
530 end
```

7 Appendice B

Di seguito sono riportati gli argomenti trattati nel corso per completezza.

Principi di programmazione

Teorema di Bohm-Jacopini: ogni programma può essere costruito utilizzando tre strutture fondamentali:

- Sequenza
- Selezione (if, switch)
- Iterazione (loop)

Diagrammi a blocchi

Ogni disciplina ingegneristica ha un proprio linguaggio grafico (es. diagrammi a blocchi). Per la programmazione, si utilizzano i diagrammi di flusso (flowchart). Un algoritmo inizia sempre con un blocco di start e termina con un blocco di end. Non ci devono essere rami che terminano nel vuoto. Il concetto di suddivisione dei problemi in blocchi è stato introdotto da Mūsā al-Khwārizmī, da cui deriva il termine algoritmo. Un algoritmo è una sequenza di istruzioni elementari da fare eseguire a un calcolatore.

Elementi base di MATLAB

L'estensione `.m` indica un file di MATLAB. MATLAB gestisce tutti gli elementi tramite gli indici di una matrice (sia vettori che matrici). Anche un singolo scalare è considerato una matrice. Il nome MATLAB deriva da Matrix Laboratory. MATLAB è un ambiente di sviluppo, non un semplice programma e presenta la seguente struttura:

- Sezione cartelle: organizza i file di progetto
- Command Window: esecuzione diretta di comandi
- Editor: scrittura e salvataggio di script `.m` (non visibile all'avvio)

Regole di nomenclatura dei file

I file MATLAB non possono:

- contenere spazi nei nomi (utilizzare underscore o CamelCase)
- iniziare con caratteri speciali o numeri

MATLAB è case-sensitive (`script01.m` è diverso da `Script01.m`).

- `clc`: svuota la Command Window
- `clear`: svuota il Workspace

Il ";" a fine riga evita la visualizzazione dell'output in Command Window, ma salva il risultato nel Workspace. La Command Window può essere utilizzata come taccuino per testare parti di codice prima di inserirle nello script (è persistente).

I commenti si indicano con il simbolo `%`.

Oltre a creare con new script i file `.m`, si può usare Function o Class. Il simbolo `>>` indica che sto operando sulla riga di comando, mentre nell'editor non ho questo simbolo; quindi, quando salvo questi comandi tutti insieme sto operando come da principio della programmazione (insieme di istruzioni di sequenza, selezione e iterazione).

Variabili

Per creare una variabile, si assegna un valore a un nome valido.

```
>> x = 5;
```

La variabile `x` appare nel Workspace e può essere richiamata successivamente. Non viene stampato il risultato perché l'istruzione termina con il ";".

Vettori

I vettori riga possono essere creati con lo spazio tra gli elementi nelle parentesi quadre o con la virgola tra gli elementi.

$$\begin{aligned} V &= [1\,2\,3]; \\ V &= [1, 2, 3]; \end{aligned} \tag{7}$$

Il vettore colonna si ottiene separando gli elementi con il punto e virgola.

$$V = [1; 2; 3; 4; 5] \tag{8}$$

Per accedere a un elemento di un vettore attraverso il suo indice, racchiudere l'indice tra parentesi tonde dopo il nome del vettore.

$$V(3) \quad \%\text{elemento in terza posizione nel vettore} \tag{9}$$

Matrici

Per accedere a un elemento inserire la coppia di indici tra parentesi tonde, separati da virgola.

$$M(r, c) \tag{10}$$

Esempio:

$$M(2, 3) \quad \%\text{elemento in seconda riga e terza colonna} \tag{11}$$

Posso modificare il valore di un elemento a un determinato indice assegnando un nuovo valore.

Se assegno un elemento a una posizione oltre i limiti di dimensione dichiarati, non dà errore ma si adatta automaticamente.

Esempio:

$M = [1, 2, 3; 4, 5, 6; 7, 8, 9]$ %matrice 3×3
$M(5, 5) = 10$ %amplia la matrice fino ad avere 5 righe e 5 colonne e aggiunge l'elemento (12)

Si può selezionare solo una riga o una colonna utilizzando ":" tra gli indici.
Esempio:

$$\begin{aligned} &M(:, 2) \quad \%\text{prende tutte le righe ma solo la seconda colonna} \\ &M(3, :) \quad \%\text{prende solo la terza riga con tutte le colonne} \end{aligned} \tag{13}$$

Stringhe

Per visualizzare una stringa di testo si possono utilizzare i comandi `fprintf("")` o `disp("")`. `\t` inserisce una tabulazione orizzontale, `\n` crea una nuova riga verticale. `disp("")` va a capo in automatico.

Per eseguire uno script `.m` salvato posso scrivere nella Command Window il nome dello script (senza estensione) e viene eseguito.

Quando richiamo uno script è come se leggessi riga per riga lo script e la eseguissi in sequenza nel Command Window.

Script esterni

Per richiamare uno script esterno `.m` si utilizza il comando `run("")`, passando il nome dello script come parametro.

Tale istruzione si può utilizzare anche come comando, senza parentesi tonde, specificando solo il nome del file con estensione, oppure direttamente indicando il nome dello script senza estensione.

Esempio:

```
run("s02.m")
run s02.m
s02
```

Functions

Per realizzare script generici e che si adattano dinamicamente al contesto di utilizzo, i file script `.m` sono limitati. Si utilizzano i file di tipo function.

La sintassi di base di un file function è: keyword `function` seguita da un vettore degli output dal nome del file e dai parametri in parentesi tonde.

Esempio:

```
function [y1, y2, ..., yn] = nomefun(x1, x2, ..., xn)
function [area] = area_cerchio(raggio)
    % calcola area di un cerchio, il cui raggio    un parametro
       di ingresso
    area = 2 * pi * raggio;
end
```

In caso di output multipli:

```
function [out1, out2] = nomefun(param)
    % Corpo della funzione
end
```

Per richiamare una function, MATLAB cerca il parametro nel Workspace globale se non forniamo il valore numerico come parametro.

Il Workspace di una function è separato da quello di uno script (locale) dove sono contenuti i parametri formali definiti, utilizzati nei calcoli interni della function stessa. Vengono, dunque, evitati conflitti tra le variabili nel Workspace globale degli script e locale delle function.

Quando la function è memorizzata in un percorso diverso rispetto alla cartella di lavoro, il comando da utilizzare è il seguente:

```
addpath(percorso)
```

Input utente

Per prelevare un input dall'utente utilizzare il comando `input`.

Esempio:

```
n = input("Inserire un valore per n: ");
```

L'utilizzo degli script `.m` al posto della Command Window, serve affinché si possa richiamare una sequenza di istruzioni in una sola volta, che possono essere richiamati in un secondo momento. Inoltre, nella Command Window se si commettono errori nel codice diventa tedioso operare nella riga di comando, rispetto a uno script.

Se prima della definizione della funzione nel file function sono presenti dei commenti, eseguendo il comando `help` + nome della funzione, questi vengono stampati a video. Ciò è utile per comprendere il funzionamento della funzione, senza doverla aprire.

Esempio:

- Function:

```
% Calcola area dell'aula
% Input:
% b - base
% h - altezza
% Output:
% area - area dell'aula
% perimetro - perimetro dell'aula
% diagonale - diagonale dell'aula
function [area, perimetro, diagonale] = calcola_geometria_aula(b,
    h)
    area = b * h; % calcola area
    perimetro = (b + h) * 2; % calcola perimetro
    diagonale = sqrt(b^2 + h^2); % calcola diagonale
end
```

- Main:

```
fprintf("Calcola la geometria dell'aula\n"); % messaggio iniziale
b = input("Inserire un valore per la base b: ");
h = input("Inserire un valore per l'altezza h: ");
[area, perimetro, diagonale] = calcola_geometria_aula(b, h); %
   calcola geometrie
fprintf("L'area dell'aula    : %.2f mq. \n", area); % stampa area
fprintf("Il perimetro dell'aula    : %.2f m\n", perimetro); %
   stampa perimetro
fprintf("La diagonale dell'aula    : %.2f m\n", diagonale); %
   stampa diagonale
```

Funzioni locali, annidate ed esterne

Le funzioni locali sono funzioni secondarie che si trovano nello stesso file di uno script e sono utili per organizzare il codice. Non sono visibili all'esterno del file, ma solo dalla funzione principale (script main) o dalle altre funzioni locali presenti nello stesso file. Così facendo, è possibile salvare più funzioni correlate in uno stesso file function, senza dover creare un file separato per ognuna delle funzioni presenti.

Esempio:

- Function:

```
% Calcola la somma e il prodotto di due numeri (funzioni locali)
% input: a, b - operandi delle operazioni
% output: risultato - array che contiene somma e prodotto
% risultato delle operazioni
function risultato = calcola_operazioni(a, b)
    s = somma(a, b); % chiama la funzione locale somma
    p = prodotto(a, b); % chiama la funzione locale prodotto
    risultato = [s, p];
end
% funzione locale somma
function s = somma(a, b)
    s = a + b;
end
% funzione locale prodotto
function p = prodotto(a, b)
    p = a * b;
end
```

- Main:

```
fprintf("Calcola operazioni somma e prodotto\n"); % messaggio
   iniziale
a = input("Inserire un valore per a: ");
b = input("Inserire un valore per b: ");
risultato = calcola_operazioni(a, b); % calcolo
somma = risultato(1);
```

```
prodotto = risultato(2);
fprintf("La somma    : %.2f\n", somma); % stampa somma
fprintf("Il prodotto    : %.2f\n", prodotto); % stampa prodotto
```

Ciclo while, do while e for

I cicli `while` e `do while` sono strutture iterative che permettono di eseguire ripetutamente un blocco di istruzioni fino al verificarsi di una determinata condizione. La loro principale differenza risiede nella posizione del controllo della condizione, che determina il numero minimo di esecuzioni del ciclo.

Il ciclo `while` è un ciclo pre-condizionato, ovvero verifica la condizione prima di eseguire il blocco di istruzioni. Se la condizione risulta falsa fin dall'inizio, il corpo del ciclo non viene mai eseguito.

Il ciclo `do while` è un ciclo post-condizionato, ovvero verifica la condizione dopo l'esecuzione del blocco di istruzioni. Questo implica che il codice viene eseguito almeno una volta, indipendentemente dalla condizione.

Il ciclo `for` è una struttura iterativa con una sintassi più compatta, utilizzata quando il numero di iterazioni è noto in anticipo.

La sintassi del ciclo `for` è la seguente:

```
for variable = inizio:incremento:fine
    % Corpo del ciclo
end
```

Dove:

- `variabile`: contatore del ciclo.
- `inizio`: valore iniziale del contatore.
- `incremento`: valore di incremento (di default = 1).
- `fine`: valore finale del contatore.

Funzioni annidate

Le funzioni annidate sono funzioni definite all'interno di altre funzioni. Possono accedere alle variabili della funzione esterna, consentendo un certo livello di condivisione dei dati senza doverli passare esplicitamente come argomenti.

Esempio:

```
function output = funzioneEsterna(input)
    % Corpo della funzione esterna
    output = funzioneAnnidata(input);
    function out = funzioneAnnidata(in)
        % Corpo della funzione annidata
    end
end
```

Le funzioni annidate in MATLAB possono accedere e modificare le variabili della funzione esterna. Sono utili per suddividere il codice senza creare funzioni separate,

migliorando leggibilità e riutilizzo senza necessità di passare molti parametri. Tuttavia, non possono essere chiamate dall'esterno della funzione che le contiene.

Ad esempio, possono essere impiegate per definire una funzione Main, che contiene ulteriori funzioni al suo interno, come accade in molti linguaggi di programmazione classici.

Esempio con funzioni locali:

```
function MainScript_Piramide_Funzioni_Locali()
    % Richiede all'utente di inserire la lunghezza della base e l'altezza della piramide
    base = input('Lunghezza della base: ');
    altezza = input('Altezza della piramide: ');
    % Calcola volume, superficie e apotema chiamando le funzioni locali
    volume = calcolaVolume(base, altezza);
    superficie = calcolaSuperficie(base, altezza);
    apotema = calcolaApotema(base, altezza);
    % Stampa i risultati con due decimali
    fprintf('Volume: %.2f\n', volume);
    fprintf('Superficie: %.2f\n', superficie);
    fprintf('Apotema: %.2f\n', apotema);
end
function volume = calcolaVolume(base, altezza)
    % Calcola l'area della base (quadrato)
    areaBase = base^2;
    % Calcola il volume della piramide: V = (areaBase * altezza) / 3
    volume = (areaBase * altezza) / 3;
end
function superficie = calcolaSuperficie(base, altezza)
    % Calcola l'apotema della piramide richiamando la funzione corrispondente
    apotema = calcolaApotema(base, altezza);
    % Calcola l'area della base (quadrato)
    areaBase = base^2;
    % Calcola l'area laterale: somma delle 4 facce triangolari
    areaLaterale = 2 * base * apotema;
    % Calcola la superficie totale: base + area laterale
    superficie = areaBase + areaLaterale;
end
function apotema = calcolaApotema(base, altezza)
    % Calcola il semilato della base
    semilato = base / 2;
    % Calcola l'apotema usando il teorema di Pitagora: a = sqrt((base/2)^2 + altezza^2)
    apotema = sqrt(semilato^2 + altezza^2);
end
```

Stesso esempio con funzioni annidate:

```
function MainScript_Piramide_Funzioni_Annidate()
    % Richiede all'utente di inserire la lunghezza della base e l'altezza della piramide
```

```
    base = input('Lunghezza della base: ');
    altezza = input('Altezza della piramide: ');
    % Calcola volume, superficie e apotema utilizzando funzioni annidate
    volume = calcolaVolume(base, altezza);
    superficie = calcolaSuperficie(base, altezza);
    apotema = calcolaApotema(base, altezza);
    % Stampa i risultati con due decimali
    fprintf('Volume: %.2f\n', volume);
    fprintf('Superficie: %.2f\n', superficie);
    fprintf('Apotema: %.2f\n', apotema);
    % Funzione annidata per il calcolo del volume della piramide
    function volume = calcolaVolume(base, altezza)
        % Calcola l'area della base della piramide (base quadrata)
        areaBase = base^2;
        % Formula per il volume di una piramide: V = (areaBase * altezza) / 3
        volume = (areaBase * altezza) / 3;
    end
    % Funzione annidata per il calcolo della superficie totale della piramide
    function superficie = calcolaSuperficie(base, altezza)
        % Calcola l'apotema della piramide richiamando la funzione corrispondente
        apotema = calcolaApotema(base, altezza);
        % Calcola l'area della base (quadrato)
        areaBase = base^2;
        % Calcola l'area laterale della piramide (somma delle 4 facce triangolari)
        areaLaterale = 2 * base * apotema;
        % La superficie totale    data dalla somma dell'area di base e dell'area laterale
        superficie = areaBase + areaLaterale;
    end
    % Funzione annidata per il calcolo dell'apotema della piramide
    function apotema = calcolaApotema(base, altezza)
        % Calcola il semilato della base (met  della lunghezza della base)
        semilato = base / 2;
        % Utilizza il teorema di Pitagora per calcolare l'apotema: apotema = sqrt((base/2)^2 + altezza^2)
        apotema = sqrt(semilato^2 + altezza^2);
    end
end
```

Interfaccia grafica

MATLAB offre strumenti per creare interfacce grafiche (GUI) attraverso la funzione `uicontrol`, che permette di generare pulsanti, caselle di testo, menu a tendina e altri

elementi interattivi.

Per prima cosa, si crea una finestra grafica (`figure`). Si aggiunge un pulsante (`pushbutton`) e si assegna una funzione di callback (`Callback`), che esegue un'azione quando il pulsante viene premuto.

Esempio:

```
% Creazione di una finestra grafica (GUI) con dimensioni specificate
f = figure('Position', [100, 100, 300, 200]);
% Creazione di un pulsante con testo, posizione e dimensioni specifiche
b = uicontrol('Style', 'pushbutton',... % Definisce il tipo di controllo (pulsante)
    'String', 'Attiva il pulsante qui',... % Testo visibile sul pulsante
    'Position', [100, 80, 100, 40]); % Definisce la posizione e le dimensioni del pulsante
% Assegna una funzione di callback al pulsante: quando viene premuto, chiama 'pulsanteAttivato'
b.Callback = @pulsanteAttivato;

% Definizione della funzione di callback associata al pulsante
function pulsanteAttivato(src, event)
    % Quando il pulsante viene premuto, stampa un messaggio nella console
    disp('Il pulsante  stato attivato!');
end
```

Vengono eseguiti i seguenti step:

1. La funzione `figure` crea una finestra grafica.
2. Il parametro `'Position'` definisce la posizione `[x, y, larghezza, altezza]`.
3. `'Style', 'pushbutton'` definisce un pulsante.
4. `'String', 'Attiva il pulsante qui'` imposta il testo visibile sul pulsante.
5. `'Position', [100, 80, 100, 40]` determina dove e quanto grande sarà il pulsante.
6. `b.Callback = @pulsanteAttivato` associa la funzione `pulsanteAttivato` al pulsante. Tale funzione accetta due argomenti (`src, event`) che identificano l'oggetto che ha attivato l'evento.
7. `disp('Il pulsante è stato attivato!')` stampa un messaggio nella console quando viene premuto il pulsante.

Memorizzazione delle informazioni

I computer non hanno memoria infinita, quindi non è possibile immagazzinare dati di grandezze continue in memoria. È necessario, pertanto, effettuare un campionamento che permetta di discretizzare i valori target tra tutti gli infiniti valori della grandezza

continua in analisi. Inoltre, non sarebbe utile ai fini ingegneristici conoscere con alta precisione (decimali dopo la virgola) una misura. Poiché, nella realtà, va sempre delimitato il dominio applicativo del progetto.

Si definisce una catena di acquisizione dati:

Realtà → Grandezza continua → Sensore → Condizionamento → Campionamento → Quantizzazione -

Informazioni in una immagine

Con soli tre colori RGB (rosso, verde e blu) è possibile definire i colori che vediamo nelle immagini. Ognuno dei tre canali ha più livelli di intensità, che permettono di rappresentare, con molteplici combinazioni, i colori.

I pixel RGB contengono sub-pixel che ne definiscono le intensità e, in base alla quantità di bit associata a ogni canale, le combinazioni sono più o meno varie.

Per ottenere 256 livelli per ogni canale RGB, sono necessari 8 bit per la codifica. Per leggere una foto in MATLAB, per vedere valori dei pixel e dei subpixel, si usa il comando:

```
% Lettura dell'immagine
immagine = imread("immagine.jpg");
% Visualizza le dimensioni dell'immagine
[M, N, D] = size(immagine);
% Subplot per il canale rosso
subplot(2, 3, 1); % Prima riga, prima colonna
canale_r = immagine; % Copia l'immagine originale
canale_r(:, :, 2) = 0; % Azzera il canale verde
canale_r(:, :, 3) = 0; % Azzera il canale blu
imshow(canale_r);
title("Canale rosso");
% Subplot per il canale verde
subplot(2, 3, 2); % Prima riga, seconda colonna
canale_g = immagine; % Copia l'immagine originale
canale_g(:, :, 1) = 0; % Azzera il canale rosso
canale_g(:, :, 3) = 0; % Azzera il canale blu
imshow(canale_g);
title("Canale verde");
% Subplot per il canale blu
subplot(2, 3, 3); % Prima riga, terza colonna
canale_b = immagine; % Copia l'immagine originale
canale_b(:, :, 1) = 0; % Azzera il canale rosso
canale_b(:, :, 2) = 0; % Azzera il canale verde
imshow(canale_b);
title("Canale blu");
% Subplot per l'immagine originale
subplot(2, 3, 4); % Seconda riga, prima colonna
imshow(immagine);
title("Immagine originale");
% Imposta tutti i pixel dei 3 canali a 255 (immagine bianca)
immagine(:, :, :) = 255;
% Subplot per l'immagine bianca
subplot(2, 3, 5); % Seconda riga, seconda colonna
imshow(immagine);
```

```
35 title("Immagine bianca");
```

Sistemi avionici

Il termine "avionica" indica la disciplina che si occupa della progettazione di tutti i componenti elettronici installati a bordo degli aeromobili e destinati all'acquisizione, elaborazione e presentazione delle informazioni utili al volo.

L'avionica indica tutte le apparecchiature che permettono la navigazione aerea, il calcolo della posizione e direzione dell'aereo e tutti i dispositivi elettronici che attuano il pilotaggio automatico del velivolo, in cui il pilota si affida ai soli strumenti di bordo.

In più l'avionica permette di monitorare tutti i parametri strutturali e prestazionali del velivolo, come ad esempio il rilevamento di eventuali guasti, ma anche il controllo dei motori al fine di eseguire un volo alla massima efficienza in termini di potenza, riducendo così il consumo di carburante e le sollecitazioni termiche e meccaniche prodotte.

La gestione del volo degli aeromobili moderni ha portato ad un avanzamento tecnologico dei comandi di volo. Infatti, per i comandi alle superfici di governo e di stabilizzazione dell'aereo sono oggi utilizzati comandi elettrici che trasmettono ai vari attuatori segnali generati dal sistema avionico o dai comandi manuali. Questo sistema è chiamato *fly by wire* e permette il buon funzionamento di una manovra oltre all'ottenimento di una buona stabilità in volo esclusivamente basandosi sul sistema avionico. Altri comandi di volo invece si avvalgono delle fibre ottiche come mezzo di trasmissione, portando ad un sistema chiamato *fly by light*, il cui vantaggio è l'eliminazione delle interferenze elettromagnetiche sugli impulsi dei comandi di volo, che possono purtroppo verificarsi a causa della presenza dei vari impianti elettrici o disturbi provenienti dall'antenna.

Strumentazione di bordo

La strumentazione di bordo indica gli strumenti utilizzati dal pilota al fine di ricevere le principali informazioni necessarie al mantenimento in volo del velivolo sulla rotta di navigazione, includendo dunque i dispositivi di comunicazione per la gestione del traffico aereo che permettono di interagire con il velivolo.

Gli strumenti di bordo possono essere suddivisi in tre gruppi principali:

1. Strumenti di volo, i quali indicano assetto, quota e velocità necessarie ad eseguire il volo anche in condizioni di scarsa visibilità esterna;

2. Strumenti dell'apparato propulsore, grazie ai quali i piloti ricevono informazioni in tempo reale, riguardanti lo stato di funzionamento dei motori;

3. Strumenti di navigazione, che svolgono la funzione di mantenere l'aeroplano sulla rotta desiderata.

Altimetro

L'altimetro è un barometro che misura la pressione atmosferica, trasformando la lettura di pressione in misura di quota grazie al rapporto tra pressione atmosferica e altezza sul livello del mare alla quale viene misurata.

Esso è costituito da una cassa a tenuta stagna collegata all'esterno tramite una presa statica, entro la quale si trova una capsula chiusa ermeticamente. All'interno della capsula

viene creata una depressione che è contrastata da una molla, la quale spinge contro le pareti, in modo da tenere la capsula nella posizione di riposo e segnare zero quando si trova in condizioni di atmosfera standard al livello del mare. All'aumentare della quota, la pressione diminuisce e la capsula si dilata e tramite un sistema di leve e ingranaggi avviene la trasformazione del moto rettilineo di dilatazione della capsula in moto rotatorio degli indici.

L'orizzonte artificiale

È lo strumento fondamentale per condurre un volo strumentale, ovvero privo di visibilità dell'orizzonte naturale. Esso è costituito da un giroscopio a tre gradi di libertà posto davanti al pilota con l'asse di rotazione verticale e rotore parallelo alla superficie terrestre indipendentemente dall'assetto assunto dall'aeroplano. Al rotore è applicata la linea dell'orizzonte che divide il "cielo" segnato in azzurro dalla "terra" segnata in marrone. Una sagoma che simboleggia il velivolo, solidale con l'involucro dello strumento, e una serie di marcature, consentono al pilota di visualizzare gli assetti via via assunti dall'aeroplano; quindi, di manovrarlo come se visualizzasse l'orizzonte naturale.

Strumenti giroscopici

Parte fondamentale della famiglia degli strumenti di volo è costituita dagli strumenti giroscopici, così chiamati perché il giroscopio è l'elemento meccanico alla base del loro funzionamento. Essi sono l'orizzonte artificiale o indicatore di assetto, il direzionale o indicatore di prua e l'indicatore di virata. I rotori dei loro giroscopi sono mantenuti in rotazione da getti d'aria generati da una pompa o da motorini elettrici.

Tali strumenti hanno visto un netto miglioramento tecnologico: a partire dagli anni '80, i giroscopi meccanici sono stati sostituiti con quelli ad anello laser. Ciò prevedeva che non si muovesse più un rotore, bensì un fascio di luce polarizzata, che tramite un sistema di specchi posti ai vertici di un triangolo, effettuava un percorso triangolare venendo quindi riflessa. Ogni minimo movimento intorno ai suoi tre assi veniva rilevato dal computer e inviato ai display degli strumenti, divenuti così di gran lunga più semplici, leggeri ed affidabili.

Strumentazione di volo in MATLAB

Nel contesto della simulazione di volo, l'accurata rappresentazione degli strumenti di bordo è essenziale per garantire un ambiente realistico e funzionale. In questa implementazione, vengono inizializzati e configurati strumenti di volo virtuali utilizzando le funzioni della libreria `uiaero`. Questi strumenti consentono di monitorare parametri critici come assetto, velocità, altitudine e temperatura dei gas di scarico, fondamentali per la gestione e il controllo del velivolo in condizioni operative. Gli output visuali dei comandi precedenti sono:

Figura 7: Orizzonte artificiale

Figura 8: temperatura dei gas di scarico

Figura 9: Altimetro

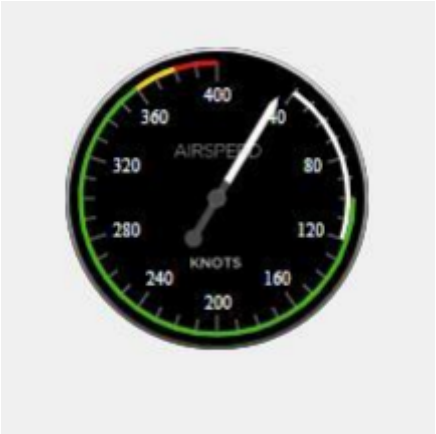

Figura 10: velocita dell'aria

Il codice seguente mostra l'inizializzazione degli strumenti principali:

```
% Inizializzazione degli strumenti di volo
horizon = uiaerohorizon; % Indicatore di assetto (orizzonte
    artificiale)
egt = uiaeroegt; % Indicatore della temperatura del gas di
    scarico (EGT)
airspeed = uiaeroairspeed; % Indicatore della velocit  dell'aria
altimeter = uiaeroaltimeter; % Altimetro per la quota di volo
```

Il codice seguente simula lo spostamento delle lancette in base al passo (sensibilità) impostato:

```
for i = 100:100:500 % Ciclo for che incrementa i da 100 a 500 con
    passo 100
    f = uifigure; % Crea una nuova finestra UI per ogni
        iterazione
    altimeterindicator = uiaeroaltimeter(f); % Crea indicatore
        dell'altimetro
    altimeterindicator.Value = i; % Imposta il valore della
        lancetta sul valore corrente di i
end
```

Il funzionamento dell'orizzonte artificiale, con simulazione dei valori, è rappresentato nel codice seguente:

```
for i = -2:1:2 % Itera da -2 a 2 con passo 1
    f = uifigure; % Crea una nuova finestra UI ad ogni iterazione
    artificialhorizon = uiaerohorizon(f); % Crea un orizzonte
        artificiale nella finestra
    % Imposta il valore dell'orizzonte artificiale:
    % - Il primo valore (20*i)    il bank angle (roll), varia da
        -40 a 40 gradi
    % - Il secondo valore (5+i)    il pitch angle, varia da 3 a 7
        gradi
    artificialhorizon.Value = [(20*i) (5+i)];
end
```

Sistemi di guida per la navigazione

Diversi sistemi sono impiegati per definire la posizione del velivolo nello spazio, essendo dunque di grande aiuto al pilota durante la crociera. La navigazione aerea è assistita dalle onde radio emesse da stazioni installate al suolo, che prendono il nome di radiofari.

Il radiogoniometro

Il radiogoniometro ha la funzione di indicare ai piloti la direzione da cui provengono le onde radio; dunque, la posizione dell'aereo rispetto al radiofaro che le ha emesse.

I radiogoniometri automatici installati nei velivoli sono gli ADF (*Automatic Direction Finder*), mentre quelli emittenti sono gli NDB (*Non Directional Beacon*), così chiamati perché trasmettono lo stesso segnale in ogni direzione.

La comunicazione ADF/NDB è ancora in uso, nonostante adesso sia stata sostituita dal sistema satellitare GPS.

Gli indicatori degli ADF che costituiscono la strumentazione aerea, sono gli RMI (*Radio Magnetic Indicator*), i quali presentano due indici sovrapposti ad un direzionale, in modo da potersi orientare simultaneamente verso due radiofari.

GPS

Il GPS si avvale di 24 satelliti Navstar, uniformemente spaziati a quattro a quattro su sei diverse orbite, alla quota di 20169 km e inclinate di 55 gradi rispetto al piano equatoriale terrestre. Ciò fa sì che da ogni punto della Terra si possano costantemente ricevere segnali provenienti da almeno sei satelliti, di cui almeno quattro fondamentali per determinare con precisione la posizione del velivolo.

La posizione fornita dal GPS è di tipo tridimensionale, ovvero espressa in latitudine, longitudine e altitudine sul livello del mare. Tale sistema misura il tempo impiegato dai segnali (emessi dai satelliti) a raggiungere il ricevitore, trasformandolo in distanza se moltiplicato per la velocità della luce. Il tempo e la posizione, come dati forniti dal GPS, sono poi elaborati dal computer di bordo.

La luce si propaga nel vuoto in linea retta alla velocità costante di 299792458 m/s, ma per semplicità, il valore è approssimato a 300000000 m/s.

ILS

Gli strumenti ILS (*Instrumental Landing Systems*), permettono di guidare la direzione di planata, permettendo ai velivoli di allinearsi precisamente con la pista in fase di atterraggio.

Satelliti TLC in MATLAB

Di seguito si presenta una visualizzazione di satelliti per TLC (Telecomunicazioni) in orbita prossima a quella geostazionaria, mediante un breve codice MATLAB.

```
% Creazione di uno scenario satellitare specificando:
% - L'ora di inizio della simulazione
% - L'ora di fine (1 giorno dopo l'inizio)
% - Il tempo di campionamento (60 secondi)
startTime = datetime(2023, 3, 21, 9, 30, 0); % Inizio simulazione
   : 21 marzo 2023 alle 09:30:00 UTC
```

```
stopTime = startTime + days(1); % Fine simulazione: 22 marzo 2023
    alla stessa ora
sampleTime = 60; % Intervallo di campionamento: 60 secondi
% Creazione dello scenario satellitare con i parametri definiti
   sopra
sc = satelliteScenario(startTime, stopTime, sampleTime);
% Definizione dei parametri orbitali per 5 satelliti usando
   elementi kepleriani
semiMajorAxis = [42000000; 43000000; 45000000; 46000000]; %
   Semiasse maggiore in metri
eccentricity = [0.02; 0.01; 0.01; 0.01; 0.1]; % Eccentricit
   dell'orbita
inclination = [0; 0; 0; 0; 0]; % Inclinazione in gradi (0
```

Nel caso di un solo satellite, in orbita geostazionaria (42162 km), il codice MATLAB della simulazione e l'output visualizzato sono i seguenti:

```
% Creazione di uno scenario satellitare con parametri
   personalizzati:
% - Tempo di inizio e fine simulazione
% - Intervallo di campionamento (frequenza temporale)
% - Inserimento di 5 satelliti con parametri orbitali specifici
% - Definizione dell'orario di inizio della simulazione (15
   aprile 2025, ore 18:00)
startTime = datetime(2025, 4, 15, 18, 0, 0);
% Fine della simulazione impostata a 24 ore dopo l'inizio
stopTime = startTime + days(1);
% Intervallo di campionamento: 60 secondi tra un frame e l'altro
sampleTime = 60;
% Creazione dello scenario satellitare con i parametri temporali
   specificati
sc = satelliteScenario(startTime, stopTime, sampleTime);
% - DEFINIZIONE DEI PARAMETRI ORBITALI PER 5 SATELLITI
% - Semiasse maggiore (distanza media dal centro della Terra) in
   metri
semiMajorAxis = 42164000;
% Eccentricit  dell'orbita (0 = cerchio perfetto, 0.02 = orbita
   leggermente ellittica)
eccentricity = 0.02;
% Inclinazione orbitale (0 gradi = orbita equatoriale)
inclination = 0;
% Longitudine del nodo ascendente (punto in cui il satellite
   attraversa l'equatore da sud a nord)
rightAscensionOfAscendingNode = 0;
% Argomento del pericentro (orientamento dell'ellisse rispetto al
    nodo ascendente)
argumentOfPeriapsis = 0;
% Anomalia vera iniziale (posizione iniziale del satellite lungo
   l'orbita)
trueAnomaly = 0;
% AGGIUNTA DEI SATELLITI ALLO SCENARIO
```

```
% Creazione dei satelliti nello scenario con i parametri orbitali definiti
% MATLAB aggiunger  5 satelliti identici in orbita secondo questi parametri
sat = satellite(sc,...
    semiMajorAxis,...
    eccentricity,...
    inclination,...
    rightAscensionOfAscendingNode,...
    argumentOfPeriapsis,...
    trueAnomaly);
% VISUALIZZAZIONE E SIMULAZIONE
% Mostra i satelliti nello scenario (attiva la visualizzazione 3D)
show(sat)
% Aggiunta delle tracce a terra (ground track) per ciascun satellite
% LeadTime = 7200 indica che viene visualizzata la traiettoria futura per 2 ore (7200 secondi)
groundTrack(sat, LeadTime=7200)
% Avvia la simulazione in tempo accelerato (30 volte pi  veloce del tempo reale)
play(sc, PlaybackSpeedMultiplier=30);
```

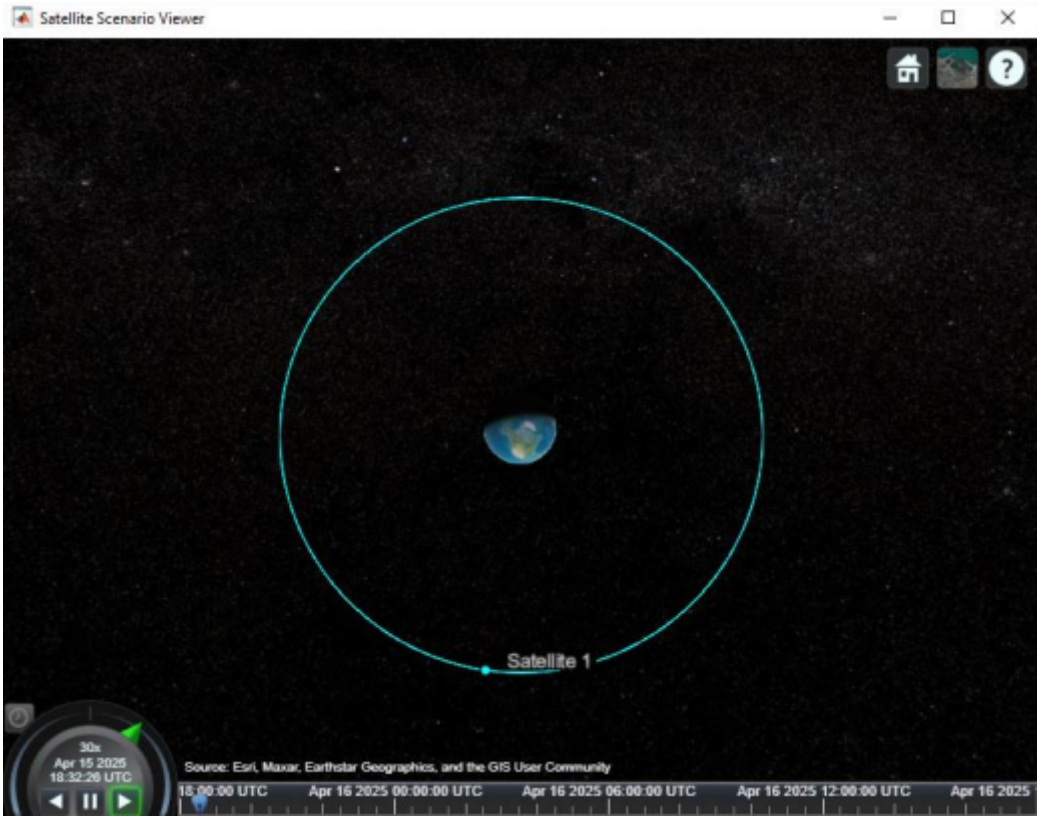

Figura 11: Simulazione in Satellite Scenario Viewer

Costellazione GPS - Modellazione e Visualizzazione 3D

In questo codice, sperimentiamo come MATLAB consenta di simulare un sistema GPS. Il codice seguente crea uno scenario 3D, aggiunge una costellazione di satelliti GPS e una stazione di terra, e infine esegue la simulazione della comunicazione tra satelliti e stazioni.

```
% Rappresentazione 3D di uno scenario satellitare con:
% - la Terra
% - una costellazione di satelliti GPS
% - una stazione di terra
% - verifica della comunicazione tra satelliti e stazione
% - Definizione dell'intervallo temporale dello scenario
startTime = datetime(2020, 1, 11); % Data/ora di inizio: 11 gennaio 2020
stopTime = startTime + days(0.5); % Fine simulazione: 12 ore dopo l'inizio
sampleTime = 60; % Tempo di campionamento: 60 secondi
% Creazione dello scenario satellitare senza simulazione automatica
sc = satelliteScenario(startTime, stopTime, sampleTime, 'AutoSimulate', false);
% AGGIUNTA DELLA COSTELLAZIONE GPS
% Aggiunta dei satelliti GPS allo scenario a partire da un file almanacco
sat = satellite(sc, "gpsAlmanac.txt");
% Simulazione manuale dello scenario nel tempo definito
% (avanza nel tempo ma non calcola automaticamente accessi o altri dati)
while advance(sc)
end
% Estrazione delle posizioni (stati) dei satelliti durante la simulazione
p = states(sat); % 'p' contiene posizione, velocit , ecc.
% Reset dello scenario al tempo iniziale per eseguire nuove simulazioni
restart(sc);
% AGGIUNTA DI UNA STAZIONE DI TERRA E ANALISI ACCESSI
% Aggiunta di una stazione di terra (posizione predefinita)
gs = groundStation(sc);
% Calcolo dell'accesso (visibilit ) tra ciascun satellite e la stazione di terra
ac = access(sat, gs);
% Simulazione manuale per determinare gli intervalli di accesso
while advance(sc)
end
% Estrazione degli intervalli temporali in cui esiste comunicazione (accesso)
intvls1 = accessIntervals(ac);
% VISUALIZZAZIONE DELLO SCENARIO
% Creazione di una finestra per visualizzare lo scenario in 3D
% 'ShowDetails' impostato su false per nascondere dettagli aggiuntivi
```

```
v = satelliteScenarioViewer(sc, 'ShowDetails', false);
% Riproduzione della simulazione nello scenario 3D
play(sc);
% MODALIT  DI SIMULAZIONE AUTOMATICA
% Abilitazione della simulazione automatica (calcola accessi,
   eventi, ecc.)
sc.AutoSimulate = true;
% Ricalcolo automatico degli intervalli di accesso con
   AutoSimulate attivo
intvls2 = accessIntervals(ac);
% Visualizzazione dello scenario con simulazione in esecuzione
play(sc);
```

Synthetic Aperture Radar (SAR)

Il Synthetic Aperture Radar (SAR) è una tecnologia radar altamente avanzata, progettata per ottenere immagini ad alta risoluzione della superficie terrestre. A differenza dei tradizionali sensori ottici, che dipendono dalla luce solare e non possono funzionare in condizioni di scarsa visibilità, il sistema SAR ha la capacità unica di vedere anche attraverso le nuvole e durante la notte. Questo è possibile grazie all'uso di onde radio nella banda delle microonde, che non sono influenzate dalle condizioni atmosferiche come la luce visibile.

SAR sfrutta un'antenna sintetica, che deriva dalla combinazione di segnali radar ricevuti da un'antenna fisicamente piccola ma, grazie a sofisticate tecniche matematiche, è possibile estendere, ottenendo immagini ad altissima risoluzione. Esso genera immagini bidimensionali (2D), che vengono analizzate secondo due parametri principali:

- Cross-range: che si riferisce alla direzione del volo della piattaforma mobile.
- Cross-track: che indica la direzione del puntamento dell'antenna radar.

Le applicazioni di questa tecnologia sono molteplici e spaziano in vari ambiti, tra cui il telerilevamento, la mappatura di precisione, il monitoraggio ambientale, la difesa e la sicurezza.

Un sistema SAR completo si compone di vari componenti chiave: piattaforma mobile, antenna radar, unità di elaborazione e software.

Nel contesto della programmazione MATLAB, vengono esplorati diversi aspetti del SAR, come la modellazione del sistema radar e dei bersagli, la simulazione del segnale radar e dell'immagine sintetica, e le tecniche di focalizzazione necessarie per ottenere immagini chiare e definite. Il modello SAR utilizzato nel codice è di tipo *stripmap*, il che significa che le immagini vengono generate sotto forma di mappe a strisce. Inoltre, viene utilizzata una modulazione di frequenza lineare (LFM), una tecnica che consente di migliorare l'efficienza del sistema, con larghezza di banda ampia e bassa potenza di trasmissione.

Spettro d'onda ed effetto Doppler

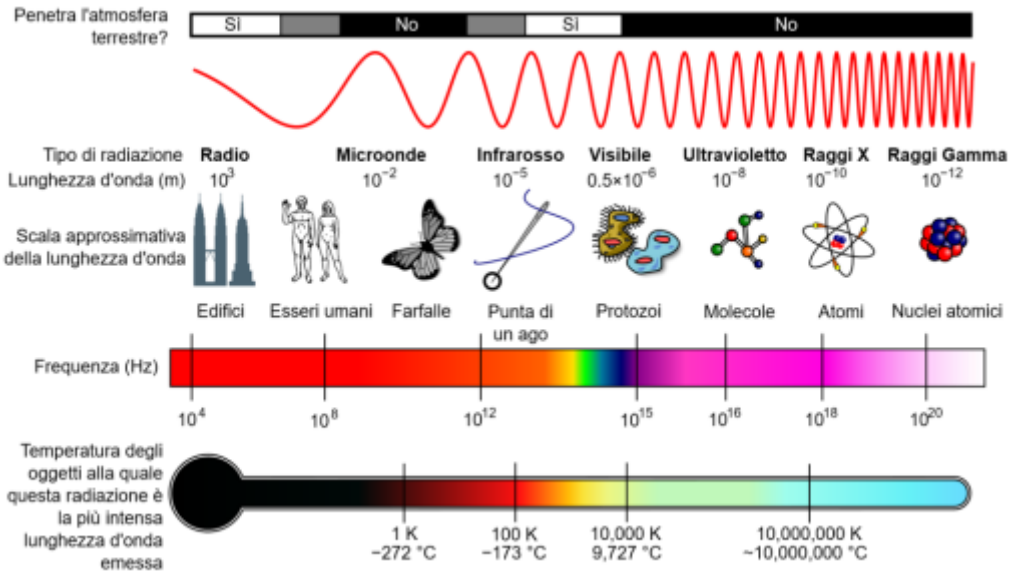

Figura 12: Rappresentazione dello spettro elettromagnetico

L'esempio avrà come frequenza centrale esattamente: 4×10^9 Hz. Dunque, si è nell'ambito delle microonde. Pertanto, nel codice servirà impostare anche la velocità della luce, ovvero 299792458 m/s.

Sigla della banda	Gamma di frequenza
L	1 – 2 GHz
S	2 – 4 GHz
C	4 – 8 GHz
X	8 – 12 GHz
K_u	12 – 18 GHz
K	18 – 26 GHz
K_a	26 – 40 GHz
Q	30 – 50 GHz
U	40 – 60 GHz
V	50 – 75 GHz
E	60 – 90 GHz
W	75 – 110 GHz
F	90 – 140 GHz
D	110 – 170 GHz

Figura 13: Suddivisione in bande del campo delle microonde

Il seguente codice MATLAB simula un sistema SAR (*Synthetic Aperture Radar*), un tipo di radar utilizzato per ottenere immagini ad alta risoluzione del terreno o di og-

Tabella 2: Suddivisione in bande del campo delle microonde

Sigla della banda	Gamma di frequenza
L	1-2 GHz
S	2-4 GHz
C	4-8 GHz
X	8-12 GHz
K_u	12-18 GHz
K	18-26 GHz
K_a	26-40 GHz
Q	30-50 GHz
U	40-60 GHz
V	50-75 GHz
E	60-90 GHz
W	75-110 GHz
F	90-140 GHz
D	110-170 GHz

getti tramite l'elaborazione di segnali riflessi, acquisiti durante il movimento del radar (tipicamente montato su un aereo o satellite).

```
% SAR - Simulazione di un sistema Radar ad Apertura Sintetica (SAR)
% Costante fisica: velocit  della luce (circa 299792458 m/s)
c = physconst('LightSpeed');
% Frequenza centrale del radar SAR (4 GHz = microonde)
fc = 4e9;
% Risoluzione desiderata in range (distanza)
rangeResolution = 3;
% Risoluzione desiderata in cross-range (lateralmente al moto)
crossRangeResolution = 3;
% Larghezza di banda del segnale (derivata dalla risoluzione)
bw = c / (2 * rangeResolution);
% Pulse Repetition Frequency (frequenza di ripetizione degli impulsi)
prf = 1000;
% Lunghezza dell'antenna virtuale formata dal movimento del radar (in metri)
aperture = 4;
% Larghezza dell'impulso (pulse duration)
tpd = 3e-6;
% Frequenza di campionamento del segnale radar (sample rate)
fs = 120e6;
% Configurazione del segnale radar LFM (chirp lineare)
waveform = phased.FMWaveform('SampleRate', fs,...
    'PulseWidth', tpd, 'PRF', prf, 'SweepBandwidth', bw);
% Parametri del volo radar (velocit  e durata)
speed = 100; % m/s
flightDuration = 4; % secondi
% Configurazione della piattaforma radar (posizione iniziale e velocit )
```

```
radarPlatform = phased.Platform('InitialPosition',
    [0;-200;500],...
    'Velocity', [0; speed; 0]);
% Tempo tra due impulsi successivi
slowTime = 1 / prf;
% Numero totale di impulsi trasmessi durante il volo
numpulses = flightDuration / slowTime + 1;
% Massima distanza osservabile dal radar (in metri)
maxRange = 2500;
% Numero di campioni per ciascun impulso in base alla distanza
    massima
truncrangesamples = ceil((2 * maxRange / c) * fs);
% Tempo veloce (campioni lungo il range per ogni impulso)
fastTime = (0:1/fs:(truncrangesamples-1)/fs);
% Distanza di riferimento per elaborazione SAR (centro della
    scena)
Rc = 1000;
% Configurazione antenna (elemento coseno, banda operativa)
antenna = phased.CosineAntennaElement('FrequencyRange', [1e9 6e9
    ]);
% Calcolo guadagno antenna a partire dall'apertura
antennaGain = aperture2gain(aperture, c / fc);
% Blocco trasmettitore
transmitter = phased.Transmitter('PeakPower', 50e3, 'Gain',
    antennaGain);
% Radiatore del segnale verso il bersaglio
radiator = phased.Radiator('Sensor', antenna,...
    'OperatingFrequency', fc, 'PropagationSpeed', c);
% Collettore dei segnali riflessi
collector = phased.Collector('Sensor', antenna,...
    'PropagationSpeed', c, 'OperatingFrequency', fc);
% Ricevitore con amplificatore e rumore (Noise Figure in dB)
receiver = phased.ReceiverPreamp('SampleRate', fs, 'NoiseFigure',
    30);
% Canale di propagazione in spazio libero bidirezionale
channel = phased.FreeSpace('PropagationSpeed', c,...
    'OperatingFrequency', fc, 'SampleRate', fs,...
    'TwoWayPropagation', true);
% Definizione dei bersagli (posizioni statiche e RCS costante = 1
    m^2)
targetpos = [800, 0, 0; 1000, 0, 0; 1300, 0, 0]';
targetvel = zeros(3); % bersagli statici
% Blocco bersagli radar
target = phased.RadarTarget('OperatingFrequency', fc, 'MeanRCS',
    [1, 1, 1]);
% Piattaforma dei bersagli (posizioni iniziali e velocit )
pointTargets = phased.Platform('InitialPosition', targetpos,...
    'Velocity', targetvel);
% Visualizzazione della scena a terra (ground truth)
figure(1);
h = axes;
```

```
plot(targetpos(2,1), targetpos(1,1), '*g'); hold all;
plot(targetpos(2,2), targetpos(1,2), '*r');
plot(targetpos(2,3), targetpos(1,3), '*b'); hold off;
set(h, 'Ydir', 'reverse');
xlim([-10 10]);
ylim([700 1500]);
title('Ground Truth');
ylabel('Range');
xlabel('Cross-Range');
% Preallocazione del segnale ricevuto (raw data SAR)
refangle = zeros(1, size(targetpos, 2));
rxsig = zeros(truncrangesamples, numpulses);
% Ciclo principale: trasmissione e ricezione per ogni impulso
for i = 1:numpulses
    % Aggiorna posizione radar e bersagli
    [radarpos, radarvel] = radarPlatform(slowTime);
    [targetpos, targetvel] = pointTargets(slowTime);
    % Calcolo distanza e angolo tra radar e bersagli
    [targetRange, targetAngle] = rangeangle(targetpos, radarpos);
    % Generazione impulso LFM (chirp)
    sig = waveform();
    sig = sig(1:truncrangesamples); % taglio alla lunghezza
       desiderata
    % Trasmissione del segnale
    sig = transmitter(sig);
    % Fissare l'angolo orizzontale (azimut) a 0
    targetAngle(1,:) = refangle;
    % Irradiazione verso i bersagli
    sig = radiator(sig, targetAngle);
    % Propagazione del segnale nello spazio
    sig = channel(sig, radarpos, targetpos, radarvel, targetvel);
    % Riflessione dai bersagli
    sig = target(sig);
    % Raccolta del segnale riflesso
    sig = collector(sig, targetAngle);
    % Ricezione e salvataggio del segnale ricevuto
    rxsig(:,i) = receiver(sig);
end
% Visualizzazione del segnale radar ricevuto (grezzo, prima
   elaborazione)
figure(2)
imagesc(real(rxsig));
title('SAR Raw Data');
xlabel('Cross-Range Samples');
ylabel('Range Samples');
% Compressione del range tramite filtro adattato (matched filter)
pulseCompression = phased.RangeResponse('RangeMethod', 'Matched
   Filter',...
    'PropagationSpeed', c, 'SampleRate', fs);
% Calcolo coefficiente di filtro adattato
matchingCoeff = getMatchedFilter(waveform);
```

```
% Applicazione del filtro al segnale ricevuto
[cdata, rnggrid] = pulseCompression(rxsig, matchingCoeff);
% Visualizzazione del segnale compresso in range
figure(3)
imagesc(real(cdata));
title('SAR Range Compressed Data');
xlabel('Cross-Range Samples');
ylabel('Range Samples');
% Compressione azimutale (cross-range)
% Metodo 1: Range Migration Algorithm (RMA)
rma_processed = helperRangeMigration(cdata, fastTime, fc, fs, prf
    , speed, numpulses, c, Rc);
% Crea griglia target
[xg, yg] = meshgrid(-500:crossRangeResolution:500, Rc - 750:Rc +
   750);
% Chiamata alla funzione corretta
bpa_processed = helperBackProjection(cdata, xg, yg, fastTime, fc,
    fs, prf, speed, crossRangeResolution, c);
% Visualizzazione risultati - RMA
figure(4)
imagesc(abs(rma_processed(1700:2300,600:1400).'));
title('SAR Data focused using Range Migration algorithm');
xlabel('Cross-Range Samples');
ylabel('Range Samples');
% Visualizzazione risultati - BPA
figure(5)
imagesc(abs(bpa_processed(600:1400,1700:2300)));
title('SAR Data focused using Back-Projection algorithm');
xlabel('Cross-Range Samples');
ylabel('Range Samples');
```

Segnali sinusoidali

Un segnale sinusoidale è costituito dalle seguenti componenti:

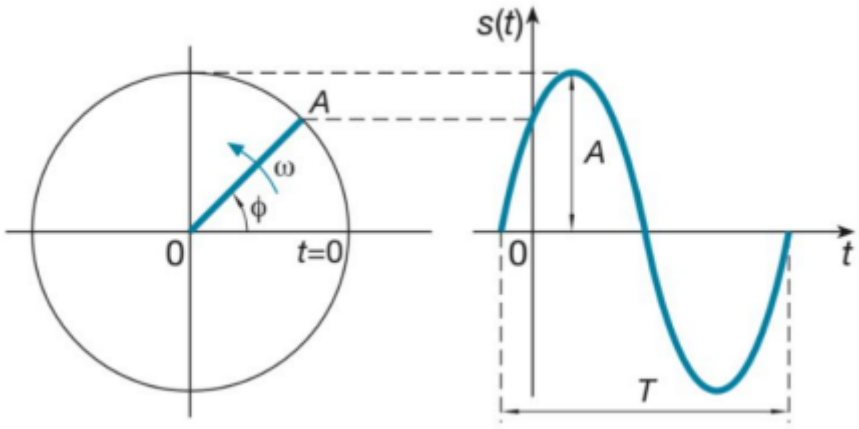

Figura 14: Rappresentazione di un segnale sinusoidale, ottenuta dalla proiezione sull'asse delle ordinate s(t) di un vettore ruotante con velocità angolare e ampiezza A.

$$s(t) = A\sin(2\pi ft + \phi) \tag{14}$$

VHF Omnidirectional Range (VOR)

Il VOR è un sistema di radionavigazione terrestre che consente agli aeromobili di determinare la propria posizione e rotta in base a segnali radio emessi da una stazione a terra, solitamente onde radio nella banda VHF ($108.0 - 117.95\,\text{MHz}$).

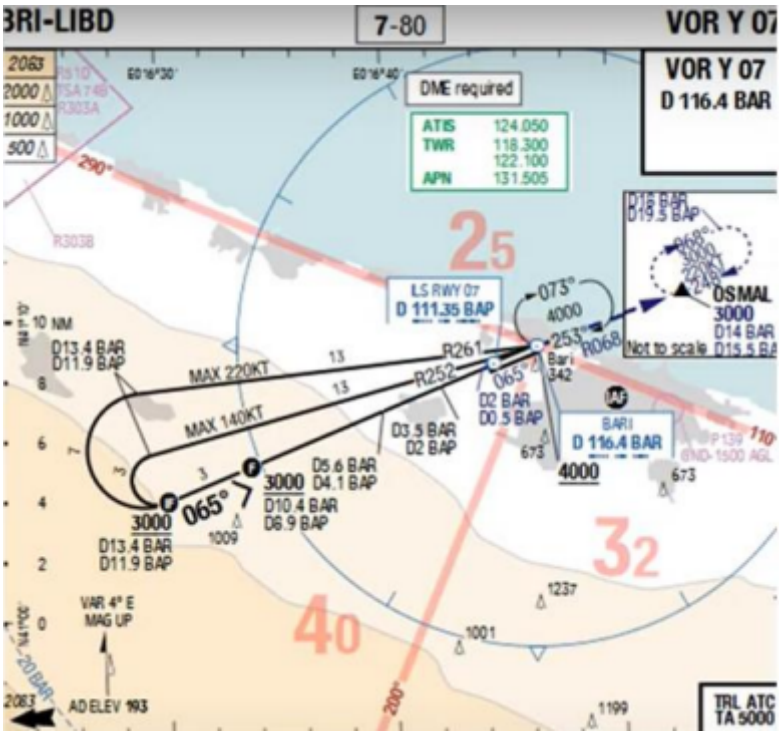

Figura 15: Carta di approdo all'Aeroporto di Bari

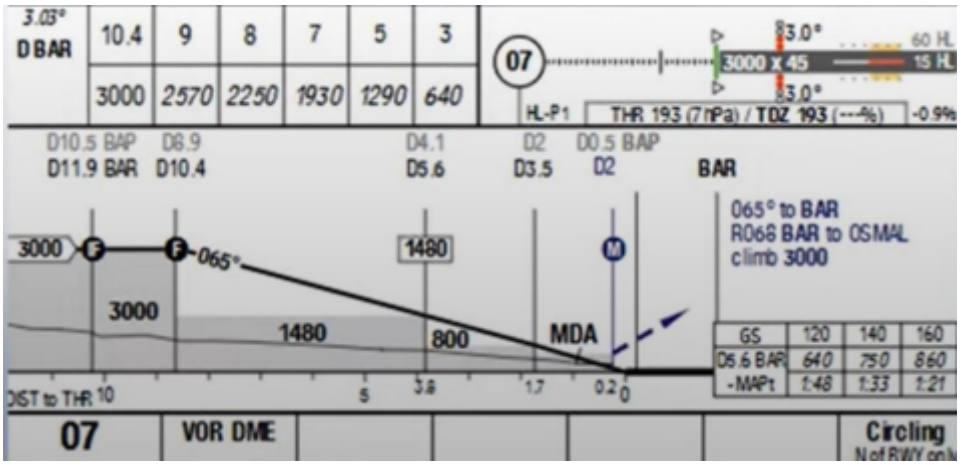

Figura 16: Aeroporto di Bari in sezione

Nella mappa si nota il riferimento al sistema VOR (*VHF Omnidirectional Range*).

Per quanto concerne il funzionamento del sistema VOR in sintesi, esso trasmette due segnali: un'onda portante modulata in AM (*Amplitude Modulation*) e un'onda modulata

in FM (*Frequency Modulation*). La differenza angolare tra questi due segnali fornisce l'informazione relativa dell'aeromobile rispetto alla stazione VOR.

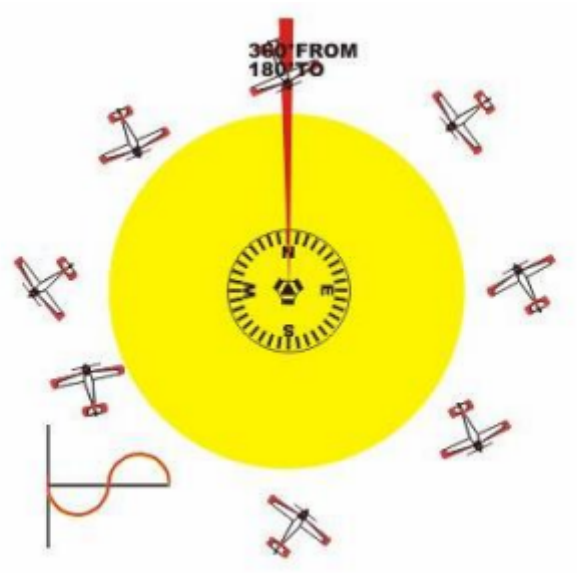

Figura 17: Virata coordinata di un aeroplano nel piano orizzontale con bussola e VOR

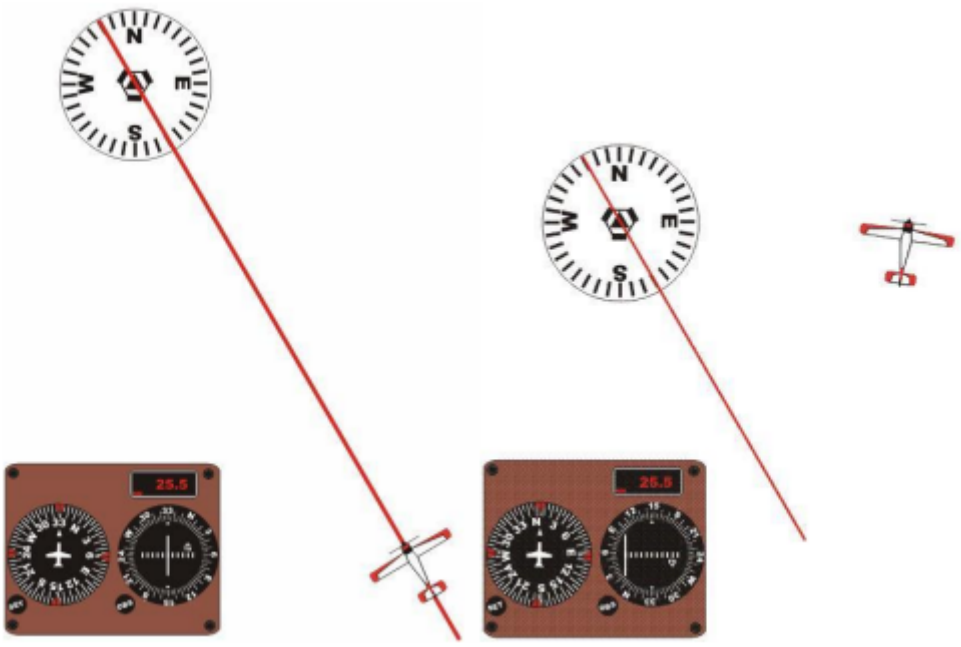

Figura 18: Esempi di VOR

Il sistema VOR, insieme al sistema DME, è un ottimo e collaudato metodo per la radionavigazione. Il principio di funzionamento si basa sull'emissione simultanea di due segnali: uno di riferimento, omnidirezionale e costante in fase, e uno variabile, la cui fase cambia in funzione dell'angolo di emissione rispetto al nord magnetico. Il ricevitore di bordo è in grado di confrontare questi due segnali e determinare l'angolo, detto radiale, che identifica la direzione del velivolo rispetto alla stazione VOR. In sintesi, l'apparato a

bordo mostrerà la posizione dell'aeromobile rispetto al radiale (rotta selezionata), ovvero, se il velivolo si trova a destra, a sinistra o in linea con la rotta selezionata.

L'indicazione fornita dal VOR non dipende dalla prua dell'aeromobile, ma esclusivamente dalla relazione geometrica tra la posizione del velivolo e la radiale selezionata. Ciò consente una navigazione più stabile e affidabile, indipendentemente dall'orientamento dell'aereo.

Il quadrante VOR (CDI) indica lo spostamento rispetto alla rotta selezionata ed è costituito da:

- Indicatore di deviazione: rappresenta lo scostamento dell'aeromobile rispetto alla radiale selezionata.
- Indicazione TO/FROM: indica se il velivolo si sta dirigendo verso la stazione VOR oppure si allontana.
- Corona graduata e manopola OBS: selettore ad anello omnidirezionale.
- Indicazione DME (se disponibile): quando lo strumento è integrato con un DME (*Distance Measuring Equipment*), ovvero apparecchiature per la misurazione della distanza, viene visualizzata anche la distanza in miglia nautiche dalla stazione VOR. Questa informazione compare su un display digitale e consente al pilota di conoscere in tempo reale la propria posizione lungo la radiale.

App per la progettazione e simulazione di scenari

`Tracking Scenario Designer` è un'App interna a MATLAB, che permette la creazione, simulazione ed esportazione di scenari realistici utili allo sviluppo e al test di algoritmi di tracciamento. Permette, inoltre, di modellare ambienti 2D e 3D nei quali piattaforme mobili, dotate di sensori, interagiscono in tempo e spazio. Diverse entità si muovono secondo traiettorie definite, mentre sensori a bordo delle stesse raccolgono dati utili alla stima della loro posizione. Risulta, dunque, uno strumento potente per la creazione, simulazione ed esportazione di scenari abbastanza realistici.

Creazione di uno scenario di base

La traiettoria può essere definita nel pannello della piattaforma tramite l'inserimento di *waypoint*. Eventualmente, è possibile studiare il comportamento dei sensori radar nel `Tracking Scenario Designer` ed esportare gli scenari creati per applicazioni pratiche.

Il ruolo della torre di controllo nel Tracking Scenario Designer

La torre può essere utilizzata come postazione per installare sensori radar, lidar o EO/IR (ottici/termici), fornendo un punto di osservazione elevato e stabile.

Spazio aereo

Lo spazio aereo è soggetto a una suddivisione verticale:

- Spazio Inferiore
- Spazio Superiore

Questa distinzione non è soltanto formale, ma risponde a diverse esigenze operative. In alto, il traffico è prevalentemente costituito da voli commerciali a lungo raggio, che seguono rotte predeterminate; in basso, si concentrano invece voli locali, traffico generale e attività militari o di addestramento.

Spazi aerei intorno agli aeroporti

- ATZ (*Aerodrome Traffic Zone*)
- CTR (*Control Zone*)
- TMA (*Terminal Control Area*)

Altitudine

Nel contesto aeronautico, l'altitudine rappresenta la misura della distanza verticale tra un aeromobile e un determinato punto di riferimento, solitamente il livello del mare o il terreno sottostante. Sebbene possa apparire come una nozione elementare, in realtà l'altitudine è una delle variabili più complesse e cruciali nella navigazione aerea. Comprenderne le varie declinazioni è essenziale:

- **Altitudine Assoluta**: rappresenta la distanza verticale esatta tra un aeromobile e la superficie terrestre immediatamente sottostante. La misurazione dell'altitudine assoluta viene effettuata mediante un altimetro radar, uno strumento che funziona inviando impulsi radio verso il terreno. Tale informazione è utile nelle fasi di decollo, atterraggio o durante il volo a bassa quota sopra rilievi montuosi o aree urbane.
- **Altitudine Vera**: indica la distanza verticale tra un aereo e il livello medio del mare (spesso indicato con l'acronimo AMSL: *Above Mean Sea Level*). Viene utilizzata nelle carte aeronautiche per definire traiettorie, rotte e separazioni tra traffico aereo. Gli strumenti di bordo, in particolare gli altimetri barometrici, stimano l'altitudine vera basandosi sulla pressione atmosferica. Per ottenere letture affidabili, questi strumenti devono essere regolarmente calibrati in base alla pressione al livello del mare fornita dai centri di controllo o dalle stazioni meteo.
- **Altitudine Indicata**: corrisponde al valore mostrato direttamente sull'altimetro di bordo, ed è calcolata sulla base della pressione atmosferica rilevata in volo. L'altimetro è generalmente tarato su valori standard atmosferici (una pressione di $1013,25\,\text{hPa}$ e una temperatura di $15°\text{C}$ al livello del mare). Quando le condizioni atmosferiche effettive si discostano da questi parametri, l'altitudine indicata può risultare diversa da quella reale. Riveste un ruolo fondamentale nella gestione del traffico aereo. Gli enti di controllo assegnano livelli di volo basati su queste letture standardizzate per garantire separazione verticale tra aeromobili. Finché tutti gli altimetri sono tarati sulle stesse condizioni di riferimento, è possibile mantenere la distanza di sicurezza tra i velivoli, anche se l'altitudine effettiva può variare leggermente da quella mostrata.
- **Altitudine di Pressione**: rappresenta la distanza verticale di un aeromobile rispetto a un livello di riferimento teorico chiamato livello di pressione standard (o *Standard Datum Plane*, SDP), dove la pressione atmosferica è fissata a $1013,25\,\text{hPa}$. Questo valore standard consente di normalizzare le letture altimetriche, rendendole

indipendenti dalle variazioni locali della pressione atmosferica. Per ottenere questa lettura, i piloti impostano l'altimetro sul valore standard senza tener conto della pressione effettiva presente in quota. Il valore mostrato rappresenta quindi la quota rispetto al SDP, non al livello reale del mare o del terreno sottostante. L'altitudine di pressione è fondamentale per la navigazione ad alta quota.

- **Altitudine di Densità**: permette di valutare il comportamento di un velivolo in funzione delle condizioni atmosferiche presenti. Con l'aumentare dell'altitudine, dell'umidità o della temperatura, la densità dell'aria tende a diminuire. Questa rarefazione dell'aria ha effetti negativi sulle prestazioni aerodinamiche dell'aeromobile: riduce la spinta generata dai motori, diminuisce la portanza delle ali e allunga le distanze necessarie per il decollo e la salita. È essenziale soprattutto in estate o in aeroporti situati a elevate altitudini, dove le condizioni ambientali possono discostarsi in modo significativo dai parametri standard. I piloti la utilizzano per prevedere in anticipo come reagirà l'aereo in fase di decollo, atterraggio o in manovre critiche, così da garantire sicurezza e prestazioni adeguate.

Droni

```
% Parametri del tempo
t = linspace(0, 20, 500);
raggio = 5;
altezza = 10;
vel_angolare = pi / 5;
% Traiettoria
x = raggio * cos(vel_angolare * t);
y = raggio * sin(vel_angolare * t);
z = linspace(0, altezza, length(t));
% Setup figura
figure;
axis equal;
grid on;
xlabel('X (m) ');
ylabel('Y (m) ');
zlabel('Z (m) ');
title('Volo realistico di un drone stilizzato');
view(3);
% Disegno traiettoria
plot3(x, y, z, 'b--');
% Definizione del drone stilizzato
drone_body = [-0.3 0.3 NaN 0 0; 0 0 NaN -0.3 0.3]; % bracci a
    croce
% Animazione
hold on;
for i = 1:length(t)
    % Posizione corrente
    xc = x(i);
    yc = y(i);
    zc = z(i);
    % Aggiorna drone
```

```
    h = plot3(xc + drone_body(1,:), yc + drone_body(2,:), zc *
        ones(1, size(drone_body, 2)), 'k-', 'LineWidth', 3);
    % Disegna il corpo centrale del drone
    plot3(xc, yc, zc, 'ro', 'MarkerSize', 10, 'MarkerFaceColor',
        'r');
    pause(0.01);
    % Cancella la posizione precedente
    if i < length(t)
        delete(h);
    end
end
```

STL

Un file STL (*Stereolithography*) è un formato standard utilizzato per rappresentare modelli 3D. È molto usato per simulazioni ingegneristiche, e anche per la visualizzazione 3D in ambienti come MATLAB.

Caratteristiche principali di un file `.stl`:

- **Rappresentazione della superficie**: Un file STL descrive solo la geometria esterna dell'oggetto, suddividendola in triangoli (facce). Ogni triangolo è definito da 3 vertici e una normale (vettore perpendicolare alla superficie).
- **Nessun colore, texture o unità di misura**: Non contiene informazioni su materiali, colori, texture.

Due versioni:

- **ASCII STL**: leggibile come testo. Ogni triangolo è descritto da righe di testo.
- **Binary STL**: molto più compatto, ma non leggibile direttamente.

Utilizzo possibile: Simulazioni in MATLAB (per visualizzare o simulare oggetti tridimensionali). Di seguito un ulteriore codice MATLAB con simulazione semplice del volo di un drone:

```
% Parametri del tempo
t = linspace(0, 20, 500); % tempo da 0 a 20 secondi
% Traiettoria del drone: spirale elicoidale
raggio = 5; % raggio della spirale
altezza = 10; % altezza massima
vel_angolare = pi / 5; % velocit  angolare
% Coordinate del volo
x = raggio * cos(vel_angolare * t);
y = raggio * sin(vel_angolare * t);
z = linspace(0, altezza, length(t)); % salita lineare
% Plot della traiettoria
figure;
plot3(x, y, z, 'b', 'LineWidth', 2);
grid on;
xlabel('X␣(m)␣');
ylabel('Y␣(m)␣');
```

```
zlabel('Z (m) ');
title('Traiettoria di volo di un drone');
axis equal;
view(3);
% Simulazione del movimento con animazione
hold on;
drone = plot3(x(1), y(1), z(1), 'ro', 'MarkerSize', 10, '
   MarkerFaceColor', 'r');
for i = 1:length(t)
    set(drone, 'XData', x(i), 'YData', y(i), 'ZData', z(i));
    pause(0.01);
end
```

UAV Scenario Designer

MATLAB presenta una App anche per i droni. L'app `UAV Scenario Designer` di MATLAB (parte dell'*Aerospace Toolbox*) è un ambiente interattivo che permette di progettare, configurare e simulare scenari UAV realistici in 3D. È pensata per facilitare lo sviluppo, il collaudo e la visualizzazione di missioni UAV (*Unmanned Aerial Vehicle*) o droni, senza la necessità iniziale di scrivere codice complesso.

Funzionalità principali di `UAV Scenario Designer`:

1. **Progettazione visiva dello scenario**:
   [label=.]Ambiente 3D realistico per costruire uno scenario UAV. Inserimento di piattaforme UAV (come droni), bersagli, edifici, ostacoli, *waypoint*, e traiettorie. Importazione di modelli STL o CAD per una visualizzazione dettagliata dei velivoli o ostacoli. Possibilità di inserire radar, sensori e torrette di monitoraggio per simulazioni complete.

2. **Configurazione delle traiettorie**:
   [label=.]Definizione interattiva di percorsi *waypoint*-based (punto-punto). Possibilità di definire velocità, altitudine, orientamento e manovre per ogni tratto. Supporto per più UAV e bersagli in contemporanea. Visualizzazione dell'intera traiettoria in tempo reale.

3. **Integrazione con sensori e modelli**:
   [label=.]Inserimento di sensori a bordo UAV (es. radar, GPS, IMU, telecamere). Aggiunta di stazioni radar a terra (es. su torre), utili per scenari di sorveglianza aeroportuale. Simulazione della linea di vista (LOS), campo visivo, e copertura radar.

4. **Simulazione e generazione di codice**:
   [label=.]Esecuzione della simulazione dello scenario UAV direttamente dall'app. Registrazione e visualizzazione dei dati di posizione, velocità, orientamento, ecc. Generazione automatica del codice MATLAB corrispondente allo scenario progettato (modificabile per simulazioni avanzate). Esportazione dei dati per testare algoritmi di *tracking*, navigazione, controllo o *sensor fusion*.

5. **Esempi di utilizzo**:

[label=.]Progettazione di missioni UAV per sorveglianza di un aeroporto. Test di tracciamento radar con bersagli mobili. Simulazione della rotta di un drone per evitare ostacoli. Studio di scenari multiveicolo con sensori collaborativi.

UAV

Il drone (tecnicamente noto come UAV, *Unmanned Aerial Vehicle*) è un velivolo senza pilota a bordo, controllato da remoto o in modo autonomo tramite software di navigazione. I droni possono essere dotati di sensori, telecamere, GPS, motori elettrici e sistemi di comunicazione, rendendoli adatti a una vasta gamma di applicazioni civili, industriali, scientifiche e militari.

Tipologie di droni:

- **Quadricottero**: il più comune, usato per riprese, ispezioni e hobby.
- **Ad ala fissa**: simile a un aereo. Maggiore autonomia, usato per mappature e sorveglianza.
- **Ibrido**: combina vantaggi di multirotore e ala fissa.

Componenti principali:

- **Telaio (Frame)**: struttura che sostiene tutti i componenti.
- **Motori e eliche**: permettono la spinta e la stabilizzazione.
- **Batteria**: alimenta il drone (di solito agli ioni di litio).
- **Controller di volo (Flight Controller)**: il "cervello" del drone, gestisce i movimenti.
- **Sensori**:
  - IMU (accelerometri e giroscopi)
  - GPS
  - Altimetri
  - Sensori anticollisione
- **Telecomando o stazione di controllo a terra**: per il pilotaggio manuale.
- **Telecamera o payload**: per acquisire immagini/video o effettuare misure.

Applicazioni dei droni:

- **Agricoltura**: monitoraggio colture, irrorazione, mappatura del terreno.
- **Industria**: ispezioni di infrastrutture, cantieri, linee elettriche.
- **Sicurezza e soccorso**: ricerca dispersi, rilevamento incendi, sorveglianza.
- **Logistica**: consegne rapide, anche in aree remote.
- **Cinema/Fotografia**: riprese aeree, filmati promozionali.

- **Ricerca scientifica**: monitoraggio ambientale, osservazione meteorologica, ecc.

Controllo e Navigazione: I droni possono operare in tre modalità principali:

- **Manuale**: il pilota controlla direttamente ogni movimento.
- **Assistita (GPS)**: il drone mantiene la posizione e l'altitudine grazie al GPS.
- **Autonoma**: il drone segue un piano di volo preimpostato con *waypoint*.

Scenari relativi a possibili approfondimenti:

- Elettronica e sensoristica.
- Sistemi di controllo automatico.
- Programmazione ad esempio con MATLAB.

In MATLAB, ad esempio, è possibile simulare il comportamento dinamico di un drone, progettarne i controlli, o visualizzarne il movimento in ambienti realistici.