POLITECNICO DI BARI

Sede di Taranto – TTEC

Corso di Laurea in Ingegneria Informatica e dell'Automazione P-TECH

Tema d'Anno in

Programmazione dei Sistemi Avionici

Titolo

Elaborato

Docente

Prof. Antonio Satriano

Studente

Roberto Carriero

Matricola

587640

Anno Accademico 2024 – 2025

Modalità d’esame

L'esame prevede le seguenti fasi:

1. Caricamento dell'elaborato
2. Discussione dell'elaborato in modalità virtuale
3. Domande orali sul codice per migliorare il voto

Format del documento d'esame (tema d’anno)

L'elaborato deve rispettare il seguente format:

- Intestazione: logo politecnico di bari, logo sede di taranto, corso di laurea, anno accademico, nome, cognome e matricola dello studente, nome del docente (antonio satriano), titolo dell'elaborato
- Indice
- Prefazione: descrizione breve dell’elaborato
- Introduzione: introduzione generale e approfondimento sulla programmazione, l’avionica e i sistemi avionici
- MATLAB: introduzione al codice e spiegazione del suo scopo
- Inclusione del codice MATLAB con commenti esplicativi
- Output del codice (screenshot di tabelle o grafici se necessari)
- Analisi dei risultati ottenuti e possibili miglioramenti
- Conclusioni

Concetti fondamentali

- Programmazione
- Scrivere istruzioni elementari in un linguaggio specifico.
- Salvare il codice in pacchetti modulari e riutilizzabili.
- Assicurarsi che il codice funzioni correttamente ed eviti ridondanze.
- Sistemi Avionici
- Definizione di sistemi.
- Definizione di sistemi avionici.
- Introduzione alla programmazione dei sistemi avionici.

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.

Appendice (nell'elaborato finale mettere gli appunti e il codice prodotto durante il corso)

MATLAB elementi base

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)
- non possono 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 >> mi 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.

```
V = [1 2 3];
V = [1, 2, 3];
```

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

```
V = [1;2;3;4;5]
```

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

V(indice)

Es.

```
V(3) % elemento in terza posizione nel vettore
```

Matrici

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

M(r, c)

Es.

```
M(2,3) % elemento in seconda riga e terza colonna
```

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.

Es.

```
M = [1,2,3;4,5,6;7,8,9] % matrice 3x3
M(5,5) = 10 % amplia la matrice fino ad avere 5 righe e 5 colonne e aggiunge l'elemento
```

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

Es.

```
M[:,2] % prende tutte le righe ma solo la seconda colonna
M[3,:] % prende solo la terza riga con tutte le colonne
```

Stringhe

Per visualizzare una stringa di testo si possono utilizzare i comandi fprintf("") o disp("").

\t inserire una tabulazione orizzontale, \n crea una nuova riga verticale.

disp("") va a capo in automatico.

Per eseguire uno script .m salvato posso scrivere nalla 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.

Es.

```
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.

Es.

```
function [y1, y2, …, yn] = nomefun(x1, x2, …, xn)
function [area] = area_cerchio(raggio)
% calcola area di un cerchio, il cui raggio è 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 contenute i parametri formali definiti, utilizzati nei calcoli interni della function stessa. Vengon, 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.

Es. n = input("Inserire un valore per n");

L’utilizzo degli script .m al posto della Command Window, serve affinchè si possa richiamare una sequenza di istruzioni in una solta 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 doverlo aprire.

Es.

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 è: %.2f mq.\n", area); % stampa area
fprintf("Il perimetro dell'aula è: %.2f m\n", perimetro); % stampa perimetro
fprintf("La diagonale dell'aula è: %.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.

Es.

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 è: %.2f\n", somma); % stampa somma
fprintf("Il prodotto è: %.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 variabile = 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.

Es.

```
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 è 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 (metà 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.

Es.

```
% 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 è 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 → Dato discretizzato

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 intesità 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:

```
immagine=imread("immagine.jpg")
```

L’ultimo numero è il subpixel in basso a destra.

Per visualizzare la dimensione (shape) dell’immagine, il comando è:

```
[M,N,D]=size(immagine)
```

All’interno di M, N, D vengono memorizzati, rispettivamente, i valori relativi al numero di righe, colonne e layer dell’immagine.

È possibile visualizzare come viene codificata l’immagine dalla Workspace, in base al formato mostrato (es. uint8).

Comando subplot

Il comando subplot in MATLAB è utilizzato per creare una griglia di sub-figure all'interno di una finestra grafica, permettendo di visualizzare più grafici o immagini in una sola finestra. In pratica, suddivide l'area di visualizzazione in una griglia specificata da righe e colonne, e permette di generare e posizionare diversi grafici all'interno di questa griglia.

La sintassi è:

subplot(m, n, p)

Dove:

- m è il numero di righe della griglia di sub-figure.
- n è il numero di colonne della griglia di sub-figure.
- p è l'indice della sub-figure dove verrà posizionato il grafico o l'immagine.

Gestione dei canali RGB di un’immagine

```
% 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);
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 esclusiva- mente basandosi sul sistema avionico Altri comandi di volo invece si avvalgono delle fibre.

Altri comandi di volo, invece, si avvalgono delle ottiche come mezzo di trasmissione, portando ad un sistema chiamato fly by light, il quale vantaggio dell'eliminazione delle interferenze elettromagnetiche sugli impulsi dei comandi di volo, che possono purtroppo verificarsi a causa della presenza dei vari impianti elettrici 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 visualizzassero 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 giro- scopi 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.

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 dei gas di scarico (EGT)
airspeed = uiaeroairspeed; % Indicatore della velocità dell'aria
altimeter = uiaeroaltimeter; % Altimetro per la quota di volo
```

Gli output visuali dei comandi precedenti sono:

Figura 1: Orizzonte artificiale

Figura 2: Temperatura dei gas di scarico

Figura 3: Velocità dell'aria

Figura 4: Altimetro

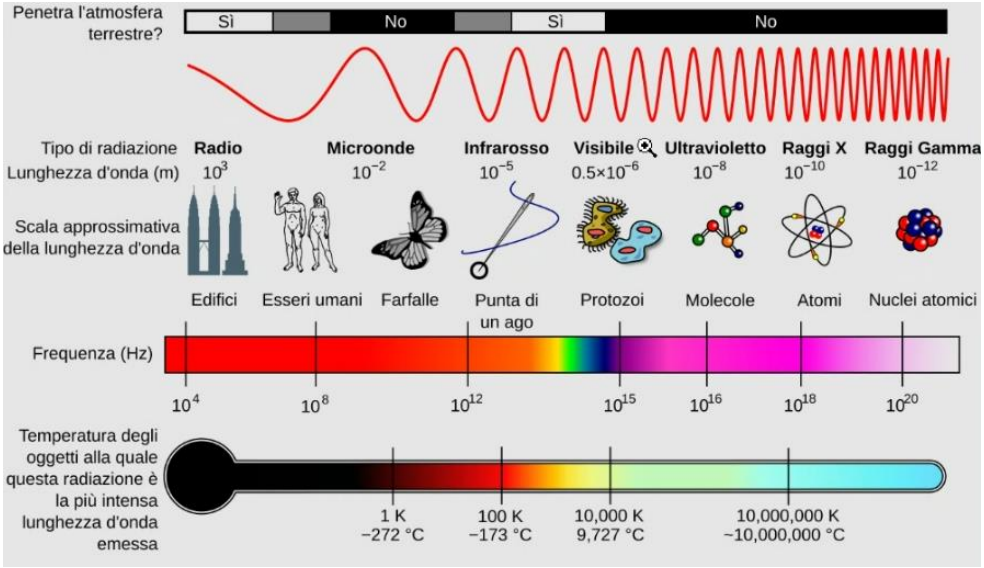
Il codice seguente simula lo spostamento delle lancette in base al passo (sensibilità) impostato:

```
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) è il bank angle (roll), varia da -40 a 40 gradi
    % - Il secondo valore (5+i) è il pitch angle, varia da 3 a 7 gradi
    artificialhorizon.Value = [(20*i) (5+i)];
end
```

Onde elettromagnetiche e cenni di telecomunicazioni



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 e 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 20.169 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 299 792 458 m/s, ma per semplicità, il valore è approssimato a 300 000 000 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
```

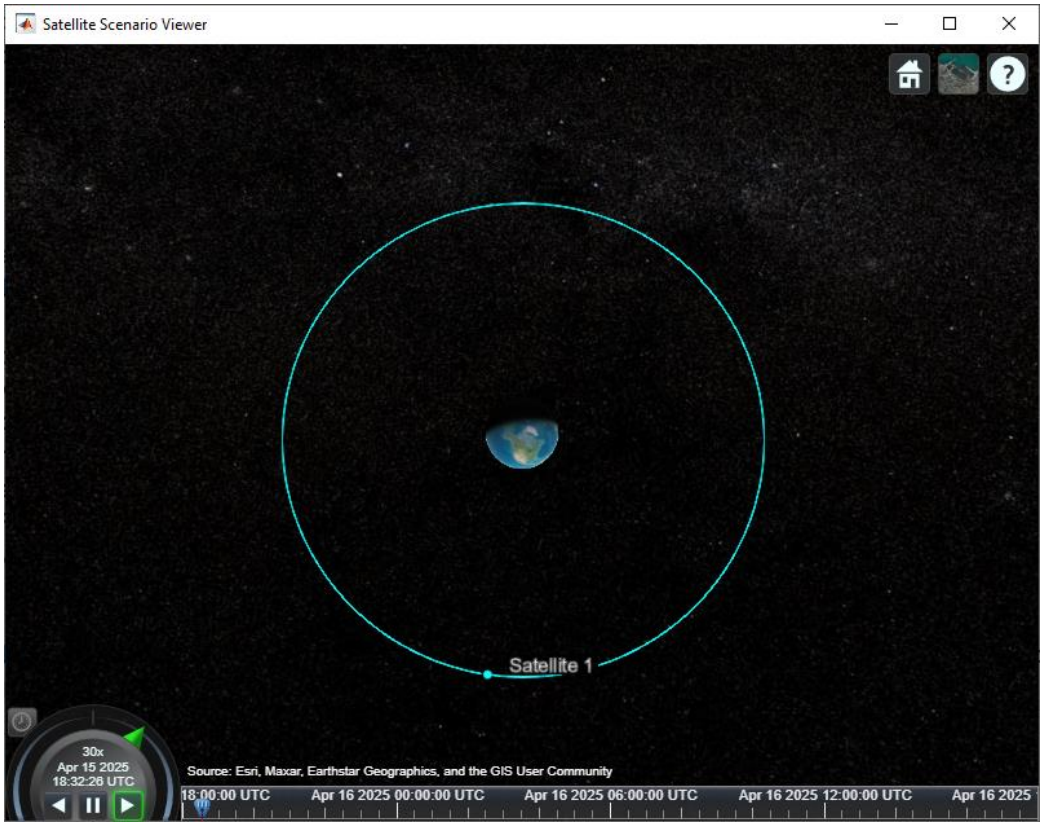
```
semiMajorAxis = [42000000; 43000000; 44000000; 45000000; 46000000]; % Semiasse maggiore in metri
eccentricity = [0.02; 0.01; 0.01; 0.01; 0.1]; % Eccentricità dell'orbita
inclination = [0; 0; 0; 0; 0]; % Inclinazione in gradi (0° → orbite equatoriali)
rightAscensionOfAscendingNode = [0; 10; 20; 10; 10]; % Longitudine del nodo ascendente (gradi)
argumentOfPeriapsis = [0; 10; 10; 10; 10]; % Argomento del pericentro (gradi)
trueAnomaly = [0; 10; 10; 10; 10]; % Anomalia vera iniziale (gradi)

% Aggiunta dei satelliti allo scenario con i parametri specificati
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);

% Visualizzazione dei satelliti e delle loro tracce a terra per un periodo di 2 ore (7200s)
show(sat) % Mostra i satelliti in orbita
groundTrack(sat, LeadTime=7200); % Visualizza le tracce a terra per i prossimi 2 ore

% Avvio della simulazione con velocità di riproduzione 30x rispetto al tempo reale
play(sc, PlaybackSpeedMultiplier=30);
```

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 3D 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;

% Eccentricità 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 aggiungerà 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 più veloce del tempo
reale)
play(sc, PlaybackSpeedMultiplier=30);
```



Costellazione GPS - Modellazione e Visualizzazione 3D

Inserire breve descrizione

In questo codice, sperimentiamo come il 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, velocità, 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 (visibilità) 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);

% ------------------------------------------------------------
% MODALITÀ 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, che 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:

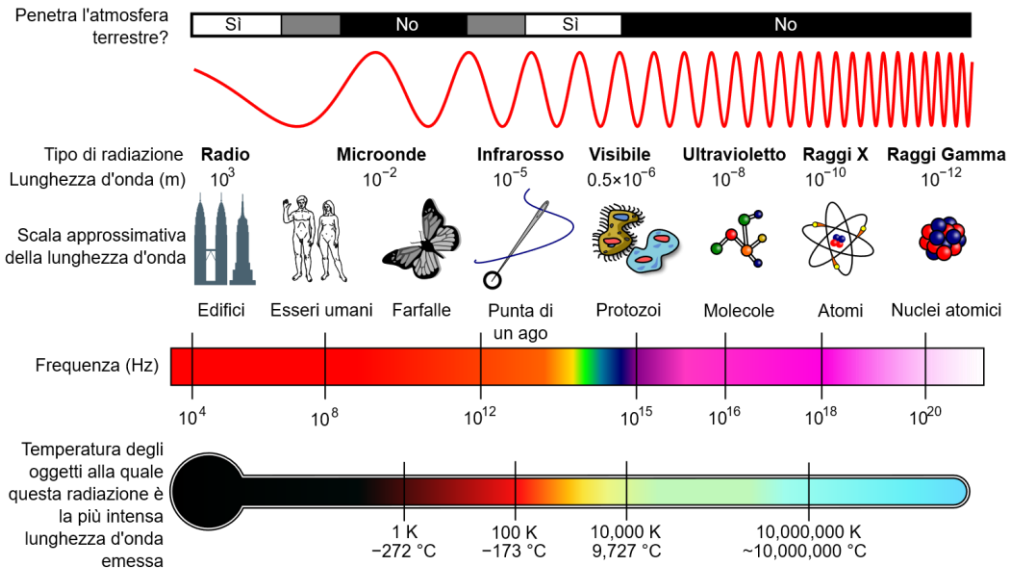
- Cross-range: che si riferisce alla direzione del volo della piattaforma mobile.
- Cross-track: che indica la direzione del puntamento dell'antenna radar.

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 (la fisica delle onde elettromagnetiche)



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

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

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 oggetti 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: velocità 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.LinearFMWaveform('SampleRate', fs, ...
    'PulseWidth', tpd, 'PRF', prf, 'SweepBandwidth', bw);

% Parametri del volo radar (velocità e durata)
speed = 100; % m/s
flightDuration = 4; % secondi

% Configurazione della piattaforma radar (posizione iniziale e velocità)
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, ...
```

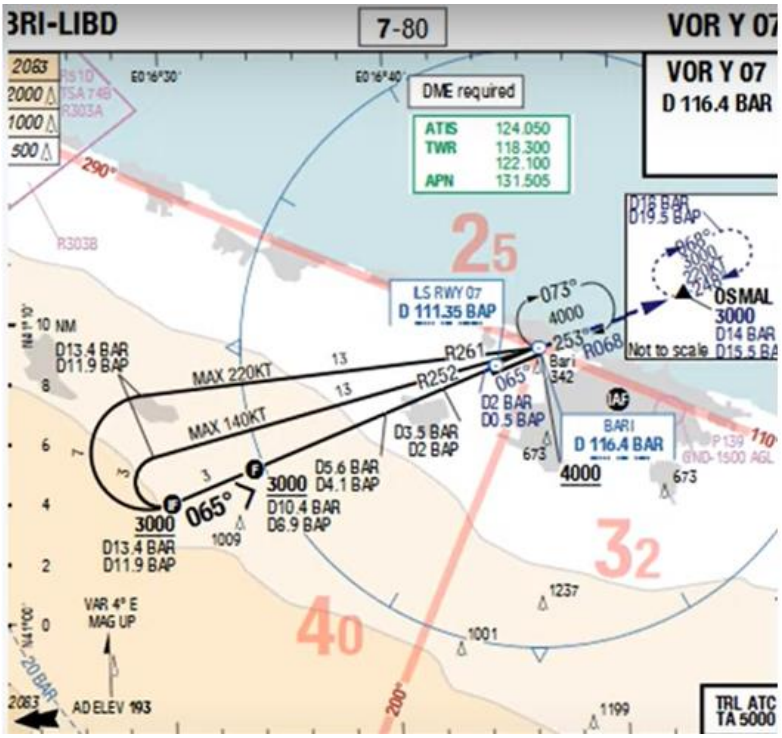
```
    'TwoWayPropagation', true);

% 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 velocità)
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 ii = 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(:,ii) = 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');
```

VHF Omnidirectional Range (VOR)

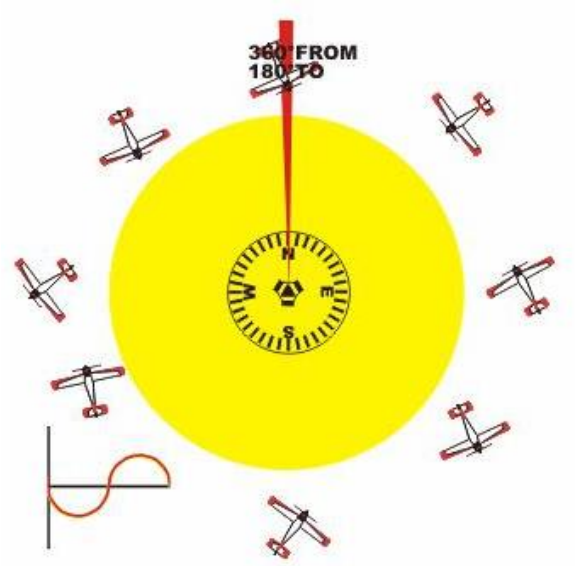
Il VOR è un sistema di radionavigazione terrestre che consente agli aeromobili di determinare la propria posizione e rotta in base a segnali radio emessi da una stazione a terra. solitamente onde radio nella banda VHF (108.0 – 117.95 MHz).



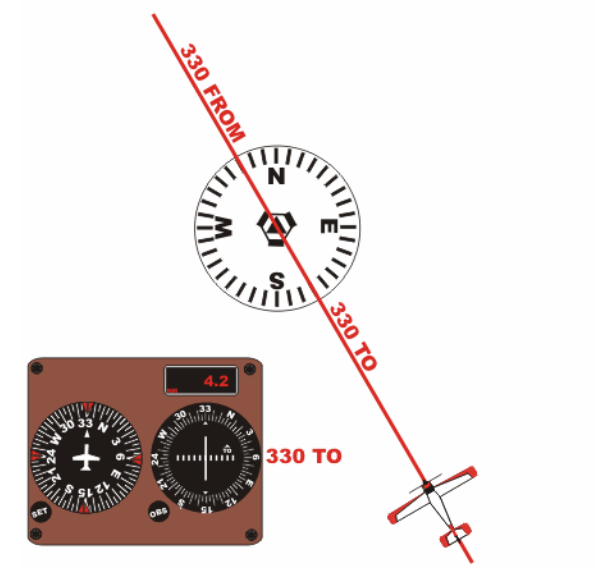
Carta di approdo all'Aeroporto di Bari

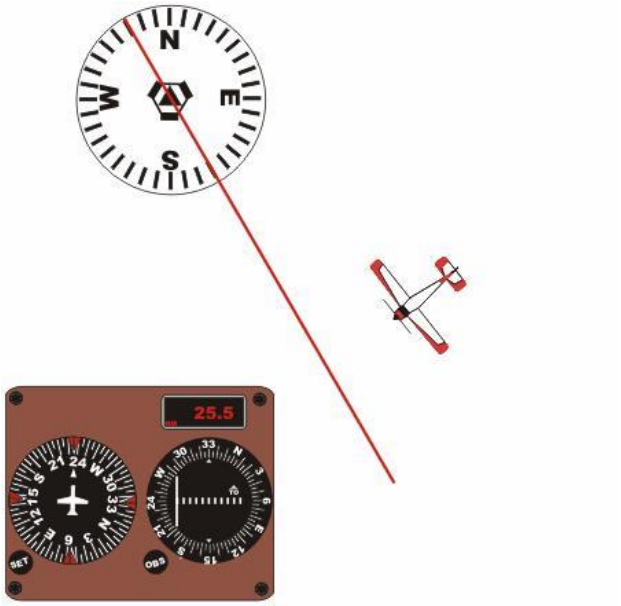
Nella precedente mappa si nota il riferimento al sistema VOR (VHF Omnidirectional Range).

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

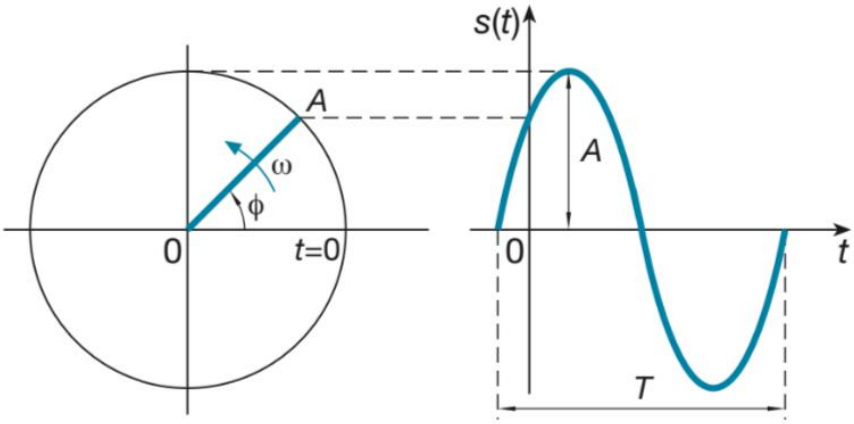


Un segnale sinusoidale è costituito dalle seguenti componenti: $s(t)=A\ sen(2\pi ft + \phi)$.


330 FROM
N
E
S
W
330 TO
4.2
330 TO
SET
OBS


N
E
S
W
25.5
SET
OBS

Il sistema VOR, insieme al sistema DME, è un ottimo e collaudato metodo per la radionavigazione.



Segnale sinusoidale ottenuto dalla proiezione sull'asse delle ordinate s(t) di un vettore ruotante con velocità angolare ω e ampiezza A.

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 sinstesi, l'apparato a bordo mostrerà la posizione dell'aeromobile rispetto al radiale (rotta selezionata), ovvero, se il se 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.

Utilizzare Tracking Scenario Designer
Cosa è VOR e cose sono le carte VOR

Traccia elaborato: Analizzare immagine di una pista ed enfatizzare un certo canale (es. rosso) se superiore a una certa soglia prestabilita (anche con combinazioni logiche di soglie) e abbassare gli altri canali.

Ricerca su lunghezze d'onda, come funziona l'acquisizione, codifica RGB e processamento immagini digitali, pixel e subpixel.

Gestire tutto con un main che acquisisca le immagini e richiami file function esterne.

Viene chiesto input all'utente qualche filtro usare se r,g,b,grayscale,riquadri e in base alla scelta viene richiamata la funzione specifica.

Commentare tutto per bene e fare introduzione e parte teorica con immagini e testo.

In appendice mettere appunti delle lezioni.

CITARE SEMPRE FONTI, ANCHE DI IMMAGINI

RIPORTARE CODICE ELABORATO

Eccentricità, orbite equatoriali, parametri kepleriani, modifiche fatte al codice, satelliti 3d e simulazione, tutto ben commentati.

INSERIRE SEMPRE DELLE CONCLUSIONI NELL'ELABORATO
